2023

# WiFi Sensing at the Edge Towards Scalable On-Device Wireless Sensing Systems

Steven M. Hernandez
*Virginia Commonwealth University*

# WIFI SENSING AT THE EDGE TOWARDS SCALABLE ON-DEVICE WIRELESS SENSING SYSTEMS

A Dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at Virginia Commonwealth University.

by

STEVEN M. HERNANDEZ

Doctorate in Computer Science - 2018-2023

Director: Dr. Eyuphan Bulut,

Associate Professor, Department of Computer Science

Virginia Commonwealth University

Richmond, Virginia

May, 2023

# Acknowledgements

I would like to begin by thanking my advisor, Dr. Eyuphan Bulut who has been a major influence, and guiding force throughout my doctoral studies. Additionally, I would like to thank, Dr. Tamer Nadeem, Dr. Changqing Luo, Dr. Ruixin Niu, and Dr. Muhammad Shahzad for serving on my committee. I would also like to thank my wife for her constant words of wisdom, and my family for all of their hard work. Finally, I would like to thank the many other people in my life whose influence has helped form my path up to this point.

## TABLE OF CONTENTS

Chapter Page

# List of Algorithms

## LIST OF TABLES

Table Page

# LIST OF FIGURES

# Abstract

WIFI SENSING AT THE EDGE TOWARDS SCALABLE ON-DEVICE WIRELESS SENSING SYSTEMS

By Steven M. Hernandez

A Dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at Virginia Commonwealth University.

Virginia Commonwealth University, 2023.

Director: Dr. Eyuphan Bulut,
Associate Professor, Department of Computer Science

WiFi sensing offers a powerful method for tracking physical activities using the radio-frequency signals already found throughout our homes and offices. This novel sensing modality offers continuous and non-intrusive activity tracking since sensing can be performed (i) without requiring wearable sensors, (ii) outside the line-of-sight, and even (iii) through the wall. Furthermore, WiFi has become a ubiquitous technology in our computers, our smartphones, and even in low-cost Internet of Things devices. In this work, we consider how the ubiquity of these low-cost WiFi devices offer an unparalleled opportunity for improving the scalability of wireless sensing systems. Thus far, WiFi sensing research assumes costly offline computing resources and hardware for training machine learning models and for performing model inference. To improve the scalability of WiFi sensing systems, this dissertation introduces techniques for improving machine learning at the edge by thoroughly surveying and evaluating signal preprocessing and edge machine learning techniques. Additionally, we introduce the use of federated learning for collaboratively training machine learning models with WiFi data only available on edge devices. We then consider privacy and security concerns of WiFi sensing by demonstrating possible adversarial surveillance attacks. To combat these attacks, we propose a method for leveraging spatially distributed antennas to prevent eavesdroppers from performing adversarial surveillance while still enabling and even improving the sensing capabilities of allowed WiFi sensing devices within our environments. The overall goal throughout this work is to demonstrate that WiFi sensing can become a ubiquitous and secure sensing option through the use of on-device computation on low-cost edge devices.

# CHAPTER 1

# INTRODUCTION

Over the years, wireless devices such as our smartphones, laptops, and routers have risen in prominence throughout our homes and offices. The radio-frequency (RF) signals emitted by these devices not only allow us to easily communicate but may also reveal a surprising amount of personal information without our knowledge. This is because radio frequency technologies transmit invisible radio-signals which travel through the air and reflect off of physical objects such as walls, ceiling, furniture and most importantly, our own bodies. RF receivers can capture amplitude and phase variations in the signals caused by physical movements and thus, may be used to passively observe our activities. Throughout this dissertation, we consider the use of everyday WiFi signals to achieve WiFi sensing with the goal of identifying both positive aspects (i.e., tracking health and safety) as well as negative aspects (i.e., illegal surveillance). Additionally, this work aims to improve the scalability and allow for more realistic WiFi sensing system deployment through the use of low cost edge devices which has not been considered in the previous research literature. This dissertation focuses on two specific topic-areas:

1. *WiFi Sensing*: Understanding the capabilities of WiFi sensing and identifying new novel use cases as well as the associated benefits and risks.

2. *Edge Learning*: Identifying methods for performing WiFi sensing (signal pre-processing, machine learning inference, machine learning personalization, collaborative intelligence) completely on the edge.

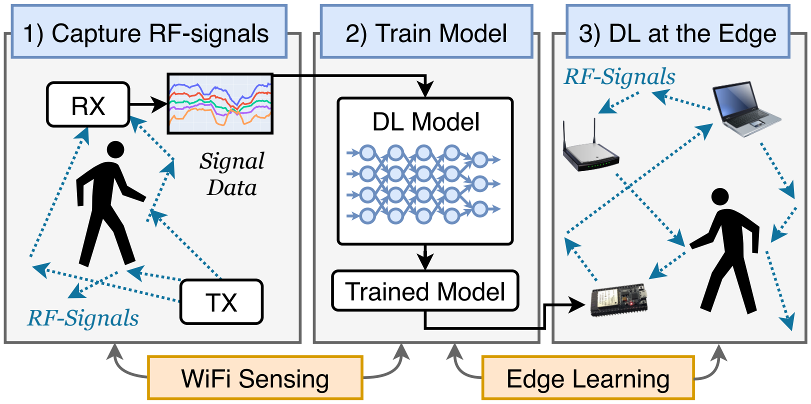


Fig. 1. Illustration of research topic-areas discussed throughout this dissertation.

Fig. 1 illustrates a simplified system diagram demonstrating these topics.

### 1.1 Benefits of WiFi Sensing

WiFi sensing can be used in the place of common sensor-based systems especially in cases where sensors must be attached to the body of a participant (i.e., wearables). While wearable sensors do offer high quality specialized sensing data, they are plagued with a few issues which can be overcome through the use of WiFi sensing. First, wearable sensor may not be worn at all times. As an example, smart watches may contain an accelerometer which can help identify and react to sudden hazardous falls which can be highly important for elderly populations. However, the smart watch must be removed to charge over night or must be removed during fall-prone activities such as bathing and showering. WiFi sensing on the other hand will be able to identify these fall activities passively without requiring devices attached to the body. Second, wearable sensors can be cumbersome and intrusive to wear. In some cases, these sensing systems reduce the mobility of the person meaning that the sensor is only used for short period of times during the day, thus reducing the amount of time which they are used. For example, the *NeuLog Respiration Belt* [1] is commonly used to achieve respiration tracking and to identify irregular breathing patterns, however the device is cumbersome and is typically powered from a wall outlet, thus precluding the user from moving naturally throughout their environment. Third, sensors are typically designed to track a single individual at a time which further increases the cost of deploying sensor-based systems in multi-person conditions. With WiFi sensing, the sensing occurs regardless of the number of people in the environment, albeit with additional complexity required for the sensing model.

Another alternative to wearable sensors is the use of camera-based sensing, however these are also plagued with issues which can be solved through WiFi sensing. First, camera-based systems are perceived to be much more privacy invasive, especially in our homes because images and videos can contain unexpected personal information which we would not want to get into the wrong hands. WiFi sensing relies on per-environment calibrations which ensure that the signals can only be used to track a set of allowed actions. Second, camera-based systems can only track activities that occur in a single line-of-sight (LOS) in front of the camera sensor. WiFi signals on the other hand are transmitted omnidirectionally allowing for sensing to be performed in both LOS and non-LOS (NLOS) conditions as well as in through-wall scenarios. Third, lighting conditions reduce the capabilities of the camera-based systems while WiFi sensing can be performed independent of any lighting sources which allows the same hardware and model to be used whether it is daytime or nighttime.

### 1.2 Benefits of Machine Learning Inference at the Edge

In the past, machine learning use was limited to high-powered server architectures within the datacenter. However, there is high latency involved with communicating raw data and predictions back and forth between the edge and the datacenter. Additionally, as more devices and sensors are added to a system, the higher the bottleneck is for communicating with so many client devices. As such, by moving tasks like signal preprocessing and machine learning inference to the edge, we can offset these issues by allowing the devices to work standalone irrespective of any datacenter. This also allows for improvements in privacy because the raw data no longer needs to leave the environment where it is produced. Finally, this ensures that devices can continue to function even when the network connection is limited or unavailable.

## 1.3 Benefits of Machine Learning Model Personalization

Most commonly, when machine learning is performed at the edge, a single model is pretrained at a central server and then deployed across all devices. However, while the model may be generalized to the data it is trained on, the model will not be personalized to the given environment which it is deployed in. Allowing models to be fine-tuned at the edge will help alleviate these issues by allowing for additional context and location-specific training using data captured at the given environment. For example, in [2], the authors demonstrate that speech recognition models can be improved by personalizing the model based on speaker-independent attributes such as accents. Furthermore, each environment may have unique insights which will be able to improve the generalizability of the model for other new locations. In [3], federated learning is used to collaboratively train a smartphone keyboard auto-suggestion system by using the vocabulary used by many different users. Overall, training machine learning models at the edge can improve model generalizability as well as personalize the model for the specific conditions, while also increasing the user privacy by retaining the raw data local to the edge devices rather than sharing it to a central server. With wireless sensing platforms, the physical environment will be highly unique depending on the size of the room, the furniture within the room, the placement of the radio hardware, and other physical properties of the environment. As such, each device will witness certain unique environment-specific features which will need to be considered by the machine learning system. Performing model personalization and model calibration schemes at the edge are important capabilities which will ensure environment adaptability and thus accurate model predictions.

## 1.4 Industry Interest

Considerable industrial interest has appeared for both wireless sensing as well as in enabling machine learning at the edge. Interest in wireless sensing has grown over the past years with the availability of consumer products such as Google Soli [4], Linksys Aware WiFi sensing mesh [5], Emerald Innovations [6], and Origin Wireless [7]. Edge-based machine learning also sees considerable interest in industry through the development of specialized neural processing hardware such as Google Edge TPUs [8] and Kendryte K210 [9] which is available through consumer development boards such as the Maixduino [10]. On-device edge learning has yet to be embraced by industry which leaves room for new industry collaboration opportunities to be formed in the future.

## 1.5 Dissertation Organization

The remainder of this dissertation is organized as follows:

1. We begin by reviewing existing literature related to WiFi sensing and edge machine learning in Chapter 2.

2. Next, we review the tool we developed for capturing and annotating the WiFi signal data for all experiments discussed throughout this dissertation in Chapter 3.

3. Chapter 4 follows with a survey of signal processing and machine learning tech-

niques used for WiFi sensing. These techniques are reviewed to determine their feasibility for on-device edge prediction making. Additionally, the techniques are evaluated on three baseline WiFi sensing tasks from small scale hand gesture recognition, to medium scale human activity detection, up to large scale localization and activity tracking.

4. After this, Chapter 5 considers methods for collaboratively training machine learning models across multiple edge devices through federated learning. Through this, it is demonstrated that collaboratively trained federated models allow for reduced personalization training for newly deployed devices. This allows for greater scalability for WiFi sensing systems.

5. Next, Chapter 6 considers adversarial attacks which are possible due to the fact that the wireless signals are designed to pass through the wall even into private areas.

6. To defend against this, in Chapter 7 we propose an anti-eavesdropper method which leverages spatially distributed antennas to prevent eavesdropper WiFi sensing devices from performing accurate sensing while still allowing and even improving the sensing accuracy of approved WiFi sensing devices.

7. Finally, we discuss concluding remarks in Chapter 8.

# CHAPTER 2

# LITERATURE REVIEW

## 2.1 Applications of WiFi Sensing

Since its inception, WiFi sensing has been leveraged for a number of novel sensing tasks. Through our survey of the existing WiFi sensing literature, we find that localization and human activity recognition are the two most common tasks for WiFi sensing followed by hand gesture recognition, crowd counting, occupancy detection and health tracking such as respiration sensing. Table 1 shows a breakdown of these applications as well as a few examples of related sub-topics.

### 2.1.1 Localization

The most common WiFi sensing research task is localization where the physical location of a target can be tracked throughout an environment using ambient WiFi signals. In the traditional WiFi based localization approaches, the target being tracked must have some transmitting or receiving hardware on their body such as in [11] where some set of static WiFi devices are placed at known anchor locations within the environment. Similarly, relative positioning [23, 24] allows the use of non-static anchor devices to achieve positioning relative to objects in an environment rather than absolute coordinates. UAVs with on-board WiFi antennas have also been used as both CSI transmitters and receivers for device-to-device localization [25, 26]. More recent works track the location of human targets in a device-free manner (i.e., without requiring a WiFi device to be placed on the individual). For example, CS-Dict [27] and the work of Zhou et al. [28] use WiFi signal data to build a database

Table 1. Most common WiFi sensing tasks in our literature survey along with some examples. ($N = 658$)

| Applications | Sub-Topics | Example |
|---|---|---|
| Localization (14.0%) | Device-Based | [11] |
| | Device-Free | [12] |
| Human Activity Recognition (13.1%) | Exercise | [13] |
| | Daily Activity Tracking | [14] |
| Gesture Recognition (9.4%) | Hand Movement | [15] |
| | Finger Movements | [16] |
| Health (7.9%) | Respiration | [1] |
| | Heart Rate | [17] |
| | Sleep | [18] |
| Crowd Counting (4.9%) | Indoor | [19] |
| | Outdoor (Pedestrians) | [20] |
| Occupancy (3.3%) | Security | [21] |
| | Context-Aware Applications | [22] |

of environmental signal fingerprints when the target is standing at different physical positions throughout an indoor environment. However, in localization tasks, physical changes in the environment may reduce the accuracy of a WiFi sensing system due to changes in the multipath environment. To account for this, some techniques have appeared such as Domain Adaption (DA) [12, 29] and Transfer Learning (TL) [30].

### 2.1.2 Human Activity Recognition

The next most common use for WiFi sensing is to perform Human Activity Recognition (HAR). Similar to the localization task where signals propagating through the environment may bounce off of a human target as the signal propagates from a transmitter to a receiver, we can also capture even finer detail about physical action being performed by the target if we watch signal variations over time. As such, some of the most common actions recognized are stationary activities like sitting or standing as well as mobile activities like walking and running [31, 32]. This can be useful to judge the occupancy of a room for applications such as smart home environments [33]. Training a model for all possible actions that a human target can perform would be infeasible due to the sheer number of possible states. Using natural language semantics, it has been shown in [34] that a model can be trained on a single action; say walking, and then used to recognize other unseen actions; for example running, due to the semantic relation between the two actions (i.e., running is like walking at a higher pace). Fall detection [35, 31, 36] can help ensure the safety of elderly or sick individuals without sacrificing their privacy within their own home as would be the case with camera based systems. Low resistance calisthenic exercise tracking provides another set of novel physical actions for device-free WiFi sensing [37, 38, 13]. WiLay [39] uses a layered approach which combines both device-free localization with HAR where initially, a model recognizes the approximate physical location of the target in the environment and then determines the human activity through the use of a model trained specifically on that target location.

### 2.1.3 Gesture Recognition

Many studies look to recognize human movements at an even finer detail such as at the hand and finger level through gesture recognition. Recognizing such fine-grained gestures can allow for novel gesture-based interactions with smart home environments [40, 15] and in-vehicle control [41]. Gestures performed by individuals can reveal unique characteristics which can then be used to authenticate valid users for a given system as shown in [15]. Finger position recognition can be used as a novel method for text input into a computer such as through recognizing the number of fingers held up by a target [42, 43], as well as through sign language and finger spelling [16, 44]. Similarly, tracking finger movement through the air can allow in-the-air handwriting recognition [45, 46]. WiFi sensing can also be used to reveal information that should not be publicly available such as passwords through keystroke detection [47, 48].

### 2.1.4 Indoor Crowd Counting and Occupancy Detection

Understanding the movement of people through indoor environments can be useful in gathering real-world customer mobility analytics as suggested in [49], for monitoring secure locations [50], as well as for detecting people indoors during rescue missions [51]. Typically, crowd counting is performed when the targets are constantly moving through an environment such as in [19]. Stationary crowd sizes can be estimated through WiFi signals by recognizing the small fidgeting movements made randomly by members in the crowd [52]. Understanding the movement of a crowd through an indoor environment can also improve safety during emergency building evacuations by tracking the number of people exiting the building as well as the number of people still within the building [53]. However, adversaries can also leverage this same ambient WiFi signals to track individuals in non-public environments which reduces privacy [54].

### 2.1.5 Health Sensing

In private residences it can be useful to monitor health related activities at all times. However, wearable sensors can be cumbersome to the user and camera based systems are too privacy invasive. WiFi sensing has gained traction in health monitoring tasks because it is both device-free and less invasive. Specifically, respiration tracking [1, 55] is one of the most common health related WiFi sensing tasks, which can be achieved by recognizing peaks in signal variation over time. Tracking the breathing patterns of multiple people in a given environment has been shown to be possible through Blind Source Separation (BSS) [56]. Moreover, tracking respiration with WiFi signals can help reveal irregular breathing patterns such as apnea or tachypnea [57]. Similarly, monitoring people during sleep periods can help detect unhealthy sleep actions such as rhythmic movement disorders [18] as well as nocturnal seizures [58]. Even more fine-grained sensing has been performed with WiFi sensing to track heart rates of individuals which can help identify variability in heart rhythms [17, 59]. However, we find that the transmitter and receiver typically must be placed very close to the chest which makes the setup impractical in real-world environments.

### 2.1.6 Other Novel Use Cases of WiFi Sensing

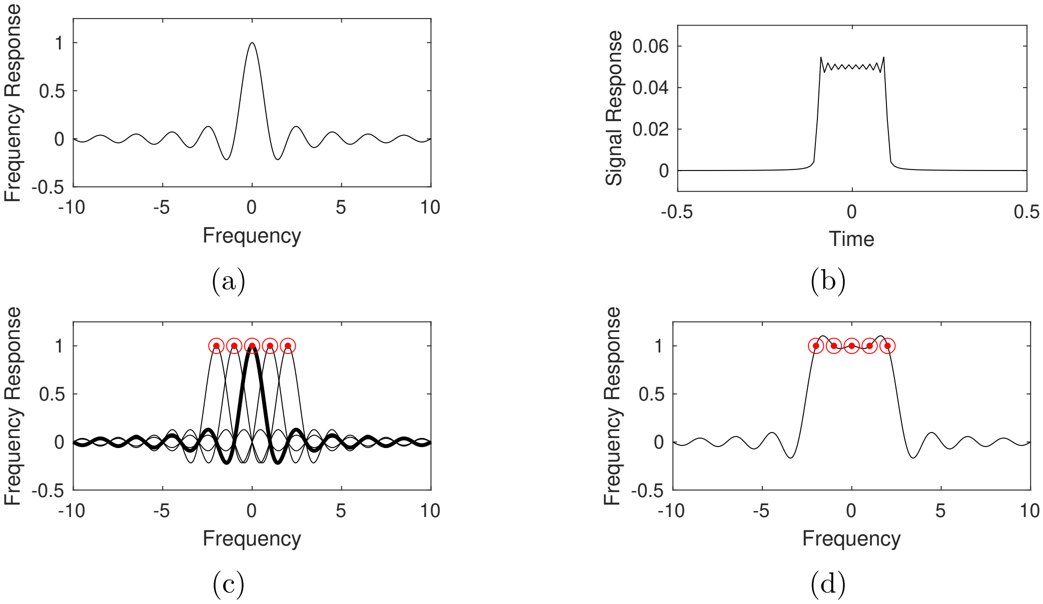
While the previously discussed applications for WiFi sensing have numerous associated research studies, there are a few use cases for WiFi sensing which have only appeared in a small number of research studies. For example, EmoSense [60] uses WiFi sensing to predict the emotion of a human subject based on the physical movements that the subject performs thus providing useful measurements for tracking the mental health of individuals. WiEat [61] leverages human movements to recognize the eating behavior of the individual to aid in health and dietary tracking. WiFi sensing has also been applied to track food and agricultural properties such as to detect fruit ripeness [62] as well as to track the moisture levels of wheat [63] to ensure that the moisture level does not cross a critical threshold which may result in crop spoilage. The moisture levels of soil [64] has also been tracked with WiFi sensing to ensure adequate water coverage while reducing overwatering for agricultural farmland. Liquid level tracking [65] as well as liquid identification [66] have been achieved with WiFi sensing through the measurement of dielectric properties of the liquid as well as the resonance frequency response of the liquid due to vibration. Vibration detection [67] through WiFi sensing has also been used for detecting faults in factory equipment.

## 2.2 Preliminary Theory on Wireless Sensing

As an introduction to the theory describing how WiFi sensing is capable of achieving these varying sensing tasks, we begin by introducing both orthogonal frequency-division multiplexing (OFDM) and channel state information (CSI). These two topics form the core from which most recent WiFi sensing innovations have arisen.

### 2.2.1 Orthogonal Frequency-Division Multiplexing

Orthogonal frequency-division multiplexing is a modulation scheme used in wireless communication systems such as 802.11n which encodes data streams into multiple overlapping subcarrier frequencies. OFDM signals are modelled in the frequency domain as $sinc(f) = \frac{sin(f)}{f}$ as presented in Fig 2a. This frequency-domain representation can be transformed into the time-domain through the Inverse Fast Fourier Transform (IFFT) to produce an approximate *rectangle* function as shown in Fig. 2b. The *sinc*



(a)

(b)

(c)

(d)

Fig. 2. Representations of subcarrier symbol encodings in OFDM systems. (a) Single subcarrier modelled as a sinc function in the frequency-domain. (b) Same subcarrier in the time-domain after applying IFFT. (c) By selecting orthogonal subcarrier frequencies, the peak of the sinc function for each subcarrier corresponds to a zero valued frequency response from all other subcarriers. (d) After summation of all subcarriers in the frequency domain, the peak values marked in red are retained as a result of this orthogonality.

function is selected because it allows each subcarrier to be placed orthogonally to one another in the frequency domain as shown in Fig. 2c where five *sinc* functions are placed such that the peak center subcarrier frequency (denoted in red) is located at a frequency where all other *sinc* functions are zero. As a result of the orthogonal placement, when taking the summation of all five selected subcarriers as shown in Fig. 2d, the peaks (denoted in red) are retained. Each subcarrier represents a single OFDM symbol transmitted over a time period of $3.2\mu s$ with $0.8\mu s$ guard period [68] and can be modulated through methods such as binary phase-shift keying (BPSK), quadrature phase-shift keying (QPSK) or quadrature amplitude modulation (QAM) depending on the desired data transmitted per OFDM symbol. Each OFDM symbol encodes binary data as a complex number through I/Q samples where $I$ is the *in-phase* component representing the real part and $Q$ is the *quadrature* component representing the imaginary part. As an example, 16-QAM is able to represent 4-bits of information with a single complex number [69].

Frequency selective fading may occur due to constructive and destructive interference caused by signals propagating over multiple paths of varying distances. Note that, because each subcarrier has slightly different frequency, not all subcarriers will witness the same constructive or destructive interference. This is an important feature of OFDM to ensure that data can still be transmitted reliably. Even so, it is important for the receiver to recognize these variations across a single OFDM symbol. As such, some subcarriers act as pilot subcarriers where the I/Q encoded symbols are already known by both the transmitter and the receiver. Through these pilot subcarriers, OFDM can correct for variations in the received signal in different subcarriers through subcarrier equalization [69].

Given a standard WiFi channel with a bandwidth of 20MHz, 64 subcarriers are created and centered around some carrier frequency such that each subcarrier represents 312.5kHz of spectrum. For example, WiFi Channel 1 has a center frequency of 2.412GHz and a frequency range from 2.401GHz to 2.423GHz[1]. Subcarriers are indexed based on this center frequency such that subcarrier 0 is the direct-current (DC) subcarrier, subcarriers $-21, -7, 7, 21$ are pilot subcarriers, all other subcarriers between $-26$ and 26 contain actual encoded data while the remaining subcarriers are null guard band subcarriers as illustrated in Fig. 3.

[1] The frequency range for WiFi channels are actually 22MHz due to older versions of the 802.11 standard. OFDM only considers a reduced bandwidth of 20 MHz.

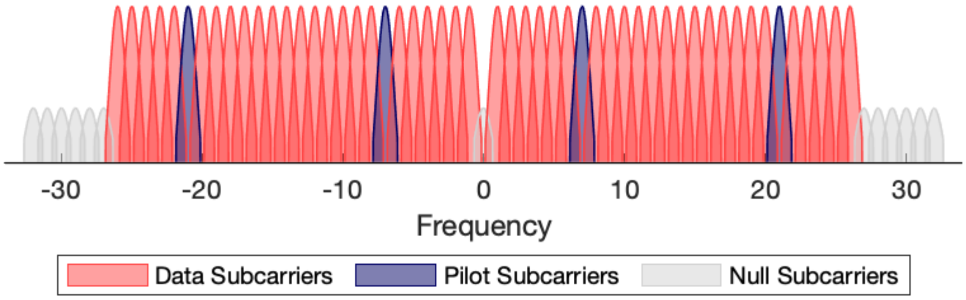


Fig. 3. Layout of subcarrier types in the WiFi frequency domain.

### 2.2.2 Channel State Information

Channel state information is the metric used in OFDM systems for describing amplitude and phase variations across subcarrier frequencies as wireless signals are transmitted between a transmitter and a receiver. Channel estimation is the process used to detect variations across the subcarriers in OFDM systems through the transmission of a set of known shared pilot symbols in a comb-type pilot pattern [70] where the pilot subcarriers remain the same over time. The CSI matrix ($\mathbb{H}$) can then be estimated using:

$$y = \mathbb{H}x + \eta, \tag{2.1}$$

where $y$ is a vector indicating the signal detected at the receiver, $x$ is the signal vector that was transmitted based on the agreed upon pilot symbols and finally $\eta$ is an additive white Gaussian noise vector. $\mathbb{H}$ is a complex matrix containing a complex value for each subcarrier ($i$) representing the Channel Frequency Response (CFR) denoted as $\mathbf{h}_i$ and represented as

$$\mathbf{h}_i = A_i e^{j\phi_i}, \tag{2.2}$$

consisting of both real ($\mathcal{R}(\mathbf{h}_i)$) and imaginary ($\mathcal{I}(\mathbf{h}_i)$) 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:

$$A_i = \sqrt{\mathcal{I}(\mathbf{h}_i)^2 + \mathcal{R}(\mathbf{h}_i)^2} \tag{2.3}$$

$$\phi_i = atan2\big(\mathcal{I}\left(\mathbf{h}_i\right), \mathcal{R}\left(\mathbf{h}_i\right)\big). \tag{2.4}$$

Due to the complexity of any given environment, the signal received is not only a result of a direct line of sight (LOS) transmission, but is also affected by the environmental multipath—the multiple physical paths that the signal travels from transmitter (TX) to receiver (RX). Thus,

$$\mathbf{h}_i = \sum_{m=1}^{N} A_m e^{\frac{-2\pi f_i d_m}{c} + j\phi_m} \tag{2.5}$$

where $A_m$, $\phi_m$ and $d_m$ are the resulting amplitude, phase and distance, respectively, from a given single multipath route where $f_i$ is the frequency for the given subcarrier $i$ and $c$ is the speed of light. Each multipath thus affects the signal through amplitude attenuation caused by the environment as well as time delay and phase shifts caused by the distance of the path. While the multipath lengths are the same across subcarriers, each subcarrier will exhibit different frequency-selective fading due to in-phase or out-of-phase multipath interference. OFDM systems are able to combat this frequency selective fading because each subcarrier has a unique frequency ($f_i$) and as such, whenever some set of subcarriers exhibit destructive fading, the other subcarriers should be free of destructive fading thus allowing communication to continue.

Given the problem of device-free sensing of human targets, two sets of paths can be considered. The first set ($\Omega_s$) represent static paths in an environment. Examples of these static paths would be transmitted signals reflected off of walls unrelated to the human target. The second set ($\Omega_d$) represent the dynamic paths, or those which are affected by the movement of a given target in the environment. Considering these two sets of paths,

$$\mathbf{h}_i = \underbrace{\sum_{m \in \Omega_s} A_m e^{\frac{-2\pi f_i d_m}{c} + j\phi_m}}_{\mathbf{h}_{static}} + \underbrace{\sum_{n \in \Omega_d} A_n e^{\frac{-2\pi f_i d_n}{c} + j\phi_n}}_{\mathbf{h}_{dynamic}}, \tag{2.6}$$

and because $\mathbf{h}_{static}$ is static over time, $\mathbf{h}_{static}$ becomes a constant value which can then be eliminated leaving only $\mathbf{h}_{dynamic}$. This is important, specifically because one of the paths found in this static component is the LOS path between transmitter and receiver. When unobstructed, the LOS path produces a signal which overwhelms other paths because of higher amplitude of the LOS path. Further, by removing $\mathbf{h}_{static}$, the remaining paths are only affected by the actions performed by the human target which can allow predictions to be better resistant to static environmental changes [71].

CSI is represented in the frequency-domain and as such can be converted to the time-domain through the Inverse Fast Fourier Transform (IFFT) by:

$$\mathbf{H}_n = \sum_{m=0}^{N-1} \mathbf{h}_m e^{-j2\pi nm/N}, \tag{2.7}$$

where $\mathbf{H}_n$ is the Channel Impulse Response (CIR) for time $n$. CIR can be transformed back to the frequency domain through the Fast Fourier Transform (FFT)

$$\mathbf{h}_m = \sum_{n=0}^{N-1} \mathbf{H}_n e^{j2\pi nm/N}. \tag{2.8}$$

Some signal processing techniques such as in [40] require filters to be applied to the CIR before converting back to the CFR for further signal processing.

Throughout this work, we collect multiple CSI samples over a time window of size $w$ which can be represented as the $S \times w$ matrix

$$\mathcal{H}[t] = \begin{bmatrix} \mathbf{h}_1[t-w+1] & \mathbf{h}_1[t-w+2] & \dots & \mathbf{h}_1[t] \\ \mathbf{h}_2[t-w+1] & \mathbf{h}_2[t-w+2] & \dots & \mathbf{h}_2[t] \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{h}_S[t-w+1] & \mathbf{h}_S[t-w+2] & \dots & \mathbf{h}_S[t] \end{bmatrix}, \tag{2.9}$$

where $S$ is the number of subcarriers and $w$ is the number of time frames where CSI is collected. After signal preprocessing steps, $\mathcal{H}[t]$ can be passed as the input into a machine learning model to make WiFi sensing predictions.

## 2.3 Alternative Wireless Sensing Modalities

The topics discussed throughout this dissertation focus entirely on wireless sensing through the use of CSI obtained with WiFi-based devices. However, these techniques and observations made through the dissertation are similarly applicable to a number of other wireless sensing technologies. Other wireless sensing modalities include the use of: ultra-wideband radio (UWB) [72, 73, 74], millimeter wave radar (mmWave) [75, 76, 77], received signal strength indicator (RSSI) [78, 79], and more. Compared to these alternative wireless sensing methods, WiFi sensing offers an important improvement in that WiFi signals are already very common in our homes and offices and other indoor environments. Thus, WiFi sensing can be deployed in radio-signal rich environments which allows for a reduction in hardware cost.

## 2.4 Machine Learning at the Edge (TinyML)

Recent pushes towards running machine learning inference at the edge have resulted in several improvements for use cases such as continuous health tracking [80], intelligent adaptive vehicle traffic control [81], and flight control and navigation for

UAVs [82]. Recently, this research field which combines machine learning and embedded systems is referred to as TinyML. Research into TinyML can be split into three categories: deep learning algorithm design, hardware design and applications of TinyML [83]. However, TinyML focuses on performing model inference using models that were pretrained at some more powerful system before being embedded into embedded microcontroller units (MCUs). Notice, we perform a more thorough literature review and investigation into the techniques required when TinyML is applied to the task of WiFi Sensing in Chapter 4.

While model inference is the primary concern of TinyML, there are a few works that do consider methods for training models on low resource embedded MCUs. For example, both TinyOL [83] and TinyFedTL [84] take the approach that a pretrained TinyML model can be personalized on-device at the edge by training a single output layer for the given machine learning model. Specifically, all layers before the final layer are stored on-board and run through inference like a normal TinyML model. The output of this TinyML model is then input into a separate single training layer which can be trained much quicker and with a simpler training algorithm than full backpropagation. TinyTL [85] takes a different approach where each layer is still trained on-board but only a subset of model parameters are trained while the others remain frozen. Specifically, the model weights are frozen and only the biases are trained on-device. This allows training to occur on all of the multiple layers throughout the machine learning model while still ensuring training is performed in a timely manner.

# CHAPTER 3

# IOT EDGE WIFI SENSING TOOLKIT (ESP32-CSI-TOOL)

We introduce several WiFi sensing datasets throughout this dissertation for a diverse range of sensing tasks. However, unlike other research areas where reference datasets are plentiful, only a small number of public WiFi sensing datasets are available for use. In this dissertation specifically, we aim to evaluate experimental conditions which have yet to be recorded or released to the WiFi sensing research community. For example, in this dissertation we evaluate:

1. new embedded WiFi sensing hardware to enabled *edge WiFi sensing*,
2. a newly proposed system which leverages multiple TX antennas to better ensure privacy of WiFi sensing systems,
3. additional novel WiFi sensing applications which have not yet been considered in the research literature.

To aid in the collection and annotation of these novel WiFi sensing datasets, this chapter introduces our Internet of Things (IoT) based WiFi sensing system which consists of the ESP32 MCU shown in Fig. 4 which can be used as both an active CSI RX device, an active CSI TX device, and a passive CSI RX device (PX).

## 3.1 ESP32-CSI-Tool

The ESP32-CSI-Tool[1] can work on a standalone ESP32 MCU and can be easily deployed with a low cost. This provides opportunities to build more practical and easy

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

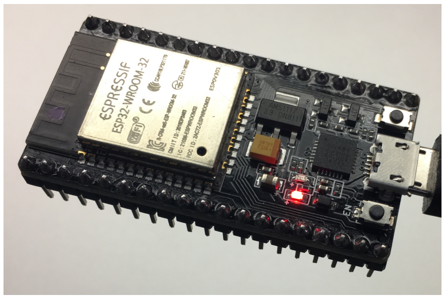


Fig. 4. Close up of an ESP32 microcontroller board.

to maintain WiFi sensing systems especially for large scale systems[2]. The codebase is implemented in C++ using the Espressif IoT Development Framework (ESP-IDF) and consists of three specific applications which run on-board the ESP32 MCU. The first application is an *active access point (AP)* which initializes the on-board WiFi stack to allow other devices to initialize a connection, make requests and receive responses from the AP. The second application is an *active station (STA)* which automatically connects to the AP, then sends requests to the server running on the AP. In general, the AP takes the role of the RX while the STA takes the role of the TX however both applications are able to receive CSI due to the bidirectional communication between AP and STA. 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. The third application is a *passive receiver (PX)* which passively listens for CSI from any nearby devices communicating on a given WiFi channel. The PX is not active and as such does not directly transmit any WiFi signals, thus, attackers may be able to leverage the module to covertly achieve WiFi sensing based surveillance.

[2] Considering existing methods such as [86] which required 10 fully-featured laptops to act as each CSI RXs.

## 3.2 Comparison

Before discussing our tool further, it is important to consider other existing tools used for WiFi sensing research; namely, (i) the *Linux 802.11n CSI Tool* for Intel 5300 Network Interface Cards (NICs), (ii) the *Atheros CSI Tool* for a range of Atheros NICs, (iii) *Nexmon Tool* for Broadcom WiFi chips, and (iv) *Universal Software Radio Peripheral (USRP)* which is a full-fledged software defined radio (SDR). Table 2 shows a comparison between each tool.

Both Intel 5300 and Atheros can be refered to simply as NICs because of their close similarity. The NICs require direct connection to either a laptop or a full desktop computer to function which results in excess hardware costs and increased size[3]. The Nexmon tool on the other hand was shown to work on a Google Nexus smartphone, however the implementation 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. USRP provides full control of the SDR to transmit and receive CSI, however because the USRP is designed as laboratory equipment, it relies on a connection to a host computer. Obviously, neither the NICs nor the USRP can run as standalone device without requiring additional hardware, nor can *Nexmon* without a full-fledged Google Nexus smartphone. However, with our ESP32 CSI tool, the ESP32 can collect and process CSI directly on-board 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 a smaller and is thus a more scalable solution. The cost (i.e., $<$ \$10) for the ESP32 is on par with the cost of a NIC, however a NIC again

[3] While modern laptops are increasingly smaller, they typically are unable to accommodate user-replaceable hardware like NICs. As such, researchers most commonly use large desktop computers as the host for NICs.

Table 2. Comparison of Tools for Collecting CSI.

| | Intel 5300 | Atheros | Nexmon | USRP | ESP32 |
|---|---|---|---|---|---|
| Standalone Operation | NO | NO | YES | NO | YES |
| with Smartphones | NO | NO | YES | NO | YES |
| Size | $> 30cm \times 20cm$ | $> 30cm \times 20cm$ | $> 15cm \times 7.5cm$ | $20cm \times 15cm$ | $5.0cm \times 3.0cm$ |
| Weight | $> 1kg$ | $> 1kg$ | $> 100g$ | $> 1kg$ | $< 10g$ |
| Cost | \$10 + Laptop | \$10 + Laptop | >\$100 | >\$1,000 | <\$10 |
| # Subcarriers | 30 | 56 | 128 | Variable | 64 |
| Resolution (imag./real) | 8 | 11 | 32 | Variable | 8 |
| Implementation Level | Kernel | Kernel | WiFi Chip Firmware | User | User |
| Codebase Size (LOC) | 2M | 2M | 1M, 60K CSI Specific | N/A | 1K |
| RAM | 8GB+ | 8GB+ | 1GB-4GB | 8GB+ | 500KB - 4MB |
| # Antenna | 3 | 3 | 1 | 1+ | 1 |
| TensorFlow | Full | Full | Lite | Full | Lite/Full |

Table 3. Use Cases for ESP32 based CSI collection

| Use case | TX | 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 |

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.

### 3.3 Use Cases

Collecting CSI with the ESP32 is highly versatile in the fact that the ESP32 can collect CSI when acting as both AP and as 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 3 shows the possible use cases that could be achieved with ESP32s in a WiFi sensing scenario. Note that this setup provides full control of both devices and thus we can control CSI sample rate as well as the position of each device within a given space. However, if WiFi transmitting devices already exist in the environment (i.e., a WiFi router or an Android or iOS smartphone), then the ESP32 is also capable of collecting CSI data from these devices too. Additionally, our ESP32 tool is also capable of sniffing ambient WiFi signals and extracting 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 [87] 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.

### 3.4 CSI Sampling Rate

CSI sampling rate refers to the number of CSI frames received by the ESP32 per second. To evaluate this sampling rate, we begin with a single TX set to transmit frames at a constant known TX rate. In Fig. 5, the RX rate indicates the number of CSI samples collected in a single second and may be different from the TX rate in cases where packets are missed due to interference or when packets are dropped due to cyclic redundancy check (CRC) errors or other communication issues. For each TX rate value, we transmit for a period of 60 seconds such that the lines in the figure indicate the mean sampling rate and the error bars indicate one standard deviation from the mean.[4]

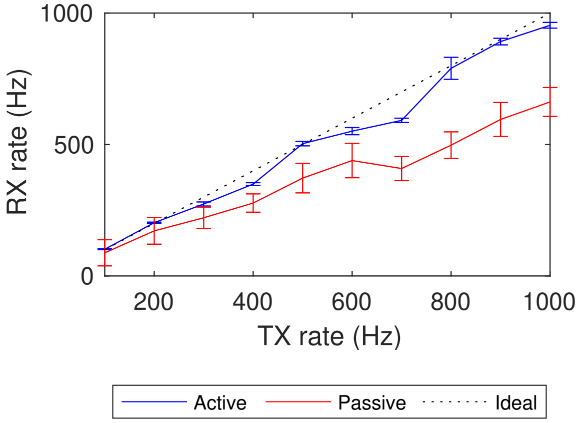


Fig. 5. Number of packets received per second at an active (i.e., connected) and passive (i.e., sniffing) receiver when a transmitter sends CSI frames at varying rates.

We begin by considering the *active* setting where the RX is the destination for each packet transmitted by the TX. The RX rate increases almost linearly as TX rate increases to 1000Hz. Small dips in RX rate appear due to the tick interrupt rate of the real-time operating system (RTOS) running on-board the ESP32 TX device which artificially reduces the actual number of frames that are sent. Overall, this shows that the ESP32 can collect CSI samples at an RX rate upwards of 1000Hz in the active scenario.

Next, we evaluate the *passive* setting where a third device (PX) is passively listening to the communication between the TX and RX. In this scenario, because the packet destination is not PX, the PX cannot request packet retransmission when communication errors may occur. As a result of this, we find the RX rate for PX is between 13% and 37% lower than the active scenario. Additionally, compared to the

[4] We calculate RX rate without sending the CSI data over serial from the ESP32 to the host device. The baud rate of the serial interface limits the CSI throughput and is not necessary for on-device model inference.

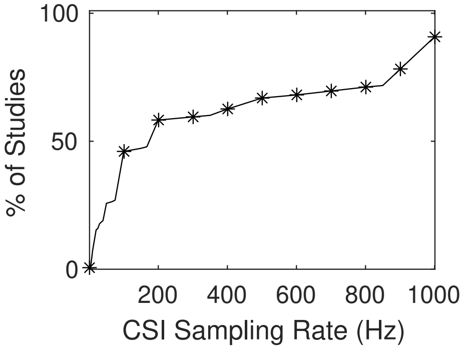


Fig. 6. CDF of CSI sampling rates from surveyed literature ($N = 176$).

active case, we can see a higher standard deviation for all values of TX rate indicating that there is a large variance in the number of samples collected over the 60 second period. In the case of TX rate of 1000Hz, this results in an RX rate of 662Hz and a standard deviation of 55Hz for the PX.[5]

Optimal sampling rates for a given sensing task can be selected based on the Nyquist-Shannon sampling theorem [88] which suggests that a sampling rate of at least $2R$Hz must be used to capture an activity performed at $R$Hz. For example, in [89] it is stated that indoor exercise activities produce motion-induced frequency shifts at a rate below $R = 40$Hz and as such, a rate of 100Hz is selected which is greater than $2R$Hz and thus should be able to capture the important movements during these activities. We found 176 research studies which specify the CSI sampling rate used in their data collection. Fig. 6 shows the CDF plot of the sampling rate used in these works. From this figure, we can see that almost 50% of works set a sampling rate of 100Hz or lower. Both the active and passive scenarios shown in Fig. 5 can achieve rates above 100Hz, and as such, we would expect that the ESP32 could be used for WiFi sensing in most of the scenarios discussed in these works. We find that

[5] PX captures CSI from the two way communication between TX and RX. In our results, we ignore half of the frames (i.e., from RX) to allow for a better understanding of packet loss with PX compared to the active RX.

when higher sampling rates are used, they are often manually reduced afterwards to decrease computation complexity as well as memory requirements. For example, in [90], a sampling rate of 1000Hz was selected, but the signal was passed through a band-pass filter between the ranges of 5Hz and 80Hz to capture the frequency changes caused by human walking within these frequency bands. Using the Nyquist theory here, we may assume that a sampling rate of 160Hz would have been sufficient. As such, while the ESP32 was unable to achieve higher sampling rates in the passive scenario, the active scenario should still be applicable for higher applications which require high sampling rates.

### 3.5 Broader Impact

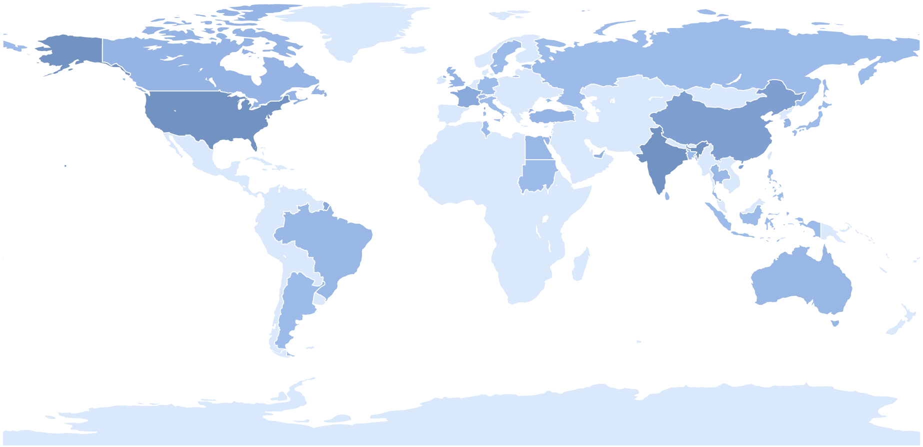


Fig. 7. Active communication with researchers from 30+ unique countries around the world regarding the work discussed in this dissertation.

The WiFi sensing research and tools discussed throughout this dissertation have resulted in active incoming communications from fellow researchers from over 30 countries around the world (see Fig. 7). This includes both academic researchers as well as industry researchers ranging from start-ups to major technology companies.

The WiFi sensing toolkit has so far achieved $> 150$ stars, $> 50$ forks and between $100 - 300$ page views per week on Github. Sharing these open source projects with fellow researchers pushes the field further by finally allowing WiFi sensing to be performed with low cost edge devices for the first time. Furthermore, by encouraging the use of edge WiFi sensing, this work encourages improved sustainability by relying on less expensive and more energy-efficient models.

# CHAPTER 4

# SIGNAL PROCESSING AND MACHINE LEARNING TECHNIQUES AND THEIR CHALLENGES IN REAL-WORLD EDGE SYSTEMS

## 4.1 Introduction

In this chapter, we survey signal processing and machine learning techniques used for WiFi sensing tasks by focusing on achieving on-device processing for low powered edge devices. Typically, the metrics used to evaluate WiFi sensing systems (i.e., prediction accuracy, training speed, and inference speed) are calculated when running on high powered desktop-level GPUs or even multi-GPU servers [91, 92]. As such, deploying conventional WiFi sensing systems is far too costly and bulky for scalable deployments in the real-world, thus constraining the practicability of on-the-edge WiFi sensing systems.

Instead, we explore the standalone ESP32 microcontroller which allows access to the rich WiFi CSI data directly from the WiFi-enabled microcontroller. This unique feature of the ESP32 allows us to easily deploy a lightweight, standalone and low cost device for CSI collection and recording as well as for signal processing and prediction making at the edge. We evaluate existing CSI signal processing techniques which are historically computed with powerful computers and demonstrate that these techniques can be performed on much smaller microcontroller devices which allows for an immediate improvement in scalability of WiFi sensing systems. Furthermore, we also demonstrate in this work that the ESP32 is capable of running machine learning inference directly on-board, further reducing its dependency on external devices and thus demonstrating the possibility of performing WiFi sensing on a standalone edge system. The main contributions of this chapter can be summarized as follows:

- We develop a taxonomy for edge WiFi sensing systems which considers theory, signal processing, data preparation, prediction making, systems-level and hardware-level concerns along with identifying new and important metrics for evaluating edge WiFi sensing systems.

- We perform a thorough survey into WiFi sensing studies to identify common signal processing techniques and to determine the feasibility of running such methods on a low-level microcontroller on the edge. We also consider which techniques require environment-specific calibration and evaluate how calibration can be performed in a new online setting unlike previous research which assume offline calibrations.

- We evaluate signal processing techniques on a variety of tasks to demonstrate the use of WiFi sensing for different real-world online use-cases including (i) *small-scale hand gesture recognition* which can be used for novel device-free human-computer interaction (HCI), (ii) *medium-scale human activity recognition* which can be used to track behaviours of a person over time, and (iii) *large-scale human activity and localization sensing* which can be used to understand the movements and behaviours of people throughout an environment.

- We evaluate different aspects of an ESP32-based WiFi sensing system such as (i) computation time required for the surveyed signal processing techniques, (ii) machine learning model inference rate, and (iii) energy consumption.

There are some previous surveys on WiFi sensing, however they typically focus on applications of WiFi sensing or deep learning techniques. However, our focus in

Table 4. Comparison of This Survey to Existing Surveys On WiFi Sensing.

| | Key Criteria | This Chapter | Jiang et al. [93] | Ma et al. [94] | Liu et al. [95] | He et al. [96] | Liu et al. [97] | Li et al. [98] | Nirmal et al. [99] |
|---|---|---|---|---|---|---|---|---|---|
| Info | Year | 2023 | 2018 | 2019 | 2019 | 2020 | 2020 | 2021 | 2021 |
| | Focus | Edge ML | Smart Home | Tasks | HAR | Tasks | HAR | ML | ML |
| Topics | CSI-based Sensing Theory | ● | ● | ● | ● | ● | ● | ○ | ○ |
| | Signal Processing Techniques | ● | ○ | ◑ | ○ | ◑ | ◑ | ○ | ○ |
| | Machine Learning | ● | ○ | ◑ | ○ | ○ | ◑ | ● | ● |
| | Real-World Integration | ● | ◑ | ○ | ○ | ○ | ○ | ○ | ○ |
| | Possible Applications | ● | ● | ● | ● | ● | ● | ○ | ○ |
| | Hardware Requirements | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| Evaluations | New Experiments | ● | ○ | ○ | ○ | ○ | ● | ○ | ● |
| | Accuracy | ● | ● | ● | ● | ● | ◑ | ○ | ● |
| | Inference Rate | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| | Energy Consumption | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ |

● Fully addressed, ◑ Partially Addressed, ○ Not addressed.

this work for the first time emphasizes the use of WiFi sensing on-board embedded edge devices. Table 4 compares this chapter to previous WiFi sensing surveys.

## 4.2 Edge WiFi Sensing Taxonomy

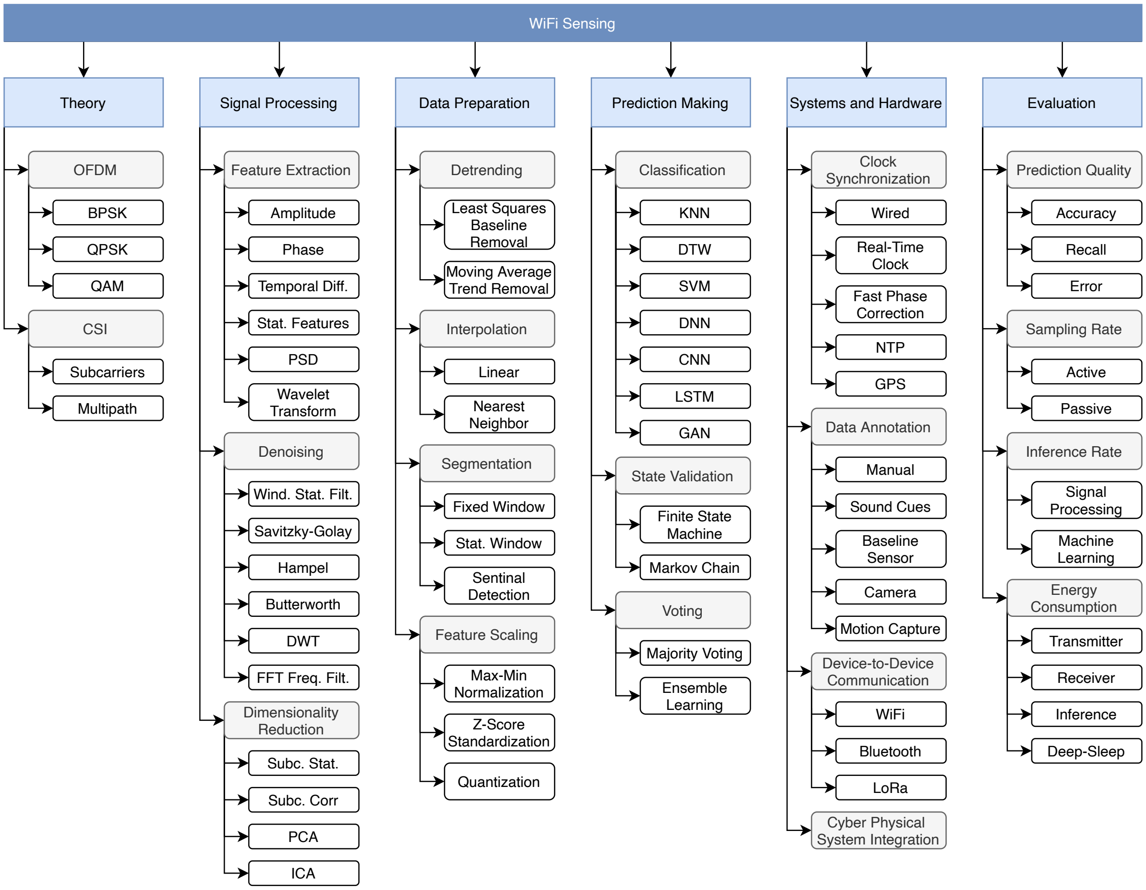


Fig. 8. Taxonomy of subjects necessary for edge-based WiFi sensing systems.

To produce an edge-based WiFi sensing system, it is important to understand common existing techniques used in current CSI-based WiFi sensing research. Since existing studies typically use high-powered desktops or servers, the complexity of signal processing algorithms has not been much of a concern in these studies. However, because we consider the use of low-power microcontroller devices for WiFi sensing, the complexity of signal processing algorithms becomes a much more important area to focus on. Each technique has unique characteristics that determine its applicability in this new scenario. Most existing research ignores the need for real-time data processing and assumes that data will instead be processed offsite at a powerful server. Since we target a WiFi sensing system on the edge, such techniques would not be possible, thus we must survey available techniques to validate their use for our proposed system.

Through a thorough survey of existing works, we develop a taxonomy of components necessary for edge based WiFi sensing systems as shown in Fig. 8. In Section 2.2, we discussed the background theory of WiFi sensing with CSI. CSI can be collected from WiFi-enabled devices such as the Intel 5300 NICs [100], Atheros NICs [101] or with edge microcontrollers such as the ESP32 [102].

In this section, we discuss four components from our taxonomy, namely: signal processing, data preparation, prediction making, and systems and hardware. We also consider the applicability of each technique in resource constrained microcontroller devices. Specifically, we focus on the online calculations that need to be performed for every received CSI sample rather than focusing on computations that can be done beforehand during an initialization phase.

### 4.2.1 Signal Processing

The first component after CSI data collection is to run the collected CSI data through signal processing. These signal processing techniques are standard tasks which will typically be applied in any WiFi sensing application. The purpose of signal processing is to achieve improved accuracy through steps like feature extraction, denoising, and dimensionality reduction. Most methods have a unique set of parameters which can be tuned to improve the accuracy of the technique for different applications. Tables 5, 6, and 7 shows an overview of the discussed signal pro-

Table 5. Feature extraction techniques along with their time and memory complexity when implemented as an online algorithm for low-resource IoT devices. Complexity variables are defined in Table 8.

| | **Technique** | **Complexity per Frame** | | **Sources** | **Advantages** | **Disadvantages** |
|---|---|---|---|---|---|---|
| | | **Memory** | **Time** | | | |
| ***Feature Extraction*** | Amplitude | $O(S)$ | $O(S)$ | [16],[103],[104] | Default CSI representation. | May contain anomalies which require denoising. |
| | Phase | $O(S)$ | $O(S)$ | [105],[106],[107] | Default CSI representation. | Requires multiple antennas for phase correction. |
| | Temporal Difference | $O(S)$ | $O(S)$ | [32],[59],[106] | Tracks relative change, not absolute change. | Typically used with CSI phase. |
| | Statistical Features | $O(S)$ | $O(S)$ | [108],[109],[110] | Easy to compute. Reduces dimensionality per CSI frame. | Loses important per-subcarrier information. |
| | PSD | $O(w)$ | $O(wlogw)$ | [31],[50],[61] | Creates frequency-domain features. | Applied to a single subcarrier. Loses other subcarrier information. |
| | Wavelet Transform | $O(S\|\psi\|J)$ | $O(S\|\psi\|J)$ | [105],[111],[112] | Creates frequency-domain features. | Higher complexity than other feature extraction methods. |

Table 6. Denoising filter techniques along with their time and memory complexity when implemented as an online algorithm for low-resource IoT devices. Complexity variables are defined in Table 8.

| | **Technique** | **Complexity per Frame** | | **Sources** | **Advantages** | **Disadvantages** |
|---|---|---|---|---|---|---|
| | | **Memory** | **Time** | | | |
| ***Denoising Filter*** | Windowed Statistical Filter | $O(wS)$ | $O(wS)$ | [39],[113],[114] | Simple implementation. | Does not retain original waveform. |
| | Savitzky-Golay Filter | $O(wS)$ | $O(wS)$ | [55],[115] | Closely maintains steep peaks and valleys in waveform. | Poor anomaly filtering. |
| | Hampel Filter | $O(wS)$ | $O(Swlogw)$ | [18],[27],[42] | Retains exact waveform except for anomalies. | Anomalies detected may in-fact be important. |
| | Butterworth Filter | $O(wS)$ | $O(wS)$ | [48],[116],[117] | Filters noise outside of frequency ranges of interest. | Frequency ranges dependant on application. |
| | DWT | $O(S\|\psi\|J)$ | $O(S\|\psi\|J)$ | [118],[119],[120] | Frequency-domain filtering can be applied to multiple frequency ranges in one-pass. | Frequency filtering is more coarse than Butterworth filter. |
| | FFT Frequency Filter | $O(S)$ | $O(SlogS)$ | [40] | Filters noise due to multipath environment. | Filter is applied per-frame, not applied over time range. |

Table 7. Dimensionality reduction techniques along with their time and memory complexity when implemented as an online algorithm for low-resource IoT devices. Complexity variables are defined in Table 8.

| | **Technique** | **Complexity per Frame** | | **Sources** | **Advantages** | **Disadvantages** |
|---|---|---|---|---|---|---|
| | | **Memory** | **Time** | | | |
| ***Dimensionality Reduction*** | Subcarrier Statistical Features | $O(S)$ | $O(S)$ | [32],[40] | Simple calibration phase. Simple online phase. | Filtered subcarriers may still have useful information. |
| | Subcarrier Correlation | $O(S)$ | $O(S)$ | [121],[122] | Simple online phase. | Correlated subcarriers may contain redundant information. |
| | PCA | $O(SC)$ | $O(S^2C)$ | [48],[123],[124] | Mixes subcarriers before reduction to retain information. | Complex calibration phase. |
| | ICA | $O(SC)$ | $O(S^2C)$ | [56],[125] | Designed to separate signal into $C$ independent sources. | Typically used with multi-antennas. |

Table 8. Variable definitions for Tables 5, 6, 7, 9 and 10.

| Variable | Description |
|---|---|
| $S$ | Number of subcarriers |
| $w$ | Window size |
| $\lvert\psi\rvert$ | Length of discrete wavelet |
| $J$ | Number of wavelet decomposition levels |
| $C$ | Number of components extracted from PCA and ICA |

cessing techniques, along with their memory complexity, time complexity as well as advantages and disadvantages of each technique. For each method, we provide multiple reference sources which explain each method in a different way or use a unique mathematical formulation rather than citing studies which simply apply the given method. We take this approach to allow the reader to gain a greater understanding of the methods from different points of view. The variables used to define time complexity and memory complexity are described in Table 8.

#### 4.2.1.1 Feature Extraction

We begin evaluating signal processing techniques by reviewing common feature extraction methods. Feature extraction transforms raw CSI data into meaningful features for further processing and for machine learning model injection.

*Amplitude and Phase:* The most fundamental feature extraction method for WiFi sensing is to convert CSI into amplitude ($A$) or phase ($\phi$) features as shown in Equation (2.3) and Equation (2.4), respectively. For each CSI frame collected, $S$ subcarriers are received which can then immediately be converted to either amplitude or phase with memory complexity and time complexity of $O(S)$. For the following sections,

$h[t]$ will denote a single CSI signal measurement for some subcarrier $s$ at time $t$ which could be either amplitude, phase or some other derived signal value.

*Temporal Difference:* For both amplitude and phase, the absolute value may not be as important as the relative change of the feature over time [59]. Instead, using the temporal difference over subsequent time steps (i.e., $h_{\text{diff}}[t] = h[t] - h[t-1]$) is a common feature extraction step. With amplitude for example, when the temporal difference is negative the amplitude has decreased which possibly indicates that the LOS between TX and RX has been blocked by some obstruction such as a human target and alternatively, when the temporal difference is positive, this may indicate the LOS has been cleared of an obstruction. Due to the noisy nature of received phase, some studies [106] have used the relative phase from the temporal difference after applying phase unwrapping to better understand how much change occurred over some time span.

*Statistical Features:* Standard statistical aggregation functions (e.g., *mean*, *standard deviation*, *median*, *kurtosis*) are used to compress the high dimensional subcarrier data per frame down to a single higher-level feature value. Furthermore, spectral statistical functions (e.g., *spectral kurtosis*, *spectral spread*, *spectral slope*) can be used given that the data is represented in the frequency domain rather than in the time domain. Depending on the statistical function, the time complexity and the memory complexity may change, but in general, when the functions are applied to the subcarriers from a single CSI frame, the time and memory complexity are $O(S)$.

More commonly, statistical features are extracted from a time-series window of size $w$ independently per subcarrier. To accomplish this, a buffer of size $O(wS)$ can be allocated to store the data for aggregation. Due to the addition of a windowed buffer, the time complexity also increases for performing the aggregation for each subcarrier. However, trivial statistical functions such as *mean* $(\mu(\cdot))$ can achieve reduced time complexity in an online system through an iterative implementation. For example, for a single subcarrier $s$ at time $t$, $\mu(t) = \frac{1}{w}\sum_{i=0}^{w-1} h[t-i]$ takes $O(w)$ time while a recursive implementation $\mu(t) = \mu(t-1) + \frac{1}{w}(h[t] - h[t-w])$ has $O(1)$ time complexity per subcarrier while still requiring the same memory complexity of $O(w)$ per subcarrier.

*Power Spectral Density:* Power Spectral Density (PSD)[1] [50] converts the time-series CSI signal $(h)$ into the frequency domain $(\tilde{h})$. Typically, we find that this conversion is only applied to a single subcarrier, however it is also possible to compute this value independently per subcarrier. To compute PSD, we keep a buffer of window size $w$ and compute

$$h_{PSD}[t] = \frac{|FFT_w(h[t-w+1:t])|^2}{w}. \tag{4.1}$$

On a single subcarrier, this method has a time complexity of $O(wlogw)$ and a memory complexity of $O(w)$. This produces a vector of size $|h_{PSD}[t]| = w$ even though the input is only a single subcarrier.

*Wavelet Transform:* Wavelet transformations compress a signal from a time-series representation and transform it into a set of time-frequency domain components. Wavelet transform is achieved by decomposing the input signal recursively into a vector of approximation coefficients $(\alpha^{(J)})$ as well as a set of detail coefficient vectors $\{\beta^{(1)}, \beta^{(2)}, \cdots, \beta^{(J-1)}, \beta^{(J)}\}$ where $J$ is the number of decomposition levels. Both approximation coefficient vector and detail coefficient vectors can be computed through downsampling convolutional equations [127]:

$$\alpha^{(J)}[t] = \sum_{i=0}^{|\psi|-1} g[i]\alpha^{(J-1)}[t-i], J \in \mathbb{Z} \tag{4.2}$$

[1] Energy Spectral Density (ESD) is another term used when PSD is computed over small time windows [126].

$$\beta^{(l)}[t] = \sum_{i=0}^{|\psi|-1} h[i]\alpha^{(J-1)}[t-i], l \in 1, \cdots, J \tag{4.3}$$

where $g$ is the high-pass filter and $h$ is the low-pass filter derived from the wavelet basis function ($\psi$) (i.e., Haar or Daubechies Wavelets) such that $g[|\psi| - n + 1] = (-1)^n \times h[n]$, where $|\psi|$ is the length of the coefficients for the wavelet basis function and $n \in \{1, \ldots, |\psi|\}$. Both $\alpha^{(J)}$ and $\beta^{(l)}$ are downsampled to remove every other element in the array such that $|\alpha^{(J)}| = \frac{1}{2}|\alpha^{(J-1)}|$ and $|\beta^{(l)}| = \frac{1}{2}|\beta^{(l-1)}|$. Note that the initial approximation vector $\alpha^{(0)}[t] = h[t]$, is our original CSI signal measurements, and as such, $|h[t]| = |\alpha^{(0)}[t]| = |\alpha^{(1)}[t]| + |\beta^{(1)}[t]| = |\alpha^{(J)}[t]| + \sum_{l=1}^{J} |\beta^{(l)}[t]|$ which shows that even though we are downsampling at each level, the number of elements retained is always $|h[t]|$ no matter the value for $J$ when computed using the pyramid algorithm [128].

Each level of decomposition also results in a halving of the sampling rate and as such, a halving of the frequency spectrum. For example, given CSI sampling rate of $R$Hz, the detail coefficient vector $\beta^{(l)}$ for level $l$ captures frequency ranges from $\frac{R}{2^l}$Hz to $\frac{R}{2^{l+1}}$Hz and $\alpha^{(l)}$ captures frequency ranges from $\frac{R}{2^{l+1}}$Hz to 0Hz. As such, different sub-ranges of frequency bands reveal more relevant information depending on the task. For example, in [111], it was found that when using six-level wavelet decomposition, the detail coefficient vectors $\beta^{(4)}, \beta^{(5)}, \beta^{(6)}$ and approximation coefficient vector $\alpha^{(6)}$ are most effective at revealing motion-induced variations for the task of occupancy detection. Similarly, in [105] a four-level wavelet decomposition was applied where both $\beta^{(3)}$ and $\beta^{(4)}$ were used as input for the task of breathing rate detection.

For a real-time online data processing system, a few important algorithm design issues must be considered. Given a $J$-level wavelet transform decomposition, due to the recursive nature of the wavelet transform, at time $t$, $\alpha^{(J)}[t]$ is a function of our signal from time $t$ all the way back to time $t - \sum_{l=1}^{J}(|\psi| - 1)2^{J-1}$ where $|\psi|$ is the

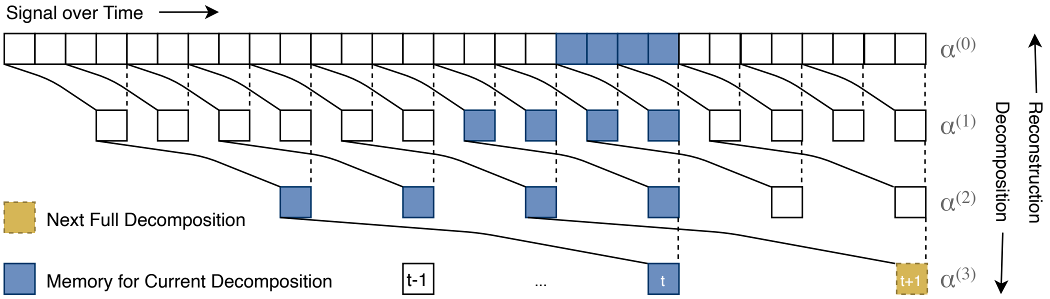


Fig. 9. Wavelet decomposition.

length of the wavelet function coefficients. As such, for the previous examples in [111] where $J = 6$ and $\psi =$ "db6" such that $|\psi| = 12$, both $\alpha^{(J)}[t]$ and $\beta^{(J)}[t]$ are functions of 694 input CSI samples. This not only introduces significant lag into the system, but also suggests a large amount of computation work.

A simplified example of wavelet decomposition is shown in Fig. 9 where the first layer of blocks represents distinct signal samples over time and each lower layer represents the $J = 3$ recursive decomposition of the approximation coefficient vector $\alpha^{(l)}$ and finally $|\psi| = 4$. In each layer, $|\alpha^{(l)}| = \frac{1}{2} \times |\alpha^{(l-1)}|$ due to downsampling. In the illustration, for decomposition layer $l = 1$ (second layer), we can see that $\alpha^{(1)}[t]$ depends on four consecutive signal samples because $|\psi| = 4$ highlighted in blue. Similarly for the second decomposition layer we can see that $\alpha^{(2)}[t]$ relies on four coefficients from $\alpha^{(1)}$ which recursively have their own dependencies in $\alpha^{(0)}$. As such, $\alpha^{(2)}[t]$ is a function of 10 original signal samples from $h$. Finally in the third decomposition layer, $\alpha^{(3)}[t]$ again only has four direct dependencies from $\alpha^{(2)}[t]$, but because of the recursive nature of the algorithm, a total of 22 signal samples are required to make up $\alpha^{(3)}[t]$. If another layer of decomposition was attempted, then 46 signal samples would be required to compute $\alpha^{(4)}[t]$ and so on. If we perform 10-layer decomposition, then 3070 input samples would be required to compute $\alpha^{(10)}[t]$.

Luckily, the recursive structure of the wavelet decomposition ensures that results can be cached rather than recomputed at each time point. In the illustration, when a CSI sample arrives at time $t$, only the blue boxes are required for the computation which means that the memory complexity for each arriving CSI sample (per subcarrier) is only $O(|\psi|J + 1)$ and due to the convolution operation, the time complexity is $O(|\psi|^2 J)$. We can see in Fig. 9 that there is some lag introduced between the computation of $\alpha^{(J)}[t]$ and $\alpha^{(J)}[t+1]$ because the approximation coefficient vector for lower intermediate levels needs to be computed before $\alpha^{(J)}[t+1]$ can be computed. This lag can be counted by the number of CSI signal samples and is equal to $2^J$ samples. This means that as $J$ increases, the prediction rate will decrease because the approximation coefficient vector and all detail coefficient vectors need to be fully computed before subsequent predictions can be performed.

#### 4.2.1.2 Denoising Filters

Noise in the collected CSI data has been a great concern for many studies. Noise can originate from a number of sources including differences in hardware such as with Central Frequency Offset (CFO) errors and Sampling Frequency Offset (SFO) errors. It can also come from environmental conditions such as through signal shadowing due to LOS interference or multipath fading where the signal arrives to the receiving antenna from multiple NLOS paths through the environment causing destructive interference. As such, a large number of unique methods have been proposed for denoising the incoming signal. Denoising is most commonly performed independently per subcarrier.

*Windowed Statistical Filter:* Through our survey, we find that simple denoising filters such as the mean [39] and median [113] windowed filters are commonly used. The mean filter computes $\hat{h}[t] = \frac{1}{w}\sum_{i=0}^{w-1} h[t-i]$ which can be calculated on a rolling basis as new data appears. Similarly, a weights vector $g$ of length $w$ can be used to produce a weighted moving average $\hat{h}[t] = \frac{1}{w}\sum_{i=0}^{w-1} g[i]h[t-i]$ which can give less weight to time instances further away from the current time instance and greater weight to more recent time instances. The median filter on the other hand requires a slightly higher time-complexity due to the use of the median function ($Med(\cdot)$) as so: $\hat{h}[t] = Med\big(\{h[t-w+1], h[t-w+2], \ldots, h[t-1], h[t]\}\big)$. Median filter makes up for this higher complexity by more robustly handling highly anomalous noise within the signal.

*Savitzky-Golay Filter (SG):* This filter fits a rolling window of data points with a low-degree polynomial through linear least squares to smooth out the incoming signal. In [115], it is suggested that SG filter can maintain the shape of the waveform better than a standard infinite impulse response (IIR) low-pass filter. To perform this smoothing, a coefficients vector $W_{SG}$ of size $k_{SG} = |W_{SG}|$ is used such that

$$\begin{aligned} h_{SG}[t] &= (W_{SG} * h)[t] \\ &= \sum_{i=0}^{k_{SG}-1} W_{SG}[i]h[t-i]. \end{aligned} \tag{4.4}$$

Obviously, when $W_{SG}[i] = \frac{1}{k_{SG}}, \forall i \in \{0, \ldots, k_{SG}-1\}$ then $\sum_{i=0}^{k_{SG}-1} W_{SG}[i] = 1$, and SG filter will simply compute the rolling average window. With more complex selection of $W_{SG}$, polynomial fitting can be achieved [129]. Due to the simple convolution operation, SG filter requires a memory complexity and time complexity of $O(k_{SG}) = O(w)$ per subcarrier. The SG filter has been suggested [130] because it preserves the steep peaks and valleys of the original signal $h$.

*Hampel Filter:* The Hampel filter [110] is used to remove anomalies without changing the signal values for non-anomalous values through:

$$\hat{h}[t] = \begin{cases} h[t] & |h[t] - Med_{t,w}(h)| \leq 3 \times MAD_{t,w}(h) \\ Med_{t,w}(h) & \text{otherwise,} \end{cases} \tag{4.5}$$

where $MAD_{t,w}(\cdot)$ is the Median Absolute Deviation (MAD) function for the signal window from $h[t-w+1]$ until $h[t]$ and $Med_{t,w}(\cdot)$ is the median function over the same window buffer.[2] Filtering anomalous data in this way is useful for filtering out outliers caused by hardware related errors such as through quantization errors while also retaining much of the original, non-anomalous signal values.

*Butterworth Filter:* Noise will be mostly produced by other physical phenomenon in the environment. For example, when recording human hand gestures, the human body may slowly move during the gestures. Furthermore, background items such as fans may produce fast variations in the noise. Thus, high-pass, low-pass and band-pass filters have commonly been employed, most commonly in the form of Butterworth filters [116]. The goal of the Butterworth filter is to produce a maximally flat amplitude response in the defined frequency bands while also reducing the amplitude response outside of the specified frequency bands [131]. Butterworth filters apply a rational transfer function to the input data given as a set of coefficients $a$ and $b$, where $|a| = |b| = n + 1 = k_{BF}$ where $n$ is the order of the Butterworth filter. Output $h_{BF}[t]$ of the transfer function is calculated recursively using the Direct-Form-II

[2] These filters are described assuming that the current time point is the final value in the window function. Some studies consider the current time point as the center of an odd-sized window. This may be helpful during rising-edge and falling-edge cases, but introduces some lag in signal processing. Only trivial changes are required to alter these windowing methods.

(DF-II) structure [131]:

$$h_{BF}[t] = \left( \sum_{i=0}^{k_{BF}-1} b[i]h[t-i] \right) - \left( \sum_{i=1}^{k_{BF}-1} a[i]h_{BF}[t-i] \right). \tag{4.6}$$

The key observation here is that while the Butterworth filter is an IIR filter; which means that each $h_{BF}[t]$ is a function of all previously seen signal elements in $h$, the buffer size required for computing $h_{BF}[t]$ is only of size $O(w)$ to store $a$, $b$ and $h_{BF}$. This makes both the time and memory complexity $O(w)$. Thus, the Butterworth filter is another reasonable candidate for computation and memory constrained systems such as low-power IoT devices.

*Discrete Wavelet Transform:* Another very common method for denoising is through the discrete wavelet transform (DWT) [118]. DWT is used to decompose time-domain signals into a time-frequency domain representation. Denoising with DWT is performed in three stages, first the signal is decomposed recursively through DWT into a vector of approximation coefficients ($\alpha^{(J)}$) as defined in Equation (4.2) as well as a set of detail coefficient vectors ($\beta^{(1)}, \beta^{(2)}, \cdots, \beta^{(J-1)}, \beta^{(J)}$) as defined in Equation (4.3) where $J$ is the number of decomposition levels. After decomposing the signal into $J$ detail coefficient vector levels, threshold based denoising is applied. For each coefficient element ($\beta^{(l)}[i]$) within each level of detail ($l$), we can then apply either a soft threshold [127] by:

$$\tilde{\beta}^{(l)}[i] = \begin{cases} sign\big(\beta^{(l)}[i]\big)\big(|\beta^{(l)}[i]| - \tau\big) & \text{if } |\beta^{(l)}[i]| \geq \tau \\ 0 & \text{otherwise,} \end{cases} \tag{4.7}$$

or a hard threshold by:

$$\bar{\beta}^{(l)}[i] = \begin{cases} \beta^{(l)}[i] & \text{if } |\beta^{(l)}[i]| \geq \tau \\ 0 & \text{otherwise,} \end{cases} \tag{4.8}$$

where $\tau$ is a pre-defined threshold. After applying this threshold method, we can reconstruct the original time-domain signal from this frequency representation. This reconstruction stage reverses the decomposition steps as illustrated in Fig. 9. To accomplish reconstruction, the inverse discrete wavelet transform (IDWT) is used [127, 118]:

$$\alpha^{(l)}[t] = \sum_{i=0}^{w-1} \bar{g}[i]\alpha_{UP}^{(l+1)}[t-i] + \sum_{i=0}^{w-1} \bar{h}[i]\hat{\beta}_{UP}^{(l+1)}[t-i], \tag{4.9}$$

where $\alpha_{UP}^{(l)}$ is an upsampled version of $\alpha^{(l)}$ accomplished by

$$\alpha_{UP}^{(l)}[n] = \begin{cases} \alpha^{(l)}\left[\lfloor \frac{n}{2} \rfloor\right] & \text{if } n \text{ is even} \\ 0 & \text{otherwise,} \end{cases} \tag{4.10}$$

which allows $|\alpha_{UP}^{(l)}| = 2|\alpha^{(l)}| = 2|\alpha_{UP}^{(l+1)}|$ and $\bar{g}$ and $\bar{h}$ are the reconstruction high-pass and reconstruction low-pass filters, respectively, such that $\bar{g}[n] = g\big[|\psi| - n + 1\big]$ and $\bar{h}[n] = h\big[|\psi| - n + 1\big]$ when $n \in \{1, \ldots, |\psi|\}$. This recursive operation is performed from $l = J$ until $l = 0$ at which point $|\hat{h}| = |\alpha^{(0)}|$ which implies that $|\alpha^{(0)}| = |h|$ showing that the computed signal length after DWT denoising (decomposition, thresholding, reconstruction) is the same as the length of the initial signal. However, depending on the levels of decomposition, there will be lag introduced relative to the size of the selected $J$ and $|\psi|$.

*FFT Frequency Filter:* A single CSI sample contains many subcarriers, each of which is a frequency-domain representation of the signal. This means that we can use the IFFT to capture the power delay profile (PDP) in the time-domain [40]:

$$\tilde{h}[t] = \sum_{i=1}^{N} a_i e^{-j\theta_i}\delta(t - t_i), \tag{4.11}$$

where $N$ is the multipath count, $a_i$ and $\theta_i$ are the amplitude and phase angle of the given multipath, $t_i$ is the time delay introduced by the given multipath and $\delta(\cdot)$ is the Dirac delta function. Given an initial CSI sample with 64 subcarriers, the output of *IFFT* will also be of size 64 where each element of the vector represents time. Due to the time delay introduced by different multipaths in the environment, each element in PDP will be affected slightly differently. This understanding has been used to achieve denoising by performing *multi-path mitigation* [40]. To do this, after transforming CSI into PDP, components with large time delays are removed as so:

$$\tilde{h}'[t] = \begin{cases} \tilde{h}[t] & \text{if } t < T_{\text{PDP}} \\ 0 & \text{otherwise,} \end{cases} \tag{4.12}$$

where $T_{\text{PDP}}$ is the allowed time delay. After this, it is possible to convert PDP from the time-domain representation back to a frequency-domain CSI representation through standard FFT. The FFT Frequency Filter is computed immediately on each incoming CSI sample independently and thus, this method will not introduce lag for a real-time system.

#### 4.2.1.3 Dimensionality Reduction

Each collected CSI sample comprises of a complex vector of $S$ subcarriers. It has been shown in previous studies (e.g., [132]) that some subcarriers have similar and thus redundant information while other subcarriers are plagued with high amounts of noise. To combat these issues, dimensionality reduction can be applied to remove the data from these useless subcarriers thus reducing the number of subcarriers to $\hat{S} < S$.

*Subcarrier Statistical Feature:* A simple method for selecting subcarriers is to consider statistical properties of each subcarrier over some pre-defined time frame. For example, many studies [133, 132] select subcarriers which exhibit the highest variance indicating that the subcarrier has a high sensitivity for the environment. Variance can be calculated on a moving window to allow subcarrier selection to change over time as shown in [132] or more commonly the variance per subcarrier can be computed beforehand in a calibration phase for the environment [133]. When computing the moving window, a buffer of size $O(wS)$ is required while pre-computed subcarrier variance values can allow the system to immediately filter out low quality subcarriers with only $O(S)$ time and memory complexity. Other metrics such as signal-to-noise ratio (SNR) [58] and mean absolute deviation  [59] have also been used as metrics that can be computed independently per subcarrier for the task of subcarrier selection.

*Subcarrier Correlation:* Another common method for subcarrier selection is to look at the relationship between each individual subcarrier by computing a correlation coefficient matrix of size $S \times S$ [134]. The goal is that subcarriers with high correlation are in agreement about the true state of the environment and thus we can assume that the noise present in the signal is appearing due to the environment rather than from some spurious noise source. Interestingly, in [130] it was shown that the phases of some subcarriers have highly negative correlation which implies that the subcarriers are being affected by the same environmental events but are being affected in opposite directions, so the absolute correlation coefficient matrix may be preferred. Subcarrier correlation will typically be computed during the calibration step to determine which subcarriers to keep and which subcarriers to filter for each CSI frame. As such, the online time complexity and the memory complexity remains at $O(S)$.

*Principal Component Analysis (PCA):* Principal Component Analysis (PCA) [135, 136] is a common method for dimensionality reduction with the added benefit of also increasing the SNR of the data through a linear transformation. PCA relies on an initial calibration phase to compute a components coefficients matrix $\mathbf{C}_{\text{PCA}}$ of size $k_{\text{PCA}} \times S$ through eigendecomposition where $k_{\text{PCA}}$ is the desired number of components to retain. To compute the $k_{\text{PCA}}$ principal components at time instance,

$t$:

$$h_{\text{PCA}}[t] = \mathbf{C}_{\text{PCA}} \times (h[t] - \mu), \tag{4.13}$$

where $\mu$ is the vector of subcarrier means found during the calibration phase such that $|\mu| = S$. This process results in a reduction in dimensionality for the CSI sample from size $S$ subcarriers down to $k_{\text{PCA}}$ principal components. The memory complexity for computing this value per sample is $O(k_{\text{PCA}}S)$ due to the size of $\mathbf{C}_{\text{PCA}}$ and the time complexity for each sample is $O(k_{\text{PCA}}S^2)$ due to the matrix multiplication required at each time instance.

*Independent Component Analysis (ICA):* Signal variations in the received CSI are affected directly by different noise sources such as environmental noise or specific physical movements. Independent Component Analysis (ICA) attempts to solve the blind-source separation problem by splitting out the noise caused by each unique source (i.e., each individual human in an environment or each distinct body part during single human body movements). ICA has been used in WiFi sensing tasks such as for separating out respiration signals [56]. ICA relies on an initial calibration phase to compute a components coefficients matrix $\mathbf{C}_{\text{ICA}}$ of size $k_{\text{ICA}} \times S$ through singular value decomposition (SVD) where $k_{\text{ICA}}$ is the desired number of components to retain.

It should be noted that while both PCA and ICA are looking to accomplish very different tasks; computationally, both methods use Equation (4.13) to perform dimensionality reduction. As such, both methods behave exactly the same while performing the algorithm online. The only unique aspects are how $\mathbf{C}_{\text{PCA}}$ and $\mathbf{C}_{\text{ICA}}$ are computed during the calibration phase.

### 4.2.2 Data Preparation

Data preparation techniques, unlike signal processing are more application-specific and thus are not appropriate for use in all tasks. Table 9 and Table 10 show an overview of the discussed data processing techniques, along with their memory complexity, time complexity as well as their advantages and disadvantages.

#### 4.2.2.1 Detrending

For real world implementations of WiFi sensing, the environment will inevitably change in some ways which will cause the absolute CSI value to fluctuate over time. Such fluctuating trends may appear over the course of a single day or in longer running systems over multiple weeks. We find that very few WiFi sensing works consider these long-term variations because the current research systems are typically used for short periods of time and in controlled scenarios. WiFi-Sleep [137] on the other hand requires CSI collection to occur throughout a full sleep cycle for an entire night which means that drift is more likely to be observed in the measured signal.

The first method for removing drift by detrending is to fit a least squares regression line [137]. After fitting this baseline, the difference between the original signal and this baseline is calculated. WiFi-Sleep for example finds that data trends non-linearly over an eight hour experiment, thus a higher-order polynomial curve is selected as the baseline. However, finding the least squares baseline at the end of the sleep period precludes the real-time online system that we target in this study.

An alternative approach is to perform Moving Average Trend Removal [138] where; similar to the previous method, a baseline $b[t]$ is calculated and removed from the original signal. To accomplish this in real time, a rolling baseline is calculated as $b[t] = \frac{1}{w}\sum_{i=0}^{w-1} h[t-i]$ where $w$ is an important variable which will determine how well

Table 9. Overview of data preparation techniques (Detrending and Interpolation).

| | **Technique** | **Complexity per Frame** | | **Sources** | **Advantages** | **Disadvantages** |
|---|---|---|---|---|---|---|
| | | **Memory** | **Time** | | | |
| ***Detrending*** | Least Squares Baseline Removal | $O(wS)$ | $O(wS)$ | [137] | High quality over long timeframes. | Requires hours of data collection before processing. |
| | Moving Average Trend Removal | $O(wS)$ | $O(wS)$ | [138] | Real-time detrending. | Poor quality with longer timeframes. |
| ***Interpolation*** | Linear Interpolation | $O(S)$ | $O(S)$ | [139],[140],[141] | Anomalies are reduced through averaging. | Alters real CSI through averaging. |
| | Nearest Neighbor | $O(S)$ | $O(S)$ | [142, 143] | Simple implementation. Retains real CSI values without alterations. | Anomalies may propagate over time. |

Table 10. Overview of data preparation techniques (Segmentation and Feature Scaling).

| | Technique | Complexity per Frame | | Sources | Advantages | Disadvantages |
|---|---|---|---|---|---|---|
| | | Memory | Time | | | |
| ***Segmentation*** | Fixed Window | $O(wS)$ | $O(wS)$ | [113, 134] | Simple. Default method for ML. | Constant model inference, high computation use. |
| | Statistical Window | $O(wS)$ | $O(wS)$ | [14],[144],[145] | Low computation statistical functions. | Movements are lost with poorly selected threshold. |
| | Sentinel Detection | $O(wS)$ | Dependent on Classifier | [118, 146] | Uses lightweight ML sentinel model. | Requires user to initiate sensing period. |
| ***Feature Scaling*** | Max-Min Normalization | $O(S)$ | $O(S)$ | [147, 63] | Constrains data to exact bounds. | Outliers cause issues with data distribution. |
| | Z-Score Standardization | $O(S)$ | $O(S)$ | [148, 149] | Constrains subcarriers to same scales. | Outliers may cause model confusion. |
| | Quantization | $O(S)$ | $O(S)$ | [150] | Reduces ML model size. Increases inference rate. | Reduces data resolution. |

the baseline matches to the actual drift appearing in the signal. In this work, we find that the experiments that we perform do not result in noticeable levels of drift, thus we do not evaluate these detrending methods. However, more work into detrending methods will be required for real world online WiFi sensing system implementations.

#### 4.2.2.2 Interpolation (of Missing Frames)

Due to the wireless communication method used for WiFi sensing, *CSI sampling jitter* can occur due to packet loss or even due to computation delays from the multi-process operating systems used [120] as well as the bursty nature of WiFi communication [151]. As a result, the timestamp for received CSI samples will not be exactly equally spaced. To account for this, the most common technique is to apply *linear interpolation* [139] such that

$$\hat{h}[t] = h[t-1] + \left(\hat{\mathcal{T}}[t] - \mathcal{T}[t-1]\right) \frac{h[t] - h[t-1]}{\mathcal{T}[t] - \mathcal{T}[t-1]}, \tag{4.14}$$

where $t$ is the index of the current time instance, $h[t]$ is the actual CSI value at the actual time $\mathcal{T}[t]$ and $\hat{h}[t]$ is the interpolated CSI value for the interpolated time $\hat{\mathcal{T}}[t]$. Alternatively, *nearest neighbor interpolation* [142], which is computed as

$$\hat{h}[t] = \begin{cases} h[t] & \text{if } \left|\hat{\mathcal{T}}[t] - \mathcal{T}[t]\right| < \left|\hat{\mathcal{T}}[t] - \mathcal{T}[t-1]\right| \\ h[t-1] & \text{otherwise,} \end{cases} \tag{4.15}$$

has also successfully been applied for WiFi sensing tasks. Both of these interpolation techniques can be achieved with $O(1)$ time complexity and $O(1)$ memory complexity per subcarrier because only the current CSI sample and the previous CSI sample are required. Fig. 10 shows an example where our system was set to transmit and receive CSI frames every 10ms (100Hz). We can see that the majority of the frames appear at 10ms, but there is some probability that the frame will appear slightly

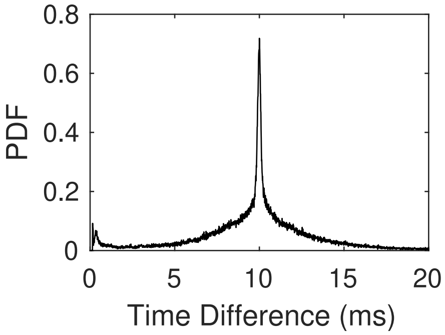


Fig. 10. Probability distribution function showing the timestamp difference between consecutively recorded CSI frames when transmitted at a sampling rate of 100Hz.

earlier or slightly later than every 10ms. When interpolation is applied to a stream of CSI samples, an interpolated sample will be generated for every interpolation interval (i.e., every 10ms) whether or not the CSI sample arrived early, late or exactly on time or even if the CSI sample was missing entirely.

#### 4.2.2.3 Segmentation

As CSI samples arrive at the RX, the system must make a decision: should the samples be used to make a prediction or should the samples be ignored? We find that most of the studies assume that all CSI samples should be used for prediction making. To accomplish this, these studies (e.g., [152, 134]) use a *fixed window* where a window of size $w$ CSI samples are input into the classification algorithms with a step size of $s_{\text{step}}$ which indicates how many CSI samples should be collected between subsequent predictions. However, the human target may not be performing any physical actions at all times of the day. To account for this, a special predicted output class of *"none"* or *"empty"* is often used (e.g., in [153]) for indicating when no actions are recognized by the system. This simplifies the structure of the overall system but can result in an overwhelming number of useless predictions. We argue that this is an even more pressing problem for low-resource embedded systems because the full inference workflow can be both time consuming and most importantly highly energy consuming. Furthermore, because we can assume that in real world systems the *"none"* class may occur with much higher frequency than other actions, the model may become overfit to this *"none"* class which will cause class imbalance due to oversampling of that single class. However, we find in some studies such as [153] that each of the actions used to train the classifier model is given equal amounts of training and testing data per class including the *"empty"* class. Evaluating without considering class imbalance is unrealistic.

Some studies have attempted to reduce how often predictions need to be made in their systems through *segmentation*. Segmentation helps identify the starting points and the ending points for potential actions. It is usually assumed that each action segment will be neighbored directly by a time period of static environmental CSI collection before and after the action. Thus, simple rolling window statistical thresholds are used to detect changes in state. Most commonly, we find that moving-window variance [154] is used to indicate the starting points and ending points of individual physical activities. The idea is that when an activity is being performed, the physical movements cause more noise from appearing in the CSI signal compared to the static environment both before and after the activity is performed. However, this method assumes that activities will always be surrounded by distinct periods without any movements, which may not be the case in practice because, for example, a walking activity may immediately be followed by another activity such as sitting down. Furthermore, when more than a single person is in the environment, activities are likely to overlap, preventing static periods from appearing in the environment. An opposite approach is to use a *motion detector* algorithm [21] to recognize when large motions are being performed in the environment and filter out any CSI collected during these large motions. This approach especially can be helpful if we are trying to monitor very small actions such as respiration as it can remove the unrelated large motions that are likely to overpower the smaller movements that are being tracked.

One other segmentation method commonly employed is to use a specific *start and stop sentinel movement* where the user must perform a given action such as moving their hand close and far away from the receiver multiple times [146]. These sentinel movements should be easy to classify by the device with low computational complexity algorithms. After performing the sentinel movement though, the device can be *"woken up"* and can begin to compute higher complexity algorithms such as deep learning inference. This is more useful for gesture recognition tasks and smart home type interactions where a user is actively involved with interactions with the system rather than passive sensing tasks such as surveillance.

#### 4.2.2.4 Feature Scaling

When data is input into machine learning algorithms, the relative magnitude of each input dimension can have great effect on the overall prediction ability of the algorithm. If any dimension has much higher magnitude than other input dimensions, the contribution of this dimension may begin to overshadow the other dimensions. The process of equalizing input dimensions is called normalization. The most common type of normalization is max-min normalization [147] where given a signal $h$ of length $T$, the normalized value $\hat{h}[t] = \frac{h[t]-\min(h)}{\max(h)-\min(h)}$. This compresses the signal range such that $0 \leq \hat{h}[t] \leq 1, \forall t \in \{1, 2, \ldots, T-1, T\}$ a condition which holds true only if all elements in $h$ are known fully upfront such as in an offline system. An alternative approach is to use Z-score standardization with $\hat{h}[t] = \frac{h-\mu(h)}{\sigma(h)}$ where $\mu(h)$ is the mean value of the signal and $\sigma(h)$ is the standard deviation value of the signal. Z-score standardization is more forgiving in that it allows $\mu(h)$ and $\sigma(h)$ to be computed on smaller subsets of the signal (i.e., from an initial calibration phase). Feature scaling is an important factor in machine learning on low-resource embedded devices. Namely, feature input must be scaled and also quantized to better reduce the computation and memory complexity of the algorithms. Quantization [155] can reduce the number of bits used for encoding machine learning model weights, model input as well as model output. For example, machine learning systems will typically encode numerical values as 32-bit floating point values. These values can be quantized down to 16-bit or 8-bit representations to (i) reduce the memory usage of the machine learning model; (ii) reduce computation time especially in cases where Single Instruction, Multiple Data (SIMD) instructions are available; and (iii) can thus reduce energy consumption.

### 4.2.3 Prediction Making

With the CSI signal data processed and prepared for machine learning, next we can use the data to make predictions about the environment.

#### 4.2.3.1 Classification and Machine Learning

Through our survey of WiFi sensing systems, we find that many different classification algorithms have been used for a wide range of tasks from simple linear and non-linear regression algorithms, and similarity based algorithms such as $k$-nearest neighbors (KNN) and dynamic time warping (DTW), to machine learning algorithms such as support vector machines (SVM) dense neural networks (DNN), convolutional neural networks (CNN), as well as deep learning algorithms such as recurrent neural networks (RNN) and long short-term memory networks (LSTM) and generative adversarial networks (GAN). While there are many common signal processing techniques used throughout WiFi sensing literature, the number of unique classifier model architectures is much larger and more diverse. While DNN and CNN may be used to describe a given model, the architecture of the model can still be structured in a great number of configurations. For example, a CNN model may begin with an arbitrary number of convolutional layers which capture spatial features in the CSI data which are then passed into an arbitrary number of standard dense layers which capture patterns from the extracted features. Each of these layers can have unique hyperparameters such as number of neurons, filter-size, activation function and a growing number of other hyperparameters.

Existing surveys such as [98] and [99] consider the use of deep learning for WiFi sensing and wireless sensing tasks, however, we are focused on performing inference on low-resource microcontrollers which have not yet been considered in these surveys or in existing research literature. As such, we focus on common steps used for running machine learning models on-board constrained microcontroller devices rather than focusing on machine learning model training methods or architecture designs.

When using any type of classification model in a constrained microcontroller device, it is important to be considerate of memory consumed by the model, computation time required for the model and finally energy usage of the model. Increasing the model size increases both memory consumption as well as computation time which subsequently increases energy consumption. As such, we can decrease all three by reducing the size of the machine learning model. Indeed, a number of methods have been used outside of the WiFi sensing research area to decrease the memory consumed by machine learning models in microcontroller environments.

A simple first step towards designing a machine learning model that is applicable in low-resource microcontrollers is to begin with an efficient architecture such as SqueezeNet [156], MobileNets [157] and EfficientNet [158] which are designed for use in embedded vision tasks. Considering the size and complexity of the selected model architecture before training and evaluation can greatly reduce the amount of work that must be performed afterwards.

In cases where an inefficient model is selected initially which either does not fit in memory or does not produce predictions quickly enough, model compression can be performed. A number of approaches have appeared in the research literature including quantization [155], pruning [159], knowledge distillation [160], and model weight sharing [161].

Quantization [155, 150] is one of the most common compression methods which reduces the number of bits used for encoding numeric values for machine learning models, thus allowing model weights to be stored in a more compact space. Two methods for training quantized models have been explored in the research literature: post-training quantization and quantization-aware training (QAT). The first method trains the given model like normal using standard 32-bit floating point computations. After training, the model weights are compressed for use in the inference phase. However, because the model is trained without considering these compressed representations, quantization error can add up resulting in increased error from the model. Instead, QAT uses these compressed numerical representations during the training phase which allows the model to become less prone to errors introduced by post-training quantization. Binarized Neural Networks [162] take the idea of quantization to its limits by compressing all numeric values in the model to a single bit indicating $+1$ and $-1$ values during both inference and training phases.

Another method for compressing model architectures is to perform network pruning [159, 150] which can be thought of as a three-step pipeline. First, a large and inefficient model is selected and trained. Next, some set of pruning algorithms are used to identify and remove useless connections in the network while retaining important weights and connections thus reducing the amount of computation that must be performed. The third and final step is to retrain the pruned model to fine-tune the model with this updated architecture. However, it has been suggested in [163] that the three-stage pruning pipeline can be bypassed by simply starting with the smaller pre-pruned architecture with randomly initialized weights. This suggestion leads us back to the idea of beginning with an efficient model architecture from the beginning.

To account for the additional considerations required for designing and evaluating machine learning models on microcontrollers, it is suggested in [164] that the hardware used for model deployment should not be an afterthought when evaluating machine learning models. Instead, factors like on-board computation time and model size should be combined with model accuracy while evaluating the model to ensure that the model is optimized not only for prediction accuracy, but also for the constraints of the embedded system. To achieve this, on-device benchmarks should be performed continuously to capture real-world metrics for memory usage, computation time when using techniques like SIMD as well as energy consumption. This is particularly important when using automated evaluation methods such as Neural Architecture Search [165] which attempts to increase model accuracy by evaluating numerous diverse model architectures automatically.

#### 4.2.3.2 State Validation

State validation is used to ensure that the predictions that are output by the classifiers are valid and possible given the previous predictions made by the system.

*Finite State Machine (FSM):* FSMs can be used to track the process of different physical properties over time. For example, in [166], the FSM tracks the beginning and ending of each individual step as a person walks. When the FSM enters specific states, the event is shared with a separate module to estimate the stride length of the person.

*Markov Chain:* Similar to the FSM approach, a Markov chain (MC) can be used to understand the probability that a transition will occur at any given time. In [77], MCs are used to track the periodic breathing behaviour of the participant while [148] uses MCs to track longer-term human activity transitions such as transitioning from sitting to standing and walking.

Overall, while state validation may be used to ensure the legitimacy of predictions, state validation is not a very common step used throughout WiFi sensing literature.

#### 4.2.3.3 Voting

Voting accomplishes a similar aim as state validation in that it attempts to improve the validity of the predicted actions. However, voting will use the predictions from multiple unique classifiers to improve the overall accuracy of the prediction system.

*Majority Voting:* A simple method to achieve consensus with multiple classifiers is to take a "majority vote". For the work in [167] and [152], one classifier is trained per TX-RX antenna pair, then a majority voting scheme is used to make predictions. In [168] majority voting is used based on a set of binary classifiers which can be simpler than multi-class classifiers, however, as the number of classes increases, many more binary classifiers must be added to the system which would only increase the complexity of the system.

*Ensemble Learning:* Another popular method for voting is to use a process such as Boosting and Bootstrap Aggregation (Bagging) that specifically trains ensembles of classifiers using a different population of training data per classifier. Through this method, ensemble learning can ensure that any classifiers with poor prediction power on a given subset of data samples are supplemented with another classifier which is specifically trained on these hard-to-predict samples. In [44], this is used to train an ensemble of SVM models for gesture recognition.

### 4.2.4 Systems and Hardware

Next, we briefly discuss some systems-level and hardware components that are required for a complete WiFi sensing system.

#### 4.2.4.1 Clock Synchronization

When multiple devices are used in a WiFi sensing system, it is important to keep internal clocks synchronized between each device in the network. For example, highly synchronized clocks are important when using phase from multiple distinct and unconnected devices. A fast phase correction algorithm was proposed in [169], which was used for vehicle speed estimation through the MUSIC algorithm [170]. A wired connection between transmitter and receiver devices as suggested in [171] allows for fine-grained and accurate synchronization, yet a wired connection defeats the purpose of WiFi communication itself and would not be possible with independently mobile TX and RX.

Coarse-grained clock synchronization is easier to accomplish. For example, using a Real-Time Clock (RTC) module like the DS3231 [172], we can achieve clock synchronization accurate to within a few seconds per year. This can be important in large networks of WiFi sensing devices which remain in sleep mode for long periods of time followed by short bursts of TX to RX transmissions such as in [64]. The lower the accuracy of the clock synchronization across devices, the more time devices must wait for paired devices to awake, and thus, the higher energy wasted. The Network Time Protocol (NTP) is used at the beginning of each experiment in both [25] and [173]. Another method is to use the timestamp returned within GPS responses as a source of truth as used in [119].

#### 4.2.4.2 Data Annotation

Annotating WiFi sensing data can be thought of as the process of labelling when physical activities have occurred while collecting the CSI data. Thus, it is an important step for both deploying the system as well as allowing the system to continue to be effective in the face of changes in the multipath within the environment. Recording camera feeds [174, 175] while performing experiments can allow for accurately tracking physical actions while capturing data to train a WiFi sensing model. Wearable sensor can also be commonly used to record baseline measurements, for example, the NeuLog Respiration Belt [56] is commonly used to track health related metrics such as breathing rate. Other works specifically instruct volunteers when to perform different actions through the use of an auditory sound like a beep [176] or a voice cue [148] as well as through tools like a metronome [177].

The clock synchronization component mentioned in the previous section is not only important for keeping WiFi sensing device in sync, but can also be important for data annotation systems which require external sensors such as in [142] which uses NTP to synchronize a VICON camera-based motion capture system with the proposed WiFi sensing system.

#### 4.2.4.3 Device-to-Device Communication

While a few works [178, 179] consider the use of multiple TX/RX pairs, most works in the literature assume only a single TX/RX pair is deployed in a given environment. However, by reducing the hardware cost and performing WiFi sensing at the edge, we can achieve much more scalable systems and thus we can introduce larger networks of WiFi sensing devices. However, by increasing the number of devices we will find additional challenges. Namely, if different pairs of devices are making predictions independently, it is important to coordinate and aggregate their predictions together to achieve a holistic view into the environment that is being sensed. A fundamental feature to accomplish this is device-to-device communication. Luckily, WiFi sensing by definition has the capabilities of performing communication between nodes through the already existing WiFi protocols. Other communication methods are also an option such as Bluetooth which can be found on-board the ESP32 WiFi sensing microcontroller, or LoRa [97] which allows for a larger communication range.

#### 4.2.4.4 Cyber Physical System Integration

The aim of this work is in allowing WiFi sensing to be performed on the edge using low cost embedded devices. By achieving this, we should then be able to integrate these systems into different cyber physical systems. For example, WiFi sensing can be used to track the occupancy of a building to intelligently control HVAC systems. Additionally, by using WiFi sensing to track health related behaviours such as irregular breathing or heart rate as well as falling, WiFi sensing systems could be integrated with emergency alerting systems to rapidly request emergency aid. Of course, with the integration of WiFi sensing into real-world system, we must also consider security aspects [180] necessary for cyber physical systems. While there has been much work in understanding the capabilities of WiFi sensing, very little work has been undergone to successfully integrate WiFi sensing into real existing cyber physical systems. By moving away from non-scalable batch-based systems to edge-based systems, we believe that WiFi sensing can continue to grow into more real-world use cases.

## 4.3 Evaluation of CSI Processing Techniques

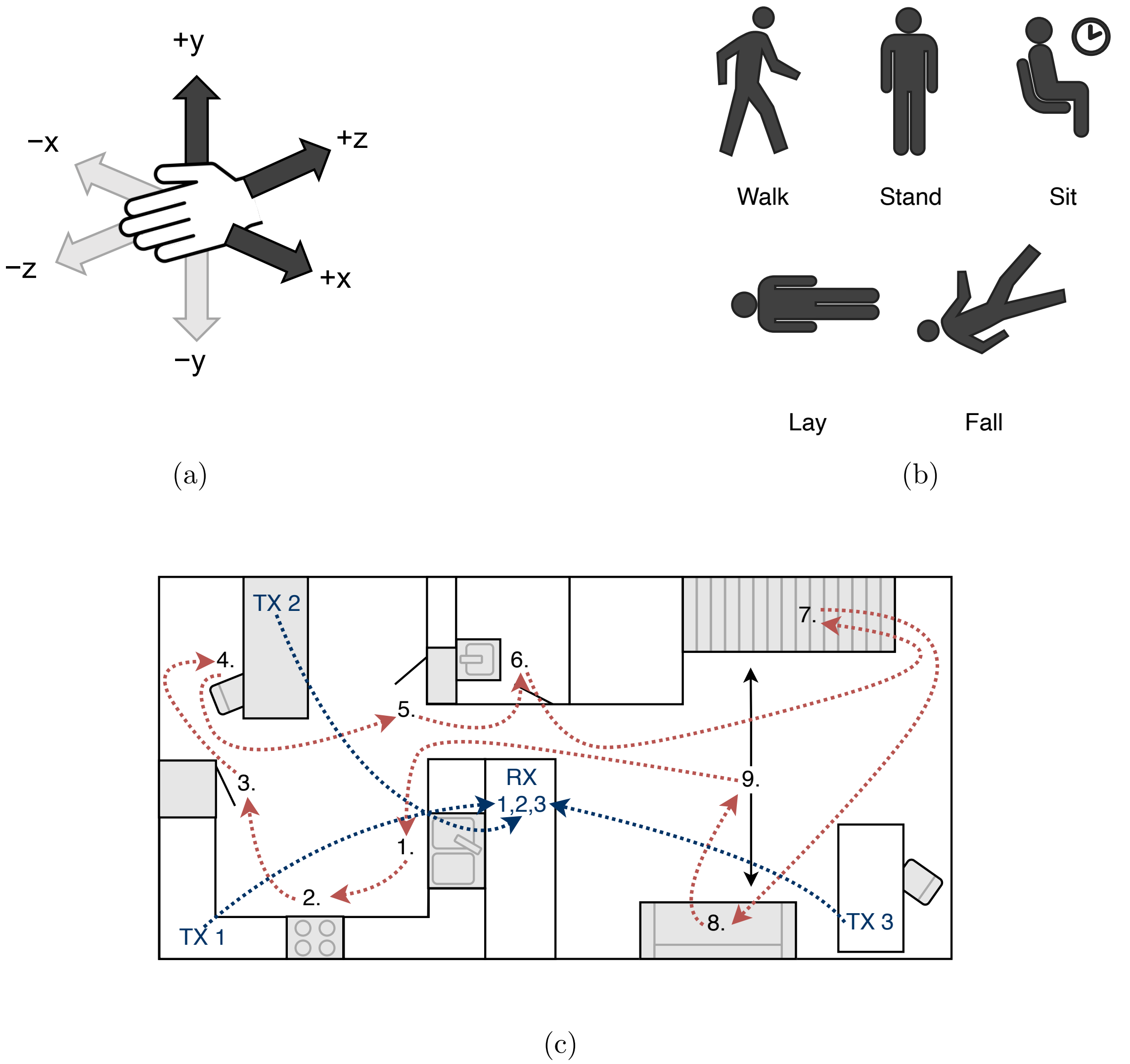
Now that we have identified a number of CSI signal processing techniques through our survey, we next evaluate how well each of these techniques perform with regards to model accuracy. To this end, we evaluate each technique on three experimental scales which represent unique use cases for WiFi sensing. Namely, small-scale hand gesture recognition can be used as a novel device-free method for HCI; medium-scale human activity recognition can be used to track behaviours of a person over some time period; and large-scale human activity and localization sensing can be used to understand human behaviours throughout an entire environment. Through these three diverse applications, we can generalize which techniques achieve high prediction accuracy for different use cases. The three tasks that we evaluate are illustrated in Fig. 11.

### 4.3.1 Experiment Descriptions

For the small-scale experiment, we train our system to recognize hand movements along three axis of motion: Z-axis (push/pull), X-axis (swipe left/right) and Y-axis (raise/lower). For each of these three physical actions, we repeat each hand movement 30 times in round-robin order to ensure that actions are interleaved over time.

The second set of experiments that we perform are for medium-scale human activity recognition. For this experiment, we perform five of the most common actions that we have identified in WiFi sensing studies, namely: walking, standing, sitting, laying, falling as illustrated in Fig. 11b. For these experiments, we repeat all actions 8 times in an interleaved round-robin order as we did in the small-scale experiment.

Finally, the third set of experiments that we perform are for large-scale localization and activity recognition tasks. In this case, we perform actions in nine distinct



(a)

(b)

(c)

Fig. 11. Activities performed for each experiment type. (a) Small-scale hand gesture recognition with three directional gestures. (b) Medium-scale human activity recognition with five different actions. (c) Large-scale human localization and activity tracking in a home environment with nine actions and three transmitter/receiver links.

locations within a home-environment as shown in Fig. 11c. Namely, we perform three actions in the kitchen: (1) wash dishes at the sink, (2) cook on stove-top, (3) open fridge; three actions in the dining room: (4) write in a book at table, (5) open and close closet door, (6) wash hands in washroom; and three actions in the living room: (7) walk up and down stairs, (8) sit on sofa, (9) walk around. For this experiment we collect data from three distinct TX-RX pairs. In our initial evaluation of these large-scale experiments we use the second TX-RX pair by itself to predict all nine actions. Later on in this chapter, we will evaluate methods for leveraging predictions from multiple devices within a single environment.

Additional information about the experimental setting are shown in Table 11. For example, for all three scales, we collect CSI at a sampling rate of 100Hz. Additionally, each experimental scale has a progressively larger sensing area where actions are performed.

### 4.3.2 Hyperparameter Optimization

For our initial evaluation of each of the three experimental scales, we wish to identify how feature extraction, denoising and dimensionality reduction techniques affect the accuracy of our model. Different hyperparameter settings such as learning-rate, number of hidden neurons and regularization methods may result in varying prediction accuracy for each methods. Additionally, each technique itself has a unique set of hyperparameters which must also be tuned to achieve better model accuracy for the given task. Table 12 shows the list of hyperparameters and possible hyperparameter-values that we used during our evaluations. The six hyperparameters marked as global are model specific parameters that are present no matter which feature extraction, denoising, or dimensionality reduction technique is used. We can see that the Hampel, window statistical filter and Savitzky Golay denoising methods each have

Table 11. Description of the three device-free experiments performed and evaluated using CSI collected from ESP32s.

| Scale | Sensing Task | # Actions | # Repetitions | # Links | Sampling Rate | Sensing Area | Figure |
|---|---|---|---|---|---|---|---|
| Small-Scale | Gesture Recognition | 3 | 30 | 1 | 100Hz | 1.0m × 1.0m | Fig. 11a |
| Medium-Scale | Human Activity Recognition | 5 | 8 | 1 | 100Hz | 2.5m × 4.0m | Fig. 11b |
| Large-Scale | Localization and Activity Recognition | 9 | 13 | 3 | 100Hz | 5.0m × 10.0m | Fig. 11c |

window-size as a common hyperparameter, however each of these methods also has other technique-specific hyperparameters as well. Performing a grid-search to evaluate every possible combination of hyperparameters would be infeasible. Instead we use the Tree-structured Parzen Estimator (TPE) using the Optuna framework [181] where we perform 100 trails, each with a uniquely selected set of hyperparameters. In the first few trials, hyperparameter values are selected randomly from the set of options shown in Table 12. Subsequent trials with TPE use the results of previous trials to guide the hyperparameter optimization towards maximizing the model accuracy. For each trial, the models train for 100 epochs, however to reduce the search time spent training the model on non-optimal hyperparameter values, trials are pruned early (i.e., before 100 epochs) if the trial validation accuracy is below the median validation accuracy of all previous trials.

Table 13 shows the result of this hyperparameter optimization method when evaluated on five feature extraction methods, five denoising methods and four dimensionality reduction methods for each of the three scales of experiments. Through this method, we perform a total of $4,200$ independent hyperparameter optimization trials. At the top of the table, we can see the accuracy of a randomly guessing model for each experimental scale based on the number of actions performed at each scale. Specifically, the small-scale would achieve an accuracy of 33.33% with three actions, medium-scale would achieve an accuracy of 20.00% with five actions, and large-scale would achieve an accuracy of 11.11% with nine actions.

### 4.3.3 Independent Evaluation of Each Method

In this study, we begin by evaluating each signal processing technique independently to recognize if there are any techniques which clearly perform better across the board. For example, when evaluating feature extraction methods, we do not apply

Table 12. List of hyperparameters and possible values used during hyperparameter optimization.

| Technique | Parameter Name | Values |
|---|---|---|
| Global | Input Window Size | $\{25, 50, \ldots, 475, 500\}$ |
| Global | Learning Rate | $\{1e^{-9}, 1e^{-8}, \ldots, 1e^{0}, 1e^{1}\}$ |
| Global | # Hidden Neurons | $\{25, 50, \ldots, 475, 500\}$ |
| Global | Droput | $\{0.0, 0.1, \ldots, 0.8, 0.9\}$ |
| Global | Kernel Regular. | $\{\mathrm{True}, \mathrm{False}\}$ |
| Global | Activity Regular. | $\{\mathrm{True}, \mathrm{False}\}$ |
| Hampel | Window Size | $\{50, 100, \ldots, 450, 500\}$ |
| Hampel | Threshold | $\{0.25, 0.5, \ldots, 3.75, 4.0\}$ |
| Win. Stat. Filter | Window Size | $\{50, 100, \ldots, 450, 500\}$ |
| Win. Stat. Filter | Stat. Function | $\{\mathrm{Mean}, \mathrm{Median}, \mathrm{STDev}, \mathrm{Var.}\}$ |
| Savitzky Golay | Window Size | $\{51, 101, \ldots, 451, 501\}$ |
| Savitzky Golay | Poly. Order | $\{1, 2, \ldots, 8, 9\}$ |
| Butterworth | Order | $\{1, 2, \ldots, 10, 11\}$ |
| Butterworth | Frequency | $\{1, 2, \ldots, 49, 50\}$ |
| Butterworth | Type | $\{\mathrm{Lowpass}, \mathrm{Highpass}\}$ |
| DWT | Threshold | $\{0.0, 0.25, \ldots, 3.75, 4.0\}$ |
| DWT | Mode | $\{\mathrm{Hard}, \mathrm{Soft}\}$ |
| Subcarrier Stat. | Max/Min | $\{\mathrm{Max}, \mathrm{Min}\}$ |
| Subcarrier Stat. | Stat. Function | $\{\mathrm{Mean}, \mathrm{Median}, \mathrm{STDev}, \mathrm{Var.}\}$ |
| Subcarrier Corr. | Max/Min | $\{\mathrm{Max}, \mathrm{Min}\}$ |

any denoising or dimensionality reduction techniques. However, when we evaluate denoising and dimensionality reduction, we keep the amplitude feature as the default because it is so commonly used throughout the research literature and because without it, the raw CSI is essentially meaningless.

Amplitude and PSD feature extraction methods achieve the highest prediction accuracy for all three scales. For medium-scale, this is significant at 86.00% and 76.12% accuracy respectively, but for small-scale and large-scale, neither method alone can surpass even 55% accuracy. This shows that using feature extraction techniques alone may not be sufficient for all types of tasks. It also shows that the actions performed during the medium-scale experiments are easier to distinguish than the small-scale and the large-scale experiments.

Moving to denoising techniques, we can see that once again, medium-scale is able to achieve greater than 80% accuracy for all denoising methods except for the Hampel filter and the FFT frequency filter. We must note however that while these denoising methods achieve good accuracy values; window statistical filter performs worse than when using amplitude without a denoising filter and DWT achieves essentially the same prediction accuracy. For the small-scale experiment, all denoising methods increase the accuracy compared to no denoising method except for the Savitzky Golay filter and the FFT frequency filter. The window statistical filter increases the accuracy the greatest by +9.89%. Three out of six denoising methods increase the accuracy for the large-scale experiment, namely Hampel filter (+0.83%), window statistical filter (+5.51%), and Savitzky Golay (+3.59%). FFT frequency filter performs consistently much worse than all other denoising methods likely due to the fact that it is calculated independently over each CSI frame rather than being calculated over a window of CSI frames. None of the evaluated denoising methods performs better in all three experimental scales. In fact, for all denoising methods, at least one of the experimental

Table 13. Comparison of feature extraction methods, denoising filter and dimensionality reduction methods on the prediction accuracy for medium scale human activity recognition.

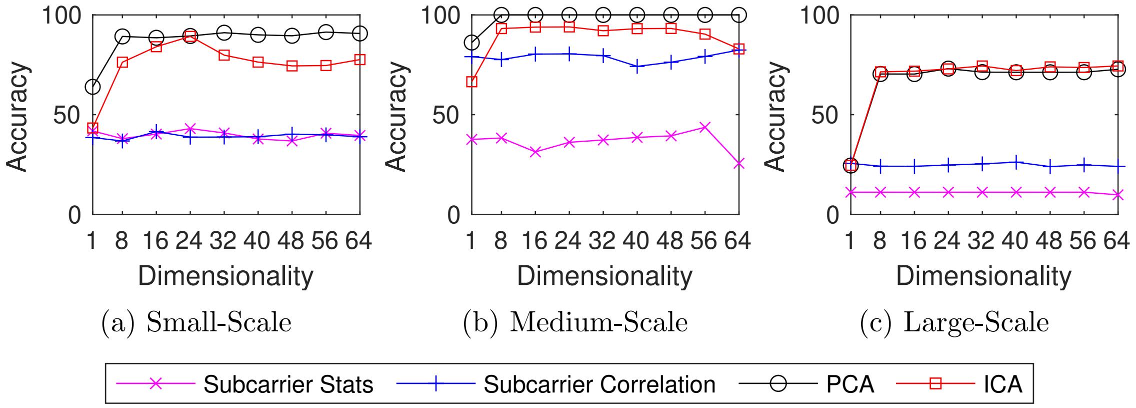
| Feature Extraction | Denoising | Dimensionality Reduction | Accuracy | | |
|---|---|---|---|---|---|
| | | | Small Scale | Medium Scale | Large Scale |
| Random Guess | | | 33.33% | 20.00% | 11.11% |
| Default: Amplitude | Default: None | Default: None | 40.33% | 86.00% | 53.99% |
| Phase | | | 36.01% | 25.05% | 11.81% |
| Amplitude (Diff.) | | | 36.36% | 26.00% | 40.02% |
| Phase (Diff.) | | | 34.75% | 24.76% | 23.40% |
| PSD | | | 40.75% | 76.12% | 48.94% |
| Default: Amplitude | Hampel | Default: None | 46.85% | 76.76% | 54.82% |
| | Window Stats Filter | | 50.20% | 81.31% | 59.50% |
| | Savitzky Golay | | 38.81% | 97.22% | 57.58% |
| | Butterworth | | 41.65% | 89.57% | 52.98% |
| | DWT | | 41.47% | 86.06% | 53.41% |
| Default: Amplitude | Default: None | Subcarrier Stats | 38.91% | 36.70% | 13.09% |
| | | Subcarrier Correlation | 36.69% | 64.87% | 26.05% |
| | | PCA | 87.36% | 100.00% | 71.24% |
| | | ICA | 81.82% | 87.26% | 72.79% |

scales results in a decrease in model accuracy compared to the baseline of using just amplitude. This shows that denoising methods are use-case specific and will not be guaranteed to provide improved accuracy.

Finally, we move on to evaluating dimensionality reduction techniques which have consistent results across each experimental scale. Using subcarrier statistics for dimensionality reduction achieves only a small improvement to the accuracy of a randomly-guessing model, thus subcarrier statistics are not a wise choice for dimensionality reduction. Using subcarrier correlation for dimensionality reduction achieves a slightly higher accuracy for both medium-scale and large-scale, but the prediction accuracy is still significantly lower than using the baseline amplitude without dimensionality reduction. Finally, PCA and ICA achieve the highest accuracy among all evaluated techniques across the three experimental scales. For small-scale and medium-scale, we find that PCA performs significantly better results than ICA, while in the large-scale experiment, both PCA and ICA achieve approximately the same prediction accuracy. Based on these results, PCA achieves the highest accuracy for all experimental scales.

### 4.3.4 Dimensionality Reduction

When we evaluate the four dimensionality reduction methods with hyperparameter optimization, we define $d = 10$ such that we reduce the CSI data from a subcarrier vector of size 64 down to a subcarrier vector of size 10. In Fig. 12, we use the same optimal hyperparameters found for Table 13, but we change the value for $d$ to understand how dimensionality affects the accuracy of the model. For all three scales, we can see that Subcarrier Statistics and Subcarrier Correlation each achieve the same accuracy no matter how the value for $d$ changes. For PCA and ICA, we can see that the accuracy starts low when $d = 1$ but quickly reaches a plateau by $d = 8$ where

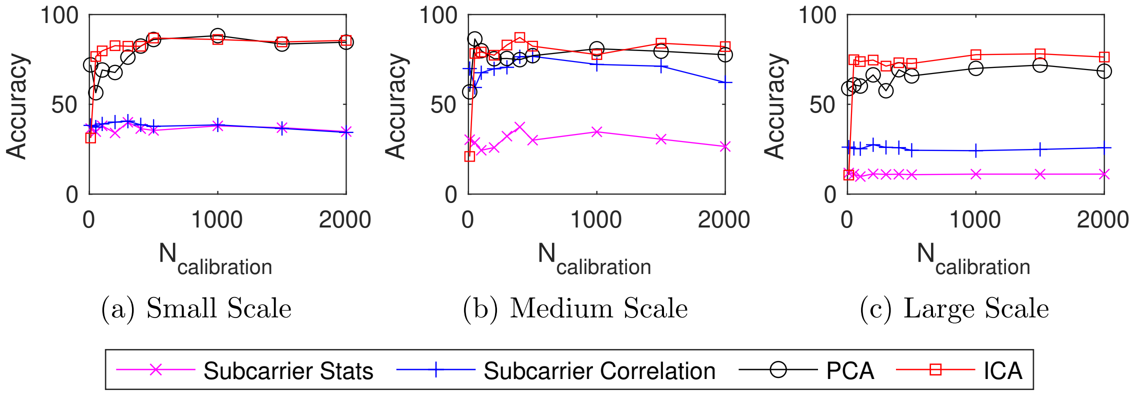


(a) Small-Scale (b) Medium-Scale (c) Large-Scale

Fig. 12. Accuracy of dimensionality reduction techniques when dimensionality ($d$) changes.

the accuracy remains relatively stable while $d$ continues to increase. However, in the small-scale evaluation we find that ICA reaches a peak accuracy when $d = 24$, but afterwards, the accuracy decreases. This implies that increasing the dimensionality with ICA results in more noisy components which makes it harder for the model to distinguish different CSI samples. PCA on the other hand exhibits robustness even as $d$ increases. In this case, PCA seems like the better choice for dimensionality reduction.

In Section 4.2, when we discussed different signal processing techniques we focused primarily on the steps that run on-device for every incoming CSI sample. For each of the dimensionality reduction methods, there is also a calibration phase which runs only when the device is deployed in the environment. For example, when subcarrier statistics are used for dimensionality reduction, we need to collect some number ($N_{\text{calibration}}$) of CSI samples to calculate some statistics of each subcarrier. Similarly, for PCA and ICA, we calculate a components coefficients matrix with $N_{\text{calibration}}$ CSI samples. We can assume that the calibration phase only runs once when the system is deployed, so we do not need to worry about the exact time-complexity of the algo-



(a) Small Scale (b) Medium Scale (c) Large Scale

Fig. 13. Accuracy of dimensionality reduction techniques when different number of CSI-samples are used to calibrate the technique.

rithm. However, we should consider how many CSI samples ($N_{\text{calibration}}$) are required to successfully calibrate each dimensionality reduction method. In the experiments shown in Table 13 we allow each method to calibrate on 100% of the available training data and 0% of the testing data. It is important to not allow the model to see any data from the testing data for fairness. In Fig. 13, we evaluate the accuracy as we change $N_{\text{calibration}}$. At 100Hz, the range for the $x$-axis of this plots shows a maximum of 20 seconds of calibration data when $N_{\text{calibration}} = 2,000$. Similar to Fig. 12, only PCA and ICA show variation as parameters change. Specifically, as $N_{\text{calibration}}$ increases, the accuracy also increases but after $N_{\text{calibration}} = 500$, we can see that the accuracy flattens out. This shows that we can calibrate our system in a new location with only $5 - 10$ seconds worth of CSI-samples which can be achieved quickly and passively. However, while small-scale and large-scale achieve approximately the same accuracy in both Table 13 and Fig. 13 when $N_{\text{calibration}} = 2,000$, medium-scale only achieves 77.6% for PCA when $N_{\text{calibration}} = 2,000$ compared to 100.0% in Table 13. This shows that while sufficient prediction accuracy is possible with low values for $N_{\text{calibration}}$, if we are able to collect more data, we might be able to increase the accuracy slightly.

Table 14. Effect of interpolation on model accuracy.

| | Small-Scale | Medium-Scale | Large-Scale |
|---|---|---|---|
| None | 87.36% | 100.00% | 71.24% |
| Nearest Neighbor | 90.06% | 100.00% | 67.18% |
| Linear | 89.34% | 100.00% | 68.97% |

However, increasing $N_{\text{calibration}}$ results in an increase in the time to collect CSI data for calibration and also increases the computation time required during the calibration phase.

### 4.3.5 Interpolation

Interpolation has been used in other studies to account for sample jitter due to packet loss or computation delays. In Table 14, we compare the accuracy achieved by two such interpolation methods: *nearest neighbor* and *linear* interpolation as well as the accuracy achieved without applying interpolation (i.e., *none*). We used the optimal hyperparameters identified when using Amplitude for feature extraction and PCA for dimensionality reduction. We find that interpolation does not result in much change in the accuracy for any of the experimental scales. As we showed in Fig. 10, the time difference between subsequent CSI samples collected using our system is relatively precise, meaning that interpolation is not needed when CSI is received at a constant rate of 100Hz. Interpolation may be more important when the CSI sampling rate is not controlled by the system or if environmental noise results in higher packet-loss.

Table 15. Effect of feature scaling on model accuracy.

| | Small-Scale | Medium-Scale | Large-Scale |
|---|---|---|---|
| None | 87.36% | 100.00% | 71.24% |
| Max-Min Normalize | 83.80% | 99.51% | 69.00% |
| Z-Score Standardize | 70.39% | 97.43% | 53.19% |
| Quantize | 69.99% | 99.92% | 55.14% |

### 4.3.6 Feature Scaling

Feature scaling can be used in machine learning workflows to ensure that features with a large range of values do not overshadow features with a smaller range of values. In Table 15, we compare three feature scaling methods: *max-min normalization*, *z-score standardization*, and *quantize* with a model trained and evaluated without feature scaling (i.e., *none*). As a baseline, we again use the optimal hyperparameters identified when using Amplitude for feature extraction and PCA for dimensionality reduction. We apply feature scaling using statistical metrics taken from the entire CSI matrix. For example, with max-min normalization, we find a single maximum value and a single minimum value using CSI measurements across all time instances and all subcarriers. Typically, normalization and standardization will find these statistical metrics individually for each feature (i.e., subcarrier or PCA component) to prevent individual features with higher ranges from overshadowing features with smaller ranges of possible values. However, with PCA, we find that the first component has the largest range of values followed by the second component and then the third component and so on. This is expected and beneficial because each subsequent component is known to have less important information. Scaling each PCA component independently removes this relative scale and makes it harder for the model to understand the relative importance of each component and thus, it is harder for the model to make accurate predictions. As such, we find that feature scaling does not result in an improvement to the prediction accuracy of the model and in fact results in decreased prediction accuracy for all experimental scales.

## 4.4 Evaluation of Edge-Based WiFi Sensing System

Now that we have reviewed the accuracy of our ESP32-based WiFi sensing system when used for WiFi sensing tasks ranging from small-scale hand gestures to medium-scale human activity recognition up to large-scale localization and activity recognition, we next evaluate the feasibility of deploying our WiFi sensing hardware into real-world scenarios. Specifically, we consider how WiFi sensing can be deployed on resource-constrained ESP32 microcontrollers. To judge the feasibility of this ESP32 system, we must evaluate both the hardware and the software running on-board. We begin by considering the rate at which the ESP32 can compute different signal processing steps as well as machine learning prediction (i.e., on-board model inference rate). We compare these rates to the inference rates achieved in other WiFi sensing studies in the literature. Finally, we review energy consumed by each component of the system to understand the feasibility of deploying the system at the edge.

### 4.4.1 Effect of Sampling Rates on Accuracy:

In this chapter, we use a sampling rate of $R = 100$Hz as the baseline for our small-scale, medium-scale and large-scale experiments. To understand how sampling rate affects prediction accuracy, we decrease $R$ while using the optimal hyperparameter values as used in Table 13. In addition to reducing the sampling rate, we must also reduce the window size $w$ down to $\hat{w} = \lfloor \frac{wR}{100} \rfloor$ so that each window still covers the same span of time no matter the value of $R$. From Fig. 14, we can see that there

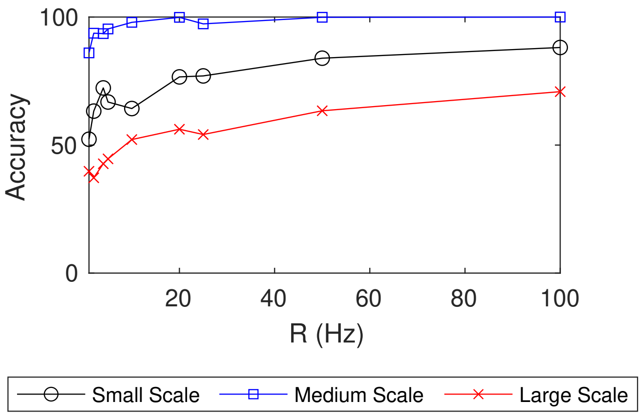


Fig. 14. Decreasing the sampling rate results in lowered accuracy for all experimental scales.

is a general trend where the accuracy decreases if $R$ decreases. For small-scale, we achieve an accuracy of 52.27% and 88.11% when $R = 1$ and $R = 100$, respectively (35.84% difference) while in the medium-scale accuracy is 86.00% and 100.00% when $R = 1$ and $R = 100$, respectively (14.00% difference) and finally large-scale accuracy is 39.77% and 70.86% when $R = 1$ and $R = 100$, respectively (31.09% difference). Medium-scale is the best at handling lower values of $R$ and can even achieve an accuracy of 99.88% when $R = 20$. Small-scale sees the largest decrease in accuracy as $R$ decreases which makes sense considering that the small-scale movements are both small and performed very quickly. From these results, we can say that increasing $R$ will result in an increase in model accuracy. However, we must also recognize that each increase for $R$ will not result in a linear increase in accuracy. For example, increasing $R$ from 10 up to 20 results in an increase of +12.37%, +1.95% and +3.64% for small-scale, medium-scale and large-scale experiments, respectively. However, increasing $R$ from 50 up to 100 only results in an increase of +4.18%, +0.001% and +7.47% for small-scale, medium-scale and large-scale experiments, respectively. This demonstrates that the curves in the figure are non-linear and as such, increasingly higher values for $R$ would be required to continue to push the accuracy higher. Of course very high sampling rates are not reasonable for edge based devices which have low computation resources and low power budgets.

### 4.4.2 Inference Rate with Signal Processing Techniques

Capturing CSI at a consistent rate is the first step towards developing a complete WiFi sensing system on the ESP32. After capturing the CSI, we must then use the CSI to make predictions in the environment. Inference rate indicates the number of samples that can be processed per second. As mentioned in Section 4.2, WiFi sensing systems will begin with signal processing followed by machine learning prediction making. As such, we begin our evaluation by reviewing the rate at which we can compute different signal processing methods directly on the ESP32. After this, we then review the effect of machine learning model architecture on inference rates. Finally, we compare the rates which our system can achieve to the rates achieved in other works.

For each CSI frame received by the system, we can perform signal preprocessing steps to extract certain features, denoise our signal or reduce the dimensionality of the CSI vector. In Tables 16, 17, and 18 we review the computation time for a set of these preprocessing steps when implemented and run on an ESP32. For preprocessing methods with relevant parameters, we also evaluate their effect on computation time and maximum sample throughput rate. We repeat the computation directly on-board the ESP32 10 times and capture the average run-time. Starting with feature extraction (Table 16), we begin by looking at the computation time for transforming the raw data to amplitude and phase. We can see that computing amplitude takes 0.54ms while phase takes just 0.16ms due to the square power required for computing amplitude. Using just these signal preprocessing techniques individually, we can

Table 16. Time to compute each feature extraction method on an ESP32 microcontroller as well as the maximum rate at which each method could be performed independent of other computation tasks.

| Method | Parameters | Time (ms) | Max. Rate (Hz) |
|---|---|---|---|
| Amplitude | None | 0.54 | 1,855 |
| Phase | None | 0.13 | 7,710 |
| Temporal Diff | None | 0.03 | 30,581 |
| Statistical Features | Mean | 0.02 | 64,935 |
| PSD | $w = 16$ | 0.57 | 1,742 |
| PSD | $w = 64$ | 2.11 | 473 |
| PSD | $w = 128$ | 4.38 | 228 |
| Wavelet Transform | $\psi = db4$ | 0.47 | 2,111 |
| Wavelet Transform | $\psi = db5$ | 0.62 | 1,621 |

achieve a maximum throughput of $1,855$Hz and $6,134$Hz, respectively which should both far exceed our CSI sampling rate. However, note that subsequent signal preprocessing steps typically assume that either amplitude or phase is computed beforehand. For example, the temporal difference feature extraction method takes only 0.2ms to compute and thus can be computed at a maximum rate of $63,291$Hz, however if the amplitude feature extraction method was performed first we must consider a summation of computation times for all methods to determine the maximum achievable rate. In this example, computing amplitude takes 0.54ms while computing the temporal difference takes 0.02ms which means that our achievable rate when using both methods together would be $\frac{1,000}{0.54+0.02} = \frac{1,000}{0.56} = 1,785$Hz. Given $N$ signal preprocessing steps where $T_n$ is the time to compute the $n$-th preprocessing step in milliseconds

Table 17. Time to compute each signal denoising method on an ESP32 microcontroller as well as the maximum rate at which each method could be performed independent of other computation tasks.

| Method | Parameters | Time (ms) | Max. Rate (Hz) |
|---|---|---|---|
| Hampel | $w = 10$ | 1.21 | 828 |
| Hampel | $w = 50$ | 4.59 | 217 |
| Hampel | $w = 100$ | 9.51 | 105 |
| Statistical Window Filter | $w = 10$ | 0.40 | 2,528 |
| Statistical Window Filter | $w = 50$ | 1.78 | 561 |
| Statistical Window Filter | $w = 100$ | 3.48 | 287 |
| Savitzky Golay | $w = 10$ | 0.77 | 1,304 |
| Savitzky Golay | $w = 50$ | 3.69 | 271 |
| Savitzky Golay | $w = 100$ | 7.33 | 136 |
| Butterworth | $w = 10$ | 1.42 | 705 |
| Butterworth | $w = 50$ | 7.22 | 138 |
| Butterworth | $w = 100$ | 14.56 | 68 |
| DWT | $\psi = db4$ | 0.72 | 1,390 |
| DWT | $\psi = db5$ | 0.88 | 1,135 |
| FFT Frequency Filter | None | 0.08 | 12,121 |

where $n \in \{1, 2..., N-1, N\}$, the maximum achievable rate can be formally calculated as

$$R_{max} = \frac{1,000}{\sum_{i=1}^{N} T_n}. \tag{4.16}$$

For signal denoising methods (Table 17), each method except DWT and FFT frequency filter uses a window size ($w$) when performing the filtering computation. DWT instead uses different wavelet functions $\psi$ where the lengths are typically much smaller while FFT frequency filter is computed only over a single CSI frame rather than over a window of frames. When $w = 100$, the maximum achievable rates were $105, 287, 136, \text{and}, 68$Hz for the Hampel filter, statistical window filter, Savitzky Golay filter and Butterworth filter, respectively. Out of all of the evaluated signal preprocessing steps, Butterworth with $w = 100$ is the only method which is unable to achieve greater than 100Hz and Hampel with $w = 100$ is the filter with the second lowest rate of 105Hz. This implies that if we want to increase the number of predictions possible per second, we must reduce $w$ for these methods to ensure that the total sample preprocessing time is low enough. With DWT, $\psi = db4$ has a filter length of only $|\psi| = 8$ while $\psi = db5$ has a filter length of $|\psi| = 10$. Even so, because of the recursive nature of the method, the time to compute is relatively high and thus the maximum achievable rate is only 582Hz and 427Hz per wavelet type. Notice, for each CSI sample passed into DWT, the number of decomposition and reconstruction levels may vary as shown in Fig. 9. To reliably calculate the average computation time in our evaluations, we assume the worst case for DWT where $L = 3$ levels of decomposition and $L = 3$ levels of reconstruction are performed for the incoming sample. As such, DWT can be expected to achieve higher rates when run in real world scenarios.

For dimensionality reduction methods (Table 18), we can see that the subcar-

Table 18. Time to compute each dimensionality reduction method on an ESP32 microcontroller as well as the maximum rate at which each method could be performed independent of other computation tasks.

| Method | Parameters | Time (ms) | Max. Rate (Hz) |
|---|---|---|---|
| Subcarrier Stats. | $k = 10$ | 0.01 | 121,951 |
| Subcarrier Stats. | $k = 32$ | 0.01 | 89,285 |
| Subcarrier Stats. | $k = 64$ | 0.02 | 65,359 |
| Subcarrier Correlation | $k = 10$ | 0.01 | 120,481 |
| Subcarrier Correlation | $k = 32$ | 0.01 | 89,285 |
| Subcarrier Correlation | $k = 64$ | 0.02 | 65,359 |
| PCA | $k = 10$ | 0.77 | 1,300 |
| PCA | $k = 32$ | 2.40 | 416 |
| PCA | $k = 64$ | 4.78 | 209 |
| ICA | $k = 10$ | 0.77 | 1,303 |
| ICA | $k = 32$ | 2.40 | 416 |
| ICA | $k = 64$ | 4.79 | 208 |

rier statistics methods and subcarrier correlation methods can be computed faster than any other method. With these reduction methods, the most time-consuming computations are calculated during the initial calibration steps where a subset of subcarriers are preselected based on statistics of each subcarrier. For each incoming CSI sample, we only need to select the $k$ subcarriers that were preselected during this process. This allows dimensionality reduction to be an extremely low-cost operation. PCA and ICA on the other hand both take approximately the same time to compute because the online portion of these algorithms is the exact same computation. The calculations performed during the initial calibration phase is what sets the two methods apart. When $k = 64$, we can see that both methods can only achieve a maximum sampling rate of just over 200Hz, however performing PCA or ICA to transform a 64 subcarrier CSI vector down to a vector of size 64 does not actually achieve dimensionality reduction. In such a case, PCA and ICA will only be acting as denoising methods. Instead, $k$ will typically be less than 64 so that PCA and ICA will not only denoise the incoming signal, but will also reduce the dimensionality and thus increase the throughput of the signal. For example, in Table 13, when evaluating these dimensionality reduction methods, we set $k = 10$, in which case, both PCA and ICA would be able to achieve a maximum rate of $1,300$ Hz or when combined with the amplitude feature extraction method can achieve a maximum rate of $\frac{1,000}{0.54+0.77} = 763$Hz.

### 4.4.3 Inference Rate with On-board Machine Learning

After performing signal preprocessing, we can pass the filtered CSI data into a machine learning classifier model. Throughout our experiments in Section 4.3, we used a DNN with four layers (one input layer, two hidden layers, one output layer) where the hidden layers each contain some number of hidden neurons (we call this number the *hidden size*). The input for the DNN is a matrix of $S \times w$ where $S$ indicates the number of subcarrier dimensions and $w$ indicates a window size parameter where $w$ consecutive CSI samples are collected and passed into the model. We use Tensorflow-Lite[3] (TFLite) which offers a method for running our machine learning models directly on embedded devices such as the ESP32.

The ESP32 microcontroller is a highly resource constrained device with lower available resources than would be expected on a typical ML server used for training and evaluating WiFi sensing models in the existing literature. For example, the ESP32 is limited to a maximum of 240MHz clock rate and 520kB of RAM. Furthermore, by default, the ESP32 only allows for 160kB of storage to be allocated to the Dynamic RAM (DRAM) Heap which is the default method for storing data such as the machine learning model definition as well as the TFLite library implementation. In Table 19 and Table 20, we list the prediction rate achieved along with the model size when run on the ESP32 with two different quantization methods as well as varying number of hidden neuron size and CSI input size. We can see that 19 of the rows do not have an associated prediction rate. This is because the model size was too large to fit in the available DRAM Heap after all other overhead was accounted for. The largest model size that was able to run on-board the ESP32 was 34.3kB when hidden size was set to 10 and input size was $16 \times 50$. It is important to notice that when we use **INT8** quantization, this same model can be reduced from 34.3kB to 10.9kB. This shows that quantization greatly reduces model size on-board the ESP32. Interestingly, in both cases where quantization is used (**INT8**) and where quantization is not used (**NONE**), using quantization counter-intuitively decreases the prediction rate. This is because, while **INT8** quantization reduces the size of the model by converting 32-bit floating point numbers to 8-bit integers for model weights,

[3] https://www.tensorflow.org/lite.

additional quantization-specific layers are automatically added throughout the model architecture which results in additional computation that must be performed for the quantized model compared to the non-quantized model. Out of the 17 models that are able to fit in DRAM, only 6 are possible without quantization while 11 are possible with quantization, showing that quantization is still important to allow for larger machine learning models. However, $16 \times 100$ is the largest input size that was possible in a single case when quantization was **INT8** and the hidden size was a paltry 10. Only a single model evaluated in the table was able to increase the hidden size to 100 hidden neurons, but this was only achievable when the input size was a meager $4 \times 10$ matrix. This shows that by default, the ESP32 is unable to allocate large architecture models with only DRAM.

However, while ESP32 modules by default are limited to 520kB of available RAM and 160kB of compile-time DRAM, some boards offer an additional PSRAM (Pseudo-Static RAM) up to a maximum size of 4MB. By using PSRAM and statically allocated TFLite models, we are able to increase the allowable size for the machine learning model definition and thus increase the hidden size and the CSI input size compared to the default ESP32 without PSRAM. We compare on-board inference rate and model size with different quantization methods, hidden size and input sizes in Table 21 and Table 22. With PSRAM, the largest machine learning model which can be used on-board the ESP32 for edge inference has a size of $3,147.5$kB when quantization is set to **INT8**, hidden size is set to 100, and CSI input size matrix is of size $64 \times 500$. It is important to achieve such high values because the hyperparameters we used during our hyperparameter search as detailed in Table 12 are similarly high, with a maximum of 500 for hidden size, and a maximum CSI input size of $64 \times 500$. It has been suggested in [165] that it can be useful to consider not only increasing the accuracy of the models during hyperparameter optimization, but to also increase

Table 19. TFLite inference rate *without PSRAM* and *without quantization* for different model hyperparameters and quantization methods. Inference rates marked (–) indicate that the model was unable to run on the microcontroller due to memory issues. Only small values for hidden size and input size are used because RAM space is so limited.

| Quantization | Hidden Size | Input Size | Rate (Hz) | Size (kB) |
|---|---|---|---|---|
| NONE | 10 | $4 \times 10$ | 9717 | 4.6 |
| NONE | 10 | $4 \times 50$ | 5054 | 10.9 |
| NONE | 10 | $4 \times 100$ | 3170 | 18.7 |
| NONE | 10 | $16 \times 10$ | 5717 | 9.3 |
| NONE | 10 | $16 \times 50$ | 1832 | 34.3 |
| NONE | 10 | $16 \times 100$ | – | 65.6 |
| NONE | 50 | $4 \times 10$ | 1915 | 29.9 |
| NONE | 50 | $4 \times 50$ | – | 61.2 |
| NONE | 50 | $4 \times 100$ | – | 100.3 |
| NONE | 50 | $16 \times 10$ | – | 53.4 |
| NONE | 50 | $16 \times 50$ | – | 178.4 |
| NONE | 50 | $16 \times 100$ | – | 334.6 |
| NONE | 100 | $4 \times 10$ | – | 96.7 |
| NONE | 100 | $4 \times 50$ | – | 159.3 |
| NONE | 100 | $4 \times 100$ | – | 237.4 |
| NONE | 100 | $16 \times 10$ | – | 143.6 |
| NONE | 100 | $16 \times 50$ | – | 393.6 |
| NONE | 100 | $16 \times 100$ | – | 706.1 |

Table 20. TFLite inference rate *without PSRAM* and *INT8 quantization* for different model hyperparameters and quantization methods. Inference rates marked (–) indicate that the model was unable to run on the microcontroller due to memory issues. Only small values for hidden size and input size are used because RAM space is so limited.

| Quantization | Hidden Size | Input Size | Rate (Hz) | Size (kB) |
|---|---|---|---|---|
| INT8 | 10 | 4 × 10 | 2620 | 3.5 |
| INT8 | 10 | 4 × 50 | 1607 | 5.0 |
| INT8 | 10 | 4 × 100 | 1087 | 7.0 |
| INT8 | 10 | 16 × 10 | 1778 | 4.7 |
| INT8 | 10 | 16 × 50 | 662 | 10.9 |
| INT8 | 10 | 16 × 100 | 372 | 18.7 |
| INT8 | 50 | 4 × 10 | 996 | 9.8 |
| INT8 | 50 | 4 × 50 | 567 | 17.6 |
| INT8 | 50 | 4 × 100 | 369 | 27.4 |
| INT8 | 50 | 16 × 10 | 636 | 15.7 |
| INT8 | 50 | 16 × 50 | – | 46.9 |
| INT8 | 50 | 16 × 100 | – | 86.0 |
| INT8 | 100 | 4 × 10 | 399 | 26.5 |
| INT8 | 100 | 4 × 50 | – | 42.2 |
| INT8 | 100 | 4 × 100 | – | 61.7 |
| INT8 | 100 | 16 × 10 | – | 38.3 |
| INT8 | 100 | 16 × 50 | – | 100.8 |
| INT8 | 100 | 16 × 100 | – | 178.9 |

Table 21. TFLite inference rate *with PSRAM* and *without quantization.* Hidden sizes and input sizes are larger than in Table 19 because PSRAM is able to accommodate these larger machine learning models during model inference.

| Quantization | Hidden Size | Input Size | Rate (Hz) | Size (kB) |
|---|---|---|---|---|
| NONE | 25 | 16 × 25 | 145.0 | 46.3 |
| NONE | 25 | 16 × 100 | 41.3 | 163.5 |
| NONE | 25 | 16 × 500 | 8.7 | 788.5 |
| NONE | 25 | 64 × 25 | 41.3 | 163.5 |
| NONE | 25 | 64 × 100 | 7.6 | 632.2 |
| NONE | 25 | 64 × 500 | 1.5 | 3132.2 |
| NONE | 100 | 16 × 25 | 28.3 | 237.4 |
| NONE | 100 | 16 × 100 | 9.6 | 706.1 |
| NONE | 100 | 16 × 500 | – | 3206.0 |
| NONE | 100 | 64 × 25 | 9.6 | 706.1 |
| NONE | 100 | 64 × 100 | 2.7 | 2581.1 |
| NONE | 100 | 64 × 500 | – | 12581.1 |
| NONE | 500 | 16 × 25 | 2.5 | 2740.4 |
| NONE | 500 | 16 × 100 | – | 5084.3 |
| NONE | 500 | 16 × 500 | – | 17584.3 |
| NONE | 500 | 64 × 25 | – | 5084.3 |
| NONE | 500 | 64 × 100 | – | 14459.3 |
| NONE | 500 | 64 × 500 | – | 64459.3 |

Table 22. TFLite inference rate *with PSRAM* and *INT8 quantization*. Hidden sizes and input sizes are larger than in Table 20 because PSRAM is able to accommodate these larger machine learning models during model inference.

| Quantization | Hidden Size | Input Size | Rate (Hz) | Size (kB) |
|---|---|---|---|---|
| INT8 | 25 | 16 × 25 | 492.1 | 13.9 |
| INT8 | 25 | 16 × 100 | 100.3 | 43.1 |
| INT8 | 25 | 16 × 500 | 21.3 | 199.5 |
| INT8 | 25 | 64 × 25 | 100.3 | 43.2 |
| INT8 | 25 | 64 × 100 | 26.4 | 160.3 |
| INT8 | 25 | 64 × 500 | 4.0 | 785.3 |
| INT8 | 100 | 16 × 25 | 71.0 | 61.6 |
| INT8 | 100 | 16 × 100 | 25.2 | 178.8 |
| INT8 | 100 | 16 × 500 | 5.7 | 803.8 |
| INT8 | 100 | 64 × 25 | 25.2 | 178.9 |
| INT8 | 100 | 64 × 100 | 7.0 | 647.6 |
| INT8 | 100 | 64 × 500 | 1.1 | 3147.5 |
| INT8 | 500 | 16 × 25 | 6.7 | 687.5 |
| INT8 | 500 | 16 × 100 | 3.6 | 1273.3 |
| INT8 | 500 | 16 × 500 | – | 4398.4 |
| INT8 | 500 | 64 × 25 | 3.6 | 1273.3 |
| INT8 | 500 | 64 × 100 | – | 3617.2 |
| INT8 | 500 | 64 × 500 | – | 16117.2 |

the on-board inference rate and reduce the model size. For simplicity, valid model sizes could be constrained to the maximum space available on the ESP32 for model definitions.[4] Depending on the hyperparameters used, we can achieve inference rates from 145Hz to 1.5Hz when not using quantization or 492.1 down to 1.1Hz when using **INT8** quantization. Using **INT8** quantization can achieve much greater inference rate compared to a non-quantized model because the model definition is stored in PSRAM which has a relatively slow-speed Serial Peripheral Interface (SPI) data bus. This means that the larger the model size, the longer it takes to transfer the model over the SPI interface. As such, models with the same hidden size and input size achieve much higher prediction rates with quantization. However, if we compare models with similar sizes such as quantization: **NONE**, hidden size: 25, input size: $64 \times 25$ where the model size is 163.5kB compared to the quantized model with quantization: **INT8**, hidden size: 25, input size $64 \times 100$ where the model size is 160.3kB, we find that they achieve a prediction rate of 41.3Hz and 26.4Hz, respectively even though the quantized model is slight smaller in size. This shows that models of similar size in memory are still slower with quantization than without. Even so, it is still the case that out of the 10 models that are too large to fit in PSRAM, only 3 models use **INT8** quantization while the other 7 models do not use quantization. Thus, quantization still proves to be an important method to increase the model architecture size when run on-board the ESP32 hardware.

With such a diverse set of possible inference rates based on the signal preprocessing steps and the model hyperparameters, we should consider what other WiFi sensing research works are able to achieve. In Fig. 15, we show the CDF plot for the

[4] We do not use on-board inference rate nor model size when performing hyperparameter search in Section 4.3 due to the additional time required to perform these evaluations as well as the additional hardware and software requirements.

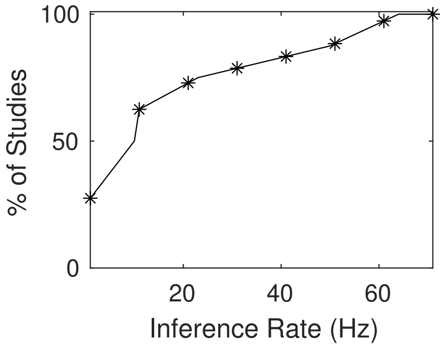


Fig. 15. CDF of machine learning inference rates from surveyed literature ($N = 11$).

inference rates found during our survey of WiFi sensing literature. While we were able to find at least $N = 176$ papers which discuss the CSI sampling rate, inference rate was discussed in far fewer ($N = 11$) research works. Furthermore, out of the 11 works which discussed achievable inference rate, more than half of the works use one or more GPU devices which would not be reasonable to deploy at the edge. Two of the eleven sources explicitly use CPUs rather than GPUs for inference, for example, [12] uses a GPU for training and CPU for testing and is able to achieve an inference rate of 12.5Hz on an Intel Core i5 CPU while [132] uses an Intel Core i5 CPU for both training and testing because of a lack of access to GPU and achieved an inference rate of less than 1Hz. The highest inference rate was achieved in [142] at approximately 60Hz when using an NVIDIA Titan XP GPU. Based on the figure, more than 50% of the works noted can only achieve an inference rate of 10Hz or less even though the models are run on far more powerful computers and servers compared to the ESP32. Due to the relatively low number of works discussing inference rate, we suggest that inference rate should be more commonly evaluated in the broader WiFi sensing research community. Otherwise, as machine learning architectures become deeper and more complex, we will not be able to gauge if the architectures are reasonable in real-time edge scenarios.

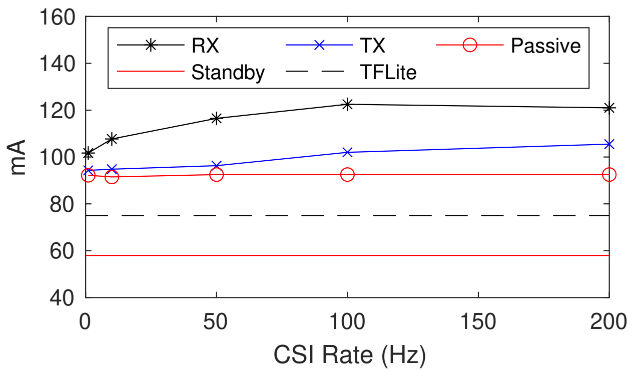


Fig. 16. Energy consumed by individual components of our ESP32 system.

### 4.4.4 Energy Consumption

In Fig. 16, we show the energy consumption for different individual tasks running on the ESP32. Specifically, we look at the energy consumption for the active RX, as well as the active TX and passive RX for different CSI sampling rates. In addition to these three applications, we look at the energy consumed by an ESP32 performing model inference with TFLite as well as the default energy consumption of an ESP32 when not running any specific computation tasks. We can see that the active RX line requires the highest amount of energy. Additionally, the energy requirement for RX also increases to 121mA when the CSI sampling rate is set to 200Hz compared to 110mA when the sampling rate is set to 10Hz. The RX acts as an access point and thus must take on additional overhead tasks such as broadcasting of the SSID and listening for probe requests from new stations. The TX on the other hand has a reduced energy consumption compared to the RX but still shows an increase from 95mA when the CSI sampling rate is 10Hz up to 105mA when the rate is increased to 200Hz. In the passive experiment, we setup a TX and RX to communicate with one another while the passive module simply listens passively to this communication traffic. In this passive case, the energy consumption does not change much as the

CSI rate increases, achieving 92mA and 93mA when the sampling rate is set to 10Hz and 200Hz, respectively.

To evaluate energy consumption when performing TFLite inference on-board, we allow the model to perform inference at the maximum achievable rates that we found in Table 21 and Table 22. We find that the energy consumption is similar no matter the sampling rate, for example with quantization method: **INT8**, hidden size: 25, input size: $16 \times 25$, the energy consumption is measured at 74.5mA when performing 492.1 samples per second while with quantization method: **INT8**, hidden size: 10, input size: $64 \times 100$ which only achieves an inference rate of 7.0 samples per second, the energy consumption is measured at 75.0mA. Additionally, we find that quantization has no effect on energy consumption. Since in these experiments we perform inference back to back without allowing the microcontroller to idle between predictions, the ESP32 is continuously performing computations at all times whether the sampling rate is low or high. As such, if we wish to decrease the energy consumed by the ESP32, we would need to allow idle time between each model inference computation so that the ESP32 is not continuously performing computational work.

By default, when the ESP32 is running in standby idle mode (i.e., no computations are being performed and the WiFi radio is not enabled), the energy consumption is 58mA. Compare to this, performing TFLite inference increases the energy consumption by approximately +17mA. Similarly, when using the WiFi radio interface, the energy consumption can increase anywhere from approximately +35mA when using the passive application mode up to as much as +63mA when using the active RX application. As such, the active RX increases energy consumption by +108.6% while the passive application increases it by +60.3% and the TFLite application increases it by +29.3%. Energy consumption may be an important concern when the ESP32 is powered by a battery. For example, a $9,000$mAh rechargable battery may power an ESP32 active RX for $\frac{9{,}000\text{mAh}}{120\text{mA}} = 75$ hours on a full charge or $\frac{9{,}000\text{mAh}}{93\text{mA}} = 96.8$ hours when running the passive firmware. The length of time required on battery is highly dependant on the application being performed and whether or not it is possible to attach the ESP32 to some central power source indoors. Furthermore, collecting CSI may not be required 24 hours every day such as cases where the indoor location is unoccupied. In which case, the ESP32 can switch to idle standby mode when WiFi sensing is not needed and can achieve $\frac{9{,}000\text{mAh}}{93\text{mA}} = 155.2$ hours on standby. Typically, when CSI is not being collected, the ESP32 can be put into an even lower power mode such as *deep sleep* mode which can achieve up to $\frac{9{,}000\text{mAh}}{7\text{mA}} = 1,285.7$ hours of battery life.

## 4.5 Lessons Learned

### 4.5.1 Selecting Signal Processing Techniques

We surveyed a set of signal processing techniques in Section 4.2 which we then evaluated based on accuracy in Section 4.3, and based on system concerns such as achievable inference rates and energy consumption in Section 4.4. As a result of these evaluations, here we summarize our observations.

#### 4.5.1.1 Feature Extraction

Through our evaluation, we find that amplitude can achieve greater accuracy compared to phase in the experimental datasets evaluated. This is because phase typically requires denoising methods which are only possible when multiple antennas with synchronized oscillator frequencies are available. Furthermore, while other feature extraction methods (i.e., statistical feature, PSD, and wavelet transform) are more specialized compared to strictly amplitude features, we find that none of these other feature extraction methods are able to achieve consistent higher accuracy than amplitude. Additionally, these other methods also require more computation and thus increase energy consumption and reduce the inference rate. Even so, we find that amplitude alone is still not a sufficient choice as input into a machine learning classifier.

#### 4.5.1.2 Denoising Filters

None of the explored denoising filters achieved top performance across all experimental scales (small, medium, large). However, we do recognize that applying certain denoising methods still improves the accuracy compared to using the default noisy amplitude signal. Since denoising filters appear to increase accuracy uniquely on a per-application basis, we suggest that selecting denoising filters should only be used when smaller accuracy improvements are required. This is especially true when denoising is used in addition to dimensionality reduction techniques. However, we find that denoising can greatly reduce the inference rate of a WiFi sensing system and as such, hyperparameters selected per denoising method must be selected appropriately.

#### 4.5.1.3 Dimensionality Reduction

Through this work, we identified that both PCA and ICA provide the most consistent results out of each of the evaluated signal processing techniques when it comes to model accuracy for all three experimental scales. As such, we suggest that by default, PCA or ICA should be used for dimensionality reduction. In addition, dimensionality reduction like PCA and ICA also decreases the input size for the model, thus further increasing the inference rate and decreasing the training time.

### 4.5.2 Feasibility of WiFi Sensing at the Edge

The goal throughout this work is to identify a taxonomy of components required for a full edge WiFi sensing system (i.e., signal processing, data preparation, prediction making, and systems and hardware). Through this survey, we have evaluated and recognized important metrics such as accuracy, inference rate, and energy consumption which must be considered to achieve edge-based WiFi sensing systems. Through this effort, we performed initial evaluations on each of these metrics to compare techniques such as feature extraction, signal denoising, and dimensionality reduction which are applicable to most WiFi sensing applications.

#### 4.5.2.1 Identify ESP32 for Edge WiFi Sensing

Towards the goal of understanding the feasibility of WiFi sensing at the edge, we have identified a hardware candidate capable of achieving edge WiFi sensing; namely the ESP32 microcontroller. This low-cost microcontroller provides WiFi communication on-board and offers access to the important CSI metric which is integral to achieving WiFi sensing tasks. Furthermore, with reasonable machine learning architectures, we can even perform prediction making directly on-board the ESP32, thus allowing for a standalone WiFi sensing device which can be leveraged in scenarios similar to any standard sensor. This is important in ensuring the scalability of WiFi sensing systems towards greater ubiquity for daily sensing tasks.

#### 4.5.2.2 Evaluated ESP32 for different use cases

Within this work, we further evaluated the use of the proposed ESP32 for different use cases, specifically: small-scale hand gesture recognition, medium-scale human activity recognition, as well as large-scale localization and activity recognition. While there are many different use cases that are possible with WiFi sensing, by demonstrating the capability of the ESP32-based edge WiFi sensing in these varying scales, we show that the ESP32 is a feasible candidate for a variety of tasks. Thus, we believe that this work will encourage further efforts in edge-based WiFi sensing with the ESP32 microcontroller.

In addition to these evaluations on different use cases, another important step to evaluate is the feasibility of performing prediction making directly on-board these small edge devices. There are a number of important issues to consider when running data processing and machine learning such as energy consumption and the low amount of storage available on-board edge devices. Towards improving this, we look at model quantization which is able to reduce storage usage as well as reduce the amount of computation and thus energy consumed in machine learning inference. While quantization is one method for improving model inference on edge devices, there are many other methods that can still be explored.

### 4.5.3 New Considerations for Edge WiFi Sensing

Through our efforts, we believe we have demonstrated the feasibility of WiFi sensing at the edge which is an important step towards achieving real-world and scalable systems which rely on WiFi sensing. However, allowing for edge-based sensing introduces some new considerations which must be taken in future research works.

#### 4.5.3.1 Need for Inference Rate Evaluations

Through our survey, we identified that very few works discuss the inference rate offered by their model architectures. Furthermore, we find that the research works that discuss inference rate tend to use high powered GPU-based systems which are not appropriate for real-world systems. Increasing inference rates offers a number of improvements including (i) improved real-time human-computer interaction responsiveness, (ii) the opportunity to decrease energy consumption by adding gaps between each prediction, and (iii) more processing time for other tasks such as communicating results with neighboring devices.

#### 4.5.3.2 Need for Lightweight Model Architecture Designs

As the popularity of deep learning continues to increase, model architectures are becoming more accurate while also becoming far more complex. While deep learning may be reasonable on highly powerful systems, they are not appropriate for edge-based systems where low-costs are required and thus only low-powered devices are available. Based on this observation, we believe that it is important that WiFi sensing researchers should consider developing lightweight model architectures to accommodate this edge-based scenario. One method we discuss through this work to achieve more lightweight architectures is through the use of quantization where full 32-bit floating weights can be reduced down to 8-bit integers, thereby reducing the overall size of the model. However, there is far more room to explore in this direction.

#### 4.5.3.3 Edge Hardware Considerations

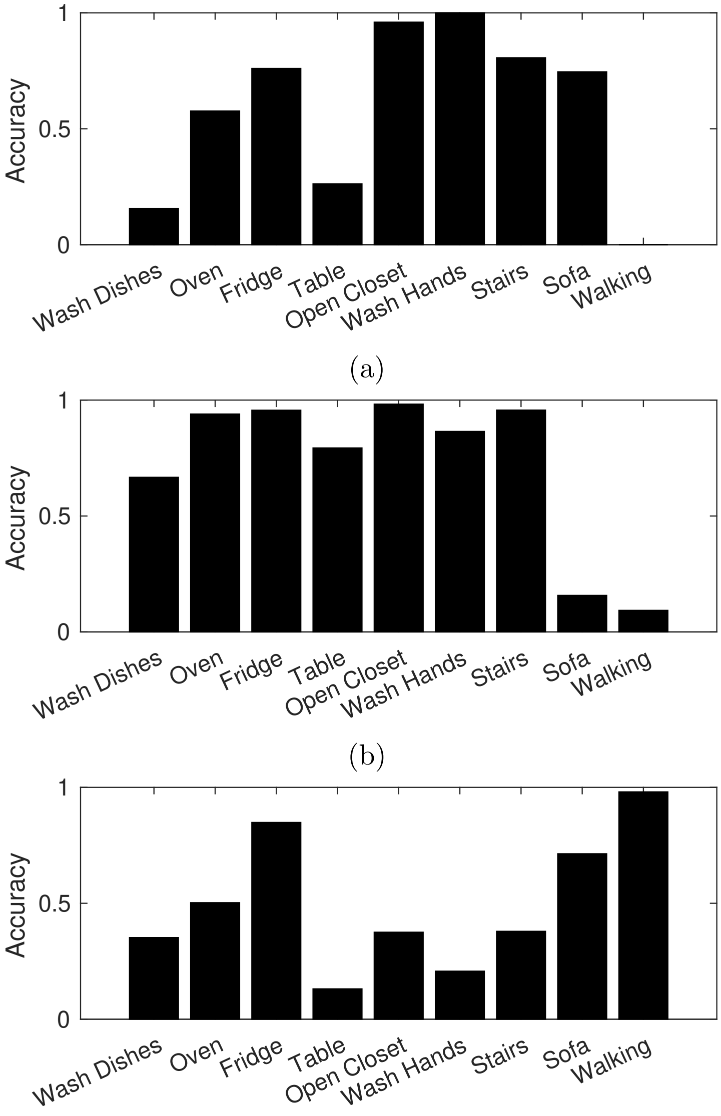
We also found that it is important to take the hardware into consideration when developing edge-based WiFi sensing solutions. For example, towards using the ESP32-MCU for WiFi sensing tasks, we identify that larger model sizes can be achieved through the use of on-board PSRAM modules, however, this results in a slight reduction in inference rate due to slow SPI speeds compared to standard RAM. Furthermore, the availability of neural accelerator hardware can offer additional improvements in inference speed. However, because these accelerators are not designed as general-purpose computation systems, they often require workflows to be converted to match the expectations of the accelerator.

## 4.6 Future Challenges

### 4.6.1 Multiple TX/RX Links

Typical WiFi sensing experiments assume a single TX and a single RX device. However, in real world environments we might have multiple TX devices such as our laptops, smartphones, and IoT devices. Leveraging multiple links that are dispersed throughout an environment may allow WiFi sensing systems to better identify physical actions in much larger environments. For example, in our large-scale experiment; as illustrated in Fig. 11c, we deploy three transmitters in three unique locations throughout the home environment. The transmitters send CSI-frames to the receiver location in the center of the environment.

So far through this chapter, we have only considered the model accuracy when using a single pair in this large-scale experiment. If we train a model for each of the links independently, we expect that some of the links will work well for some of the classes while other links will work better on other classes. In Fig. 17, we can see the prediction accuracy for each of the nine activities when using models trained independently at each of the three links. The model trained on CSI from Link 1 (Fig. 17a) achieves poor prediction quality for three classes: Wash Dishes, Writing at Table, and Walking in Living Room. However, the model trained on CSI from Link 2 and Link 3 (Fig. 17b and Fig. 17c) are able to supplement these inadequacies by achieving much higher accuracy for these three classes. In Table 23, we can see the accuracy achieved by each link independently as well as if we perform different link-selection methods. The first method; labelled *Best Case*, indicates the accuracy if any of the three locations make a correct prediction while the second method; labelled



(a)

(b)

(c)

Fig. 17. Prediction accuracy for all 9 classes of activities given different TX/RX links pairs. (a) TX/RX Link 1. Total Accuracy: 58.52%. (b) TX/RX Link 2. Total Accuracy: 71.24%. (c) TX/RX Link 3. Total Accuracy: 49.89%.

Table 23. Results of various link-prediction selection methods showing that successfully determining the most qualified link will allow for a higher prediction accuracy.

| Link Selected | Accuracy |
|---|---|
| Link 1 | 58.52% |
| Link 2 | 71.24% |
| Link 3 | 49.89% |
| Best Case | 89.14% |
| Worst Case | 29.65% |

*Worst Case*, indicates the accuracy if any of the three locations makes an incorrect prediction. This gives us our bounds for how well our WiFi sensing system could perform when using the three independently trained model. The best case scenario shows an improvement of +17.90% compared to using a Link 2 and +39.25% for Link 3. Thus, we can see that leveraging multiple links leaves room for major improvements for the accuracy of a WiFi sensing system.

A few methods for link selection have appeared due to the multiple antennas available on hardware like the Intel 5300 NIC. For example, WiWrite [45] selects two of the on-board antennas with the highest correlation while Wi-Mose [182] uses the antenna link with the highest variance. More research must be performed to (i) identify methods for leveraging diversely positioned devices, (ii) communicate predictions amongst these devices, (iii) leverage additional links such as from IoT devices.

### 4.6.2 Long-Term Model Adaptation

In this chapter, we primarily focused on understanding the feasibility of running different signal processing techniques over time, however this work does not consider how these methods can adapt to changes over time. Indeed, we find that there are no noticeable trends in the datasets that we collected for this work. However, longer-term CSI data collection (i.e., months or years) may introduce variations that current research works are unable to adapt to. Detrending streaming data signals is a common tactic to handle variations over long periods of time that may reduce the accuracy of a given method. So far in the research literature, we find only three works which consider detrending. These works are: [137] which focuses on sleep stage monitoring, [138] which tracks human walking speeds, and [130] which captures respiratory information over time. In one other work [183], it is suggested that WiFi sensing models can be continuously trained on-device using online stream sampling which allows the system to adapt to changes over time. We suggest that more research work must be done to further identify and understand methods for adapting to changes in CSI signals over time as well as for adapting to physical changes in the signal multipath environment.

### 4.6.3 Real-Time Segmentation

As CSI samples stream into a WiFi sensing system, segmentation can be used to determine whether or not an important action is being performed at any moment. This is useful to reduce how often machine learning inference needs to be performed and thus can also reduce the energy usage of the overall system. In this chapter, we use a *fixed window* approach where predictions are made for a rolling window of CSI. We discussed additional methods for performing segmentation in Section 4.2.2.3, however these methods are specialized to specific use-cases [146, 118] and may not be generalizable to other applications of WiFi sensing. Most works [117], [184], [185]; including the work performed in this chapter, assume that we can evaluate our model only on CSI samples with an associated action. However, more often than not, a WiFi sensing system deployed in an environment will not see any actions being performed (i.e., in the middle of the night). As such, segmentation is another important challenge that we must continue to consider into the future.

### 4.6.4 Integration with Physical Systems

WiFi sensing can be used to recognize an outstanding number of unique physical actions or properties of a given environment. However, while we have seen a great number of laboratory experiments demonstrating novel methods for sensing, to the best of our knowledge, none of these works integrate WiFi sensing predictions into real physical systems. To push WiFi sensing forward as a technology, we need to not only think about interesting use cases and interesting sensing modalities, but we need to deploy these systems and allow them to be leveraged in the real-world such as through intelligent HVAC systems [186, 187], integration with health alert systems [123, 188], and home or office security monitoring and alerting systems [189, 134]. When integrating WiFi sensing into physical systems, additional issues will arise related to device-to-device communication, clock synchronization across devices [190], and knowledge sharing between edge devices [191].

## 4.7 Chapter Contributions and Summary

In this chapter, we considered techniques and challenges when designing real-world WiFi sensing systems that make predictions at the edge. We discussed the theory for topics such as OFDM and CSI which have given rise to a number of novel WiFi-based sensing applications. Through an extensive survey of hundreds of WiFi sensing research works, we identified many signal processing techniques that are commonly applied to incoming CSI data to achieve signal denoising, dimensionality reduction and others. We discussed the mathematics behind these techniques to understand the feasibility of performing each technique on-board low-cost edge devices. It is not only important to understand whether the techniques are possible at the edge, but to also understand if the method is useful in providing improvements in prediction accuracy.

To this end, we performed an extensive set of CSI data collection experiments at small-scale (hand gesture recognition), medium-scale (human activity recognition), and at large-scale (activity and location sensing). Using different experimental scales allows us to identify techniques which result in consistent prediction improvements for many different WiFi sensing applications. For these three experiments, we collected CSI using the ESP32 WiFi-enabled edge microcontroller. The ESP32 is a perfect candidate for edge-based WiFi sensing because it can collect CSI on-board without requiring additional hardware and also because it is low-powered and low-cost. After evaluating the accuracy achieved by each method, we then evaluated the time to compute each signal processing technique on-board the ESP32 microcontroller and recognized which techniques are possible to run in real-time on the incoming stream of CSI data. Additionally, we evaluated the use of TFLite for performing machine learning inference on-board the ESP32. We identified that PSRAM and quantization are required to accommodate larger model architectures on low-resourced edge devices like the ESP32.

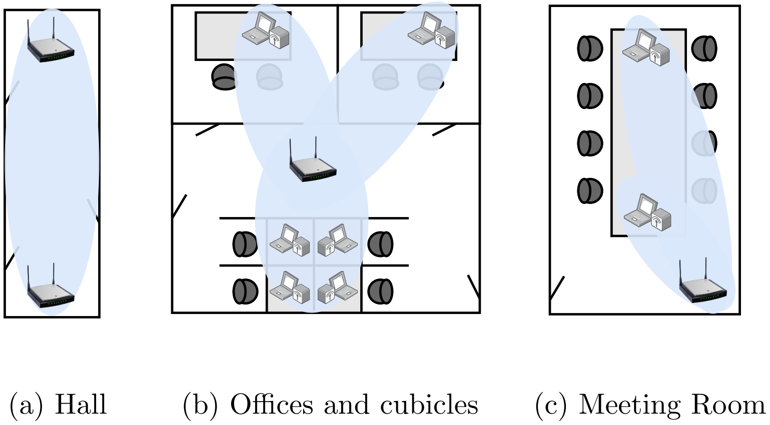
# CHAPTER 5

# SCALABLE WIFI SENSING USING EDGE BASED FEDERATED LEARNING

## 5.1 Introduction

Leveraging existing WiFi AP for sensing purposes provides a remarkable advantage compared to sensor-based methods by reducing costs associated with both hardware and labor for deploying new sensors. However, typical WiFi sensing systems require large amounts of data to train their deep learning models [192, 193, 194, 195], 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. 18a-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. 18a, 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. 18b. Additionally, some other envi-



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

Fig. 18. WiFi sensing environments in an office building. (a) Less cluttered environment. (b) Highly cluttered environment with through-wall sensing. (c) Highly cluttered dynamic environment.

ronments may have very dynamic physical features such as meeting rooms as shown in Fig. 18c 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. 19 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 [136, 148, 196, 144, 197] looking into the creation of generalizable models that can also be used in new environments, however these studies mostly focus on creating a single model upfront rather than in designing a

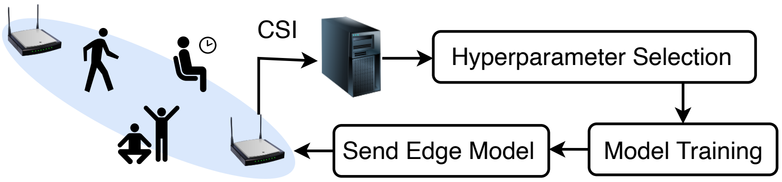


Fig. 19. Typical WiFi sensing system diagram.

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) [198]. 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. The contributions of the work in this chapter 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.

## 5.2 Preliminaries

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.

In this chapter, a rolling mean denoising approach is taken, which uses a window of size $W$ samples, and applies the mean function over the previously collected $W-1$ samples along with the currently collected sample across a single subcarrier. More formally, we calculate

$$\hat{A}^{(i)}_{(t)} = \frac{1}{W}\sum_{w=0}^{W-1} A^{(i)}_{(t-w)}. \tag{5.1}$$

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

Using the preprocessed CSI data, we then train a machine learning model using a 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

$L_2$ regularization for the kernel weights as well as an $L_1$ regularization penalty term for the activation output of each layer. The size of the input matrix for the model is $100 \times 64$, 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 $d \in (0, 1)$ 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 $\eta = 10^{-5}$ to minimize the loss function

$$\mathcal{L}(\theta, x, y) = \frac{1}{N}\sum_{i=1}^{N}\big(\mathcal{F}_\theta(x_i) - y_i\big)^2, \tag{5.2}$$

where $\theta$ is the set of parameters for the model ($\mathcal{F}$), $\mathcal{F}_\theta(x)$ is the prediction output of the model given input $x$, and $y$ is the expected output based on the given action class which is a one-hot encoded vector of size $N$, where $N$ 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 (i.e., RNN or LSTM architectures) which would require more training data and longer training periods.

## 5.3 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. 19.

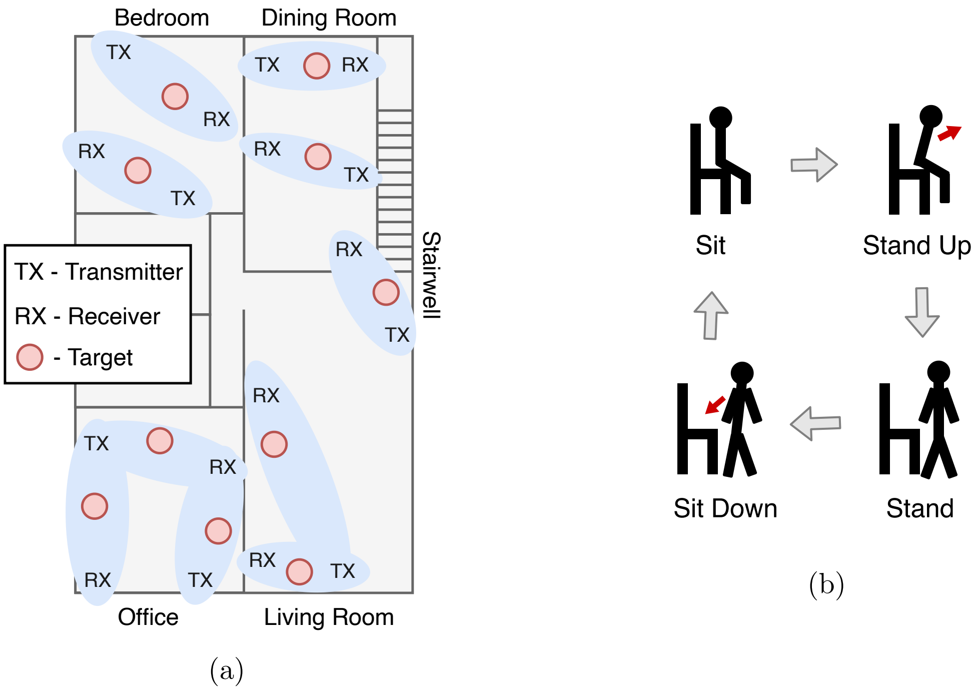


Fig. 20. (a) Illustration of apartment environment where experiments are performed. TX, RX and human target are shown for each room location. (b) Four distinct actions (i.e., sit, stand up, stand and sit down) are recorded and annotated in each location in round-robin order.

### 5.3.1 Experimental Setting

In this study, we consider a collaborative network of WiFi sensing devices in different and disconnected physical locations. The illustration in Fig. 20a 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

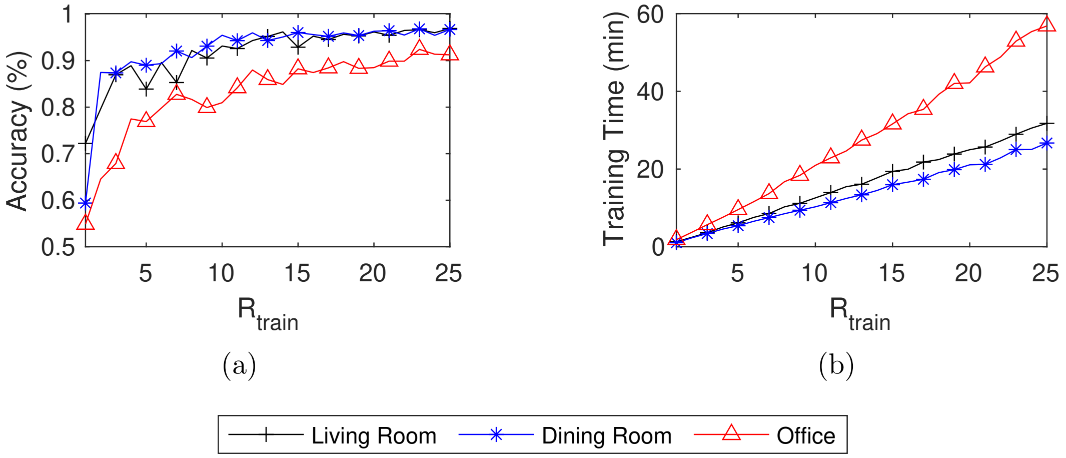


Fig. 21. (a) Accuracy for each locally trained model when different numbers of training repetitions of an action are performed in the location. (b) Training time to perform 100 epochs of training on a Raspberry Pi Edge device.

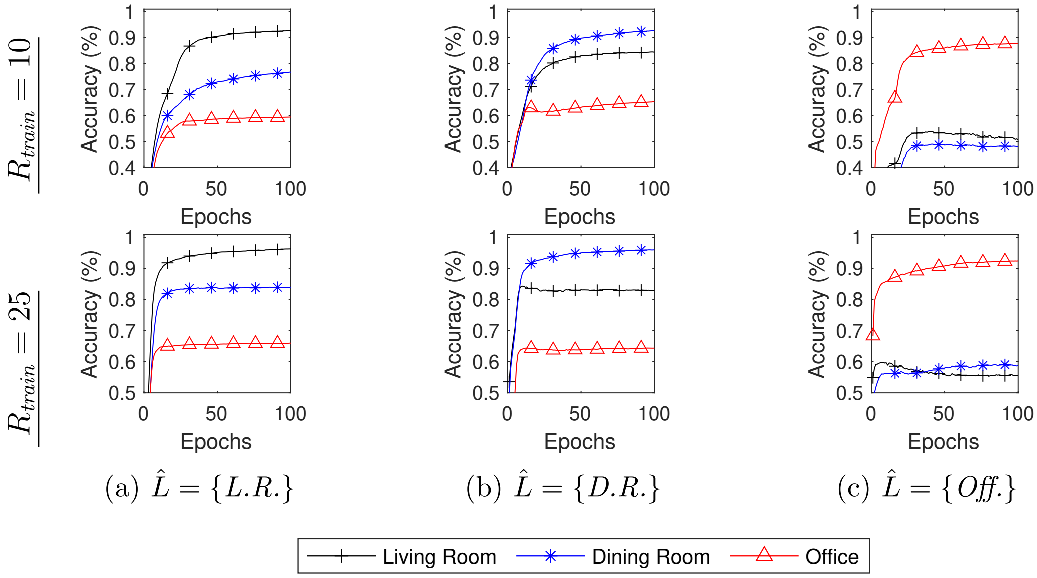
performs four actions (sitting, standing up, standing, sitting down) as illustrated in Fig. 20b. In each location, we perform 50 individual repetitions of each of the four actions in round-robin order resulting in a total of $2,000$ annotated action-segments with corresponding CSI samples. For each location, 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.

### 5.3.2 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 important when we aim to use preexisting WiFi infrastructure to build a WiFi sensing system [93]. 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. 21a shows the accuracy of a non-collaborative local model with different numbers of training repetitions ($R_{train}$). 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. 21b 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 $\hat{L}$ throughout the chapter) which we use to globally train a single machine learning model that will then be shared with all locations ($L$) including those which have zero collected or annotated CSI samples.

In Fig. 22, 23, and 24 we show the prediction accuracy of our model ($\mathcal{F}_\theta$) when evaluated on each location $L_i \in L$, where $L = \{\text{Living Room}, \text{Dining Room}, \text{Office}\}$, after being trained on CSI data from different sets of $\hat{L} \subseteq L$. 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 $R_{train} \in \{10, 25\}$ to identify how increasing $R_{train}$



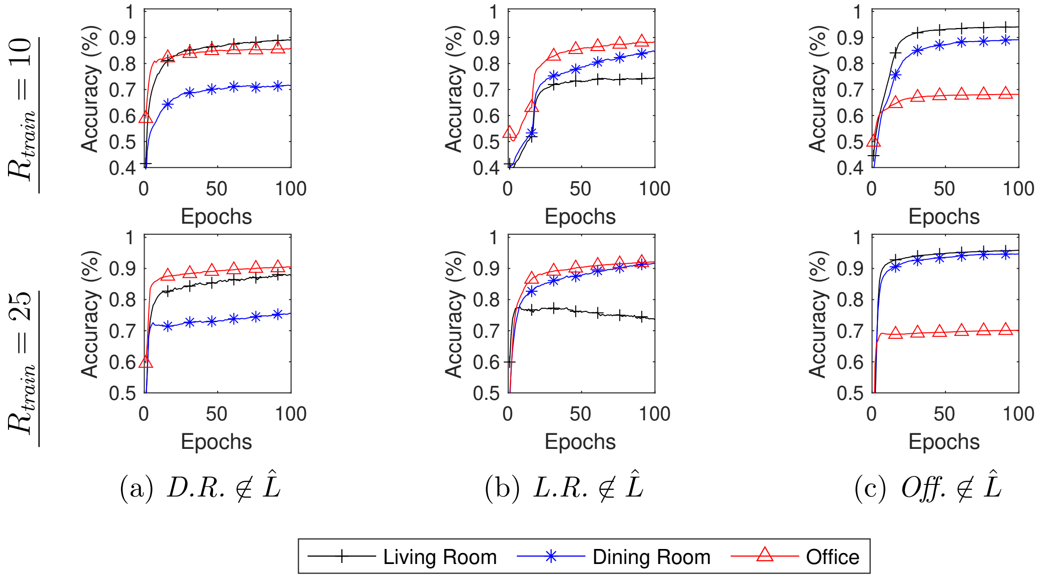
(a) $\hat{L} = \{L.R.\}$ (b) $\hat{L} = \{D.R.\}$ (c) $\hat{L} = \{Off.\}$

Fig. 22. Prediction accuracy in three locations (Living Room ($L.R.$), Dining Room ($D.R.$), Office ($Off.$)) when trained with data from only one location ($\hat{L}$).

affects the prediction capability of locations in $\hat{L}$ as well as locations not in $\hat{L}$ (i.e., $L - \hat{L}$).

In Fig. 22a, we consider the case when $\mathcal{F}_\theta$ 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 $R_{train} = 10$ and $R_{train} = 25$, 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 $R_{train}$ 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. 22b we train only on the Dining Room location. When

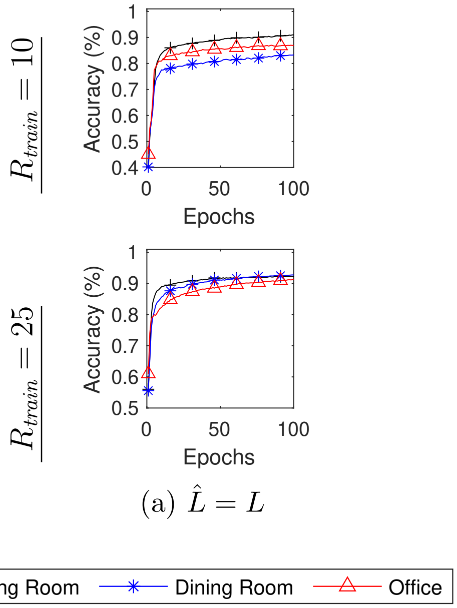


(a) *D.R.* $\notin \hat{L}$ (b) *L.R.* $\notin \hat{L}$ (c) *Off.* $\notin \hat{L}$

Fig. 23. Prediction accuracy in three locations (Living Room (*L.R.*), Dining Room (*D.R.*), Office (*Off.*)) when trained with data from only two locations ($\hat{L}$).

$R_{train} = 10$ 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. 22c 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. 22a 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) $\hat{L} = L$

Fig. 24. Prediction accuracy in three locations (Living Room ($L.R.$), Dining Room ($D.R.$), Office ($Off.$)) when trained with data from all locations ($\hat{L}$).

a gap of approximately 44% difference between the accuracy in the seen and unseen locations.

Training our model on data from two locations rather than one produces Fig. 23. The model accuracy in the unseen locations mostly converge to a static value except in Fig. 23b when $R_{train} = 25$, where the gap in accuracy between seen and unseen locations grows as training continues. In Fig. 23b, when $R_{train} = 10$, the gap is only +10.04%, but when $R_{train} = 25$, the gap increases significantly to +18.05%. Out of each of these three cases where $|\hat{L}| = 2$, Fig. 23c shows the largest overall gap of +21.04% when $R_{train} = 10$ and +24.53% when $R_{train} = 25$. 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., $\hat{L} = L$) as we can also see from the results in Fig. 24. 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 [199, 200, 201], 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.

### 5.4 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 [198], 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 ($\theta$) minimizing:

$$\min_{\theta \in \mathbb{R}^d} \sum_{k=1}^{|L|} \frac{1}{n_k} \sum_{x_i, y_i \in P_k} \mathcal{L}(\theta, x_i, y_i), \tag{5.3}$$

where $L$ is the set of the clients at diverse physical locations, $P_k = \{\mathbf{x}_k, \mathbf{y}_k\}$ is the set of annotated data points for a client $k$ where $\mathbf{x}_k$ is the set of CSI input and $\mathbf{y}_k$ is the corresponding set of annotated labels recorded at the location of client $k$ such that $n_k = |P_k|$, and $\mathcal{L}(\cdot)$ is a loss function as described in Equation (5.2).

The issue with this optimization is that $|L|$, 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 ($\hat{L} \subseteq L$) to iteratively update the model weights. For each client ($k \in \hat{L}$), we learn unique model parameters ($\theta_k$) by optimizing:

$$\min_{\theta_k \in \mathbb{R}^d} \frac{1}{n_k} \sum_{i \in P_k} \mathcal{L}(\theta_k, x_i, y_i). \tag{5.4}$$

After $N_{epochs}$ of training per client, each client sends $\Delta\theta_k$ to a central server to perform *Federated Averaging (FedAvg)* [198] to determine new model parameters for the next round $(r + 1)$:

$$\theta^{(r+1)} = \theta^{(r)} + \sum_{k=1}^{K} \frac{n_k}{n} \Delta\theta_k, \tag{5.5}$$

where $\Delta\theta_k$ is gradient change over top of $\theta^{(r)}$ learned from the data available to client $k$. Whenever new clients join the network, they can use the pretrained federated model parameters ($\theta^{(r)}$) for the given round as-is if no additional labelled data points are available for the locations or alternatively can perform $N_{post\text{-}epochs}$ 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 [202] 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

**Algorithm 1:** WiFederated Learning

```
Input: Set of all WiFi sensing locations $L$.
       Global model $G$ where $G.W$ is the set of weights for the model.
       Local model $M_l \ \forall l \in L$.
$G.W = RandomWeights()$
forall $r \in \{1, \ldots, N_{rounds}\}$ do
    forall $l \in L$ do // Share weights.
        $M_l.W = G.W$
    $\hat{L} = SelectClients_{Train}(L)$
    forall $l \in \hat{L}$ $s.t.$ $\hat{L} \subseteq L$ do // In parallel.
        $M_l.train(N_{epochs}, R_{train})$
    $\hat{L}' = SelectClients_{FedAvg}(\hat{L})$
    $G.W = \frac{1}{|\hat{L}'|} \sum_{l \in \hat{L}'} M_l.W$ // FedAvg.
forall $l \in L$ do // Post-Train.
    $M_l.train(N_{post\text{-}epochs}, R_{post})$
```

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, 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. $L$ is the set of locations where we aim to perform WiFi sensing and $G$ is our global federated model where $G.W$ is the set of weights for all layers of the model. $L$ 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 ($N_{rounds}$) before terminating, however federated training can be performed 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 ($\hat{L}$) to train locally for some fixed number of training epochs ($N_{epochs}$). 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 overfit onto location-specific data. Additionally, high $N_{epochs}$ will result in over-utilization of individual devices which will result in a higher power consumption for these edge devices. Moreover, as mentioned in [203, 204], 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 $N_{epochs}$ 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 $R_{train}$

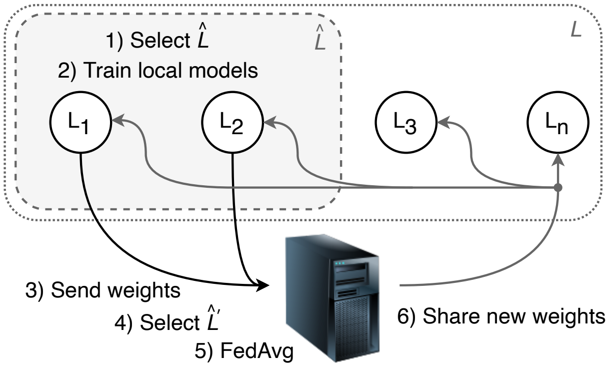


Fig. 25. Illustration of one round of the WiFederated system.

for each location. Furthermore, after the personalization step, we may decide to further refine our selected clients down to $\hat{L}' \subseteq \hat{L} \subseteq L$ 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 $N_{rounds}$, each location in $L$ can perform $N_{post\text{-}epochs}$ of post-training using $R_{post}$ 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.

The illustration in Fig. 25 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 $L_i \in \hat{L}$, the server can perform another round of client selection to select $\hat{L}' \subseteq \hat{L}$ 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.

## 5.5 Evaluation

In this section, we evaluate the proposed *WiFederated* learning framework for use in the setting described in Section 5.3.1.

### 5.5.1 Impact of Averaging Interval

Two types of clients exist in our system, clients *participating* in federated averaging (i.e., the clients in $\hat{L}$) and clients with data that is *unseen* during the training phases (i.e., the clients not in $\hat{L}$).[1] We first look at how the federated averaging process

[1] For the initial evaluations in this section, we say $\hat{L}' = \hat{L}$. Client selection methods for selecting $\hat{L}'$ are further evaluated in Section 5.5.6.

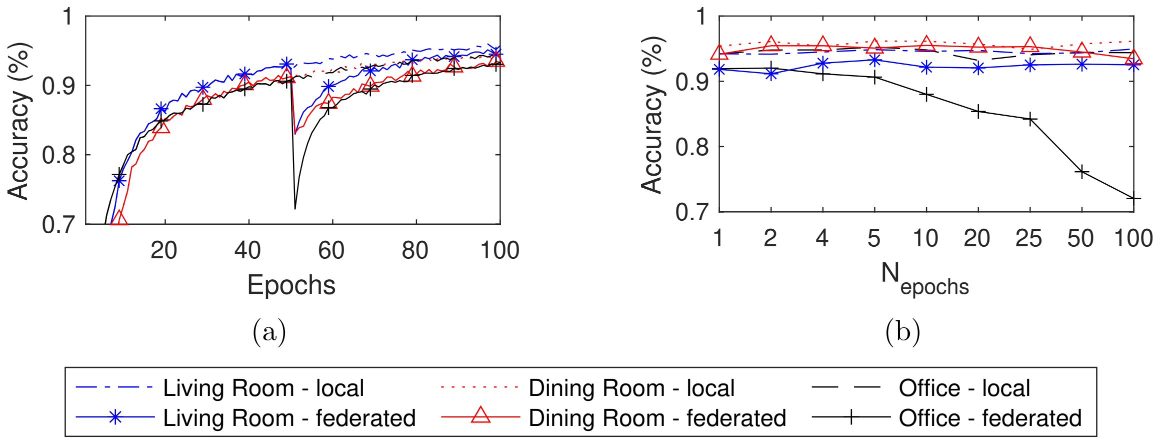


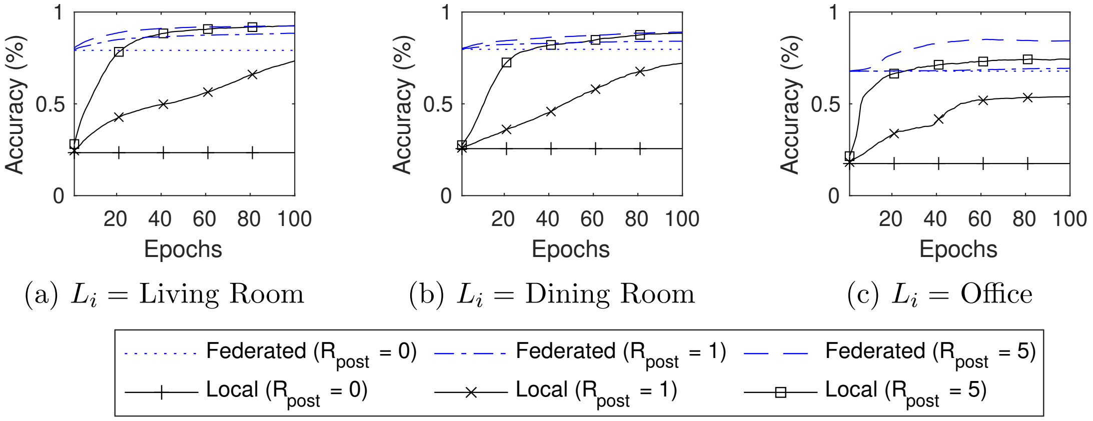
Fig. 26. (a) Accuracy of federated learning over 100 epochs when $N_{epochs} = 50$ and $\hat{L} = L$ versus local machine learning. (b) Accuracy after applying final round of FedAvg for different values of $N_{epochs}$.

affects the prediction capability of the locations *participating* in training the federated model. To evaluate this, we set $\hat{L} = L = \{\text{Living Room}, \text{Dining Room}, \text{Office}\}$ and evaluate on each of these clients individually. Fig. 26a shows the prediction accuracy when the number of epochs per round $N_{epochs} = 50$. 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 $\hat{L}$ is to sacrifice some amount of predictive capability so as to benefit other new locations.* Even so, we can see in Fig. 26a 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. 26b shows the accuracy comparison between locally trained models and federated models during the final FedAvg aggregation. We can see that, as $N_{epochs}$ increases, the accuracy for the Office location decreases indicating that if we set $N_{epochs}$ 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 $N_{budget} = N_{epochs} \times N_{rounds} = 100$ epochs, increasing $N_{epochs}$ decreases the number of rounds and thus consequently decreases the network communication. This is because at the end of each round, $|\hat{L}|$ locations must first communicate $\Delta\theta_k$ back to the central server for FedAvg aggregation and then the server must communicate $\theta^{(r+1)}$ back to $|L|$ clients.

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

### 5.5.2 Impact on Unseen Locations

It is useful to see how FL performs from the perspective of clients within $\hat{L}$. 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 $L_i$, we set $\hat{L} = L - \{L_i\}$ when training our federated model

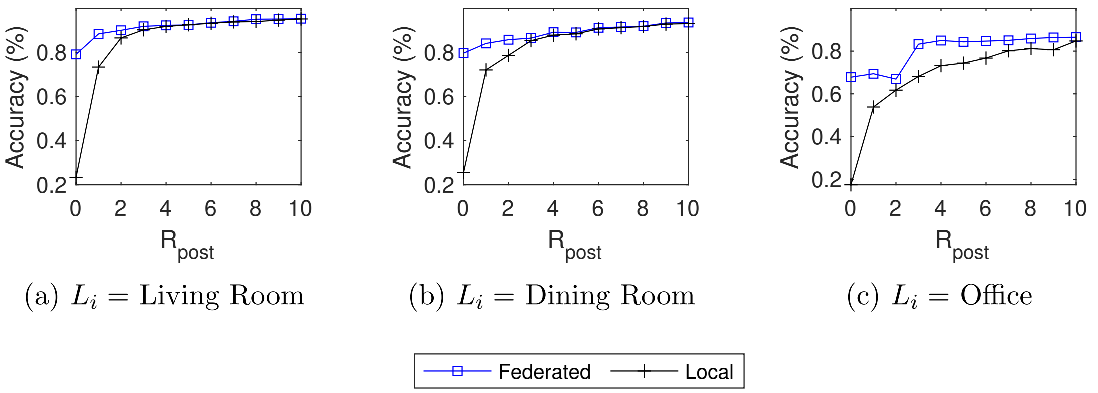


(a) $L_i$ = Living Room (b) $L_i$ = Dining Room (c) $L_i$ = Office

Fig. 27. Accuracy during post-training (personalization) over 100 epochs starting with a randomly initialized local model versus starting with a federated model trained on $\hat{L} = L - L_i$.

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 $L_i$. 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 $\hat{L}$ 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, $\hat{L}$ can also change for every subsequent round to prevent wasting networking resources or power at any single location and also to prevent overfitting on data from the selected locations.

Consider the three subfigures in Fig. 27 where $L_i$ 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 ($R_{post}$) from location $L_i$. For the following evaluations note that a single post-training repetition ($R_{post} = 1$) includes a sample for each class type, as we



(a) $L_i$ = Living Room

(b) $L_i$ = Dining Room

(c) $L_i$ = Office

Fig. 28. Accuracy for federated model versus randomly initialized local model after 100 epochs of post-training (personalization) with different post-training repetitions ($R_{post}$) at $L_i$.

discussed in Section 5.3.1. Consider the case where a new client is added to the network without any post-training repetitions (i.e., $R_{post} = 0$). 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 $\theta$, the accuracy when $R_{post} = 0$ is approximately 25% for each value of $L_i$ 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 $\hat{L}$. For the federated model, these 100 post-training epochs are represented in Algorithm 1 as $N_{post\text{-}epochs}$. We can see that, with the federated model, the accuracy for $R_{post} = 0$ 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 $\hat{L}$, 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.

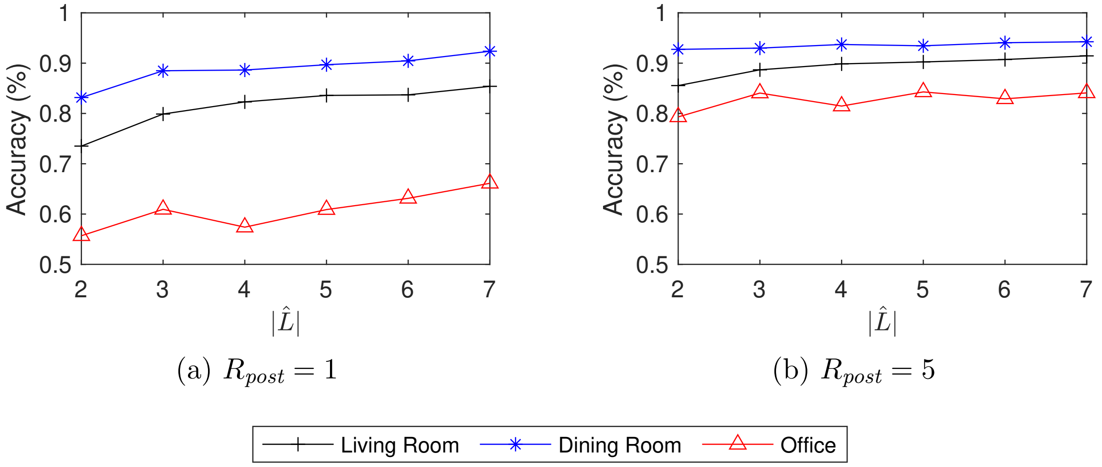
Suppose now that a new client is added to the network, but we are able to perform very few repetitions of our actions in the environment (i.e., $R_{post} \in \{1, 2, 3\}$). We can see in Fig. 28 that the addition of these post-training repetitions allows both models to increase their predictive accuracy by the end of the 100 post-training epochs. However, with small values for $R_{post}$, 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.

### 5.5.3 Impact of the Number of Training Locations

So far, we have evaluated the WiFederated system when the number of training clients $|\hat{L}| \in \{2, 3\}$, 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. The same three locations (i.e., $\{\text{Living Room}, \text{Dining Room}, \text{Office}\}$) are used 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

$$\hat{L} \subseteq L - \{\text{Living Room}, \text{Dining Room}, \text{Office}\}.$$

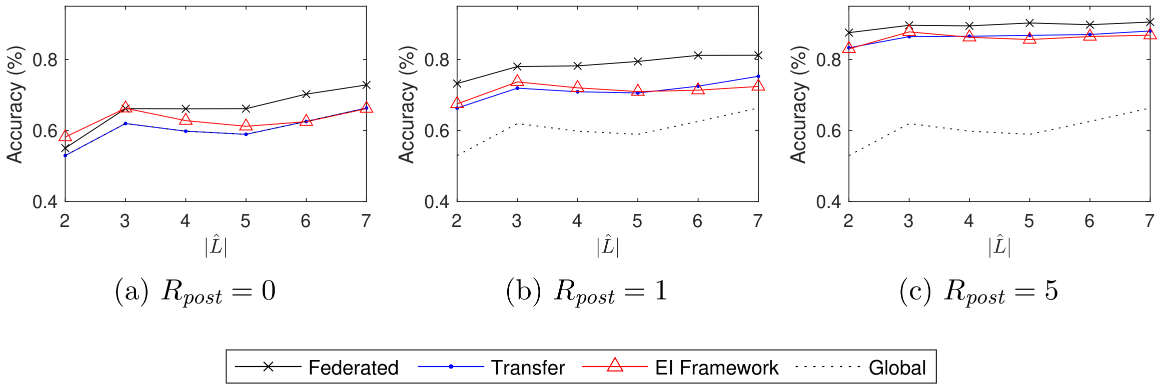
Fig. 29 shows the accuracy for each of the evaluation locations as $|\hat{L}|$ increases up to 7 which corresponds to an increasing trend in the accuracy for the evaluation locations. When $|\hat{L}| = 7$ all seven candidate locations are selected for federated averaging. When $|\hat{L}| < 7$, different combinations of clients could be selected for $\hat{L}$. To account for this, we repeated each experiment 10 times with randomly selected value for $\hat{L}$ each time. This ensures that we do not accidentally select values for



(a) $R_{post} = 1$ (b) $R_{post} = 5$

Fig. 29. Accuracy for WiFederated as $|\hat{L}|$ increases.

$\hat{L}$ which are consequently overly good or overly bad. Furthermore, we use a static client-selection process where the clients in $\hat{L}$ 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 $|\hat{L}|$ client locations so that future client locations require a much shorter data collection and annotation period. When the technician selects these first $|\hat{L}|$ 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 $|\hat{L}|$ from 2 up to 7 clients offers an accuracy increase of +11.88% for $L_i$ = Dining Room, +9.21% for $L_i$ = Living Room and +10.43% for $L_i$ = Office when the number of post-training repetitions $R_{post} = 1$ and +5.91% for $L_i$ = Dining Room, +1.52% for $L_i$ = Living Room and +4.77% for $L_i$ = Office when $R_{post} = 5$. 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.



(a) $R_{post} = 0$ (b) $R_{post} = 1$ (c) $R_{post} = 5$

Fig. 30. Comparison of four methods when using different numbers of pretraining locations with different post-training repetitions.

### 5.5.4 Comparison with State of the Art Approaches

In some recent studies[205, 197], 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 $\hat{L}$ must be aggregated to a central location to train the model. Another approach is the EI framework [196] 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 $\mathcal{L}_a$ while the second branch acts as a domain discriminator with loss value $\mathcal{L}_d$. The domain discriminator attempts to recognize the domain or physical location where the CSI data was collected. The goal is to minimize $\mathcal{L}_{EI} = \mathcal{L}_a - \mathcal{L}_d$ 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. 30a-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. 30a, both *Transfer* and *Global* achieve the same accuracy. This is because when $R_{post} = 0$, 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 $|\hat{L}|$ locations at a global server before sharing the model with the client. Fig. 30b demonstrates how *Transfer* is different from *Global*. Namely, because $R_{post} = 1$, 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 the *Transfer* model by +7.69% when $R_{post} = 0$, +8.87% when $R_{post} = 1$ and +4.26% when $R_{post} = 5$. With EI, we find that when $R_{post} = 0$, and $|\hat{L}| \in \{2, 3, 4\}$, the accuracy is similar to our federated approach but when $|\hat{L}| \in \{5, 6, 7\}$ the accuracy is more similar to the transfer learning approach. We find that as $|\hat{L}|$ 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 $R_{post} \in \{1, 5\}$, 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 $|\hat{L}|$ 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. 31, we demonstrate how the time to train global and transfer learning models increases as

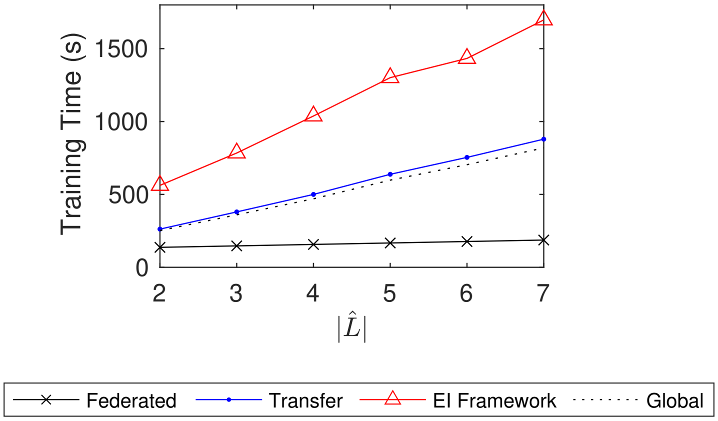


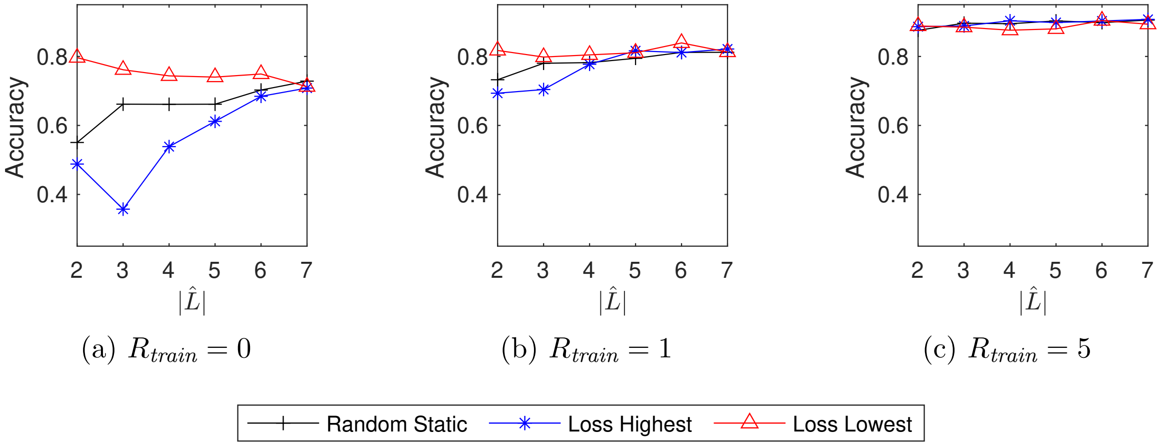
Fig. 31. Training times required for each method with federated learning being the fastest thanks to parallelism.

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. 21b 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.

### 5.5.5 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. 30a-c, we allow all models to initially be trained on the datasets available in $\hat{L}$ for a total of $N_{budget} = 100$ epochs after which the federated models, transfer learning models and adversarial

EI Framework models are trained for an additional $N_{\textit{post-epochs}} = 100$ epochs. For simplicity in our notation, we say that $e = N_{budget} = N_{\textit{post-epochs}}$. The time complexity of training is related to the product of the number of epochs ($e$) and the number of samples ($S$) used which we can denote as $O(e \times S)$. In the case of FL, the data from each location is trained independently and distributed in parallel, thus the time complexity would be $O\big(e \times max(S_l \; \forall \; l \in \hat{L})\big)$. On the other hand, transfer learning would have a dataset size $S = \sum_{l \in \hat{L}} S_l$, thus the time complexity is $O\big(e \times \sum_{l \in \hat{L}} S_l\big)$, which becomes $O(e \times S_l \times |\hat{L}|)$ if we assume $S_l = S_k \; \forall l, k \in L$ 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 $O(e \times S_l \times |\hat{L}|)$. The post training step at a given location for FL, transfer learning and EI is performed on $S_{post}$ number of post training samples, where typically $S_{post} << S_l$, for an additional $e$ epochs which results in an added time complexity of $O(e \times S_{post})$. Thus, we can conclude that FL has a time complexity of $O\big(e \times (S_l + S_{post})\big)$, global learning has a time complexity of $O(e \times S_l \times |\hat{L}|)$, and transfer learning and EI both have a time complexity of $O\big(e \times (S_l \times |\hat{L}| + S_{post})\big)$, where $|\hat{L}|$ 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 $|\hat{L}|$ 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.



(a) $R_{train} = 0$

(b) $R_{train} = 1$

(c) $R_{train} = 5$

Fig. 32. Impact of client selection with different number of training repetitions ($R_{train}$).

### 5.5.6 Impact of Client Selection

Client selection can be performed at two different times during the entire FL process as denoted in Algorithm 1. First, we select $\hat{L}$ to be the clients that are used to train in parallel. After this, we can further reduce this selection to $\hat{L}' \subseteq \hat{L} \subseteq L$. 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 $\hat{L}$ 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 ($k$) whose $\Delta\theta_k$ 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 $\mathcal{L}(\cdot)$ for all clients in $\hat{L}$ to determine which clients should be used for FedAvg. It was previously recognized [206] 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. 32 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 $\hat{L}'$ 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 $|\hat{L}| = 2$ we achieve an accuracy of 79.89% with *Loss Lowest* versus 55.05% for *Random Static*. Similarly with $R_{post} = 1$, the *Loss Lowest* approach achieves 81.74% compared to 73.25%.

## 5.6 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.

### 5.6.1 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. While Chapter 4 discussed the use of WiFi sensing signal processing techniques on-board the ESP32 edge device, here, we additionally analyze and compare single-board edge computer like the Raspberry Pi and NVIDIA Jetson for WiFi sensing models.

*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. 33, we show the time required for training a local model on different values of $R_{train}$ for both a Raspberry Pi

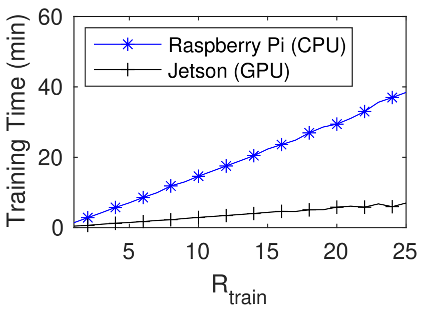


Fig. 33. Average training time for edge devices.

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

Table 24. Average prediction rate for edge devices.

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 $R_{train} \leq 6$. 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 [207, 208] and code libraries such as MicroMLP [209] 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 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 24, we can see a comparison of prediction rates possible with each edge device. Even

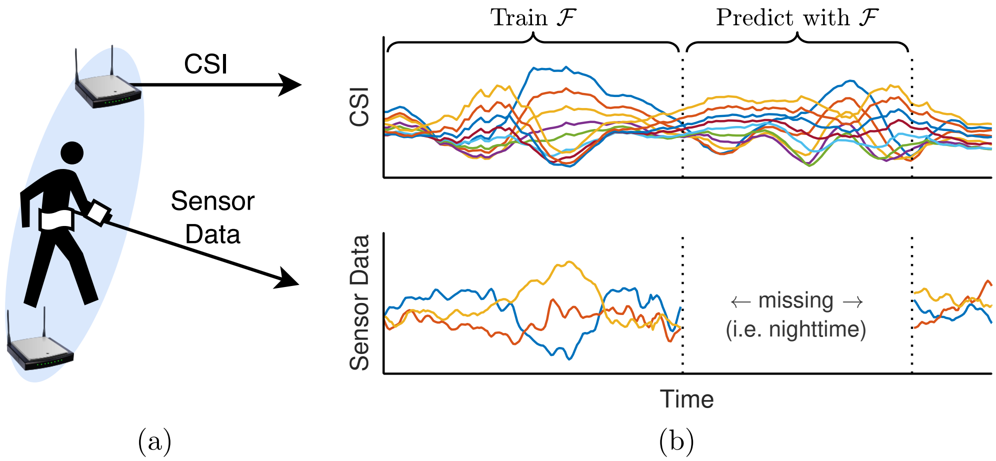


Fig. 34. Example scenario for continuous learning. (a) User with a wearable sensor while CSI is collected in the background. (b) When sensor data is available, both CSI and sensor data can be used to train $\mathcal{F}$. When sensor data is unavailable (i.e., at nighttime when wearable sensors are removed), CSI can be used with $\mathcal{F}$ to continue monitoring.

though both Raspberry Pi and NVIDIA Jetson devices are able to achieve prediction rates of greater than $1,000$Hz, 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.

### 5.6.2 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 [105]. 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. 34a illustrates that CSI can be collected in the background at the same time a wearable sensor is being worn and Fig. 34b 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 when the user removes 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 $\mathcal{F}$. Then, whenever sensor data is unavailable, $\mathcal{F}$ 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 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 [210] over time due to environmental changes.

## 5.7 Chapter Contributions and Summary

In this chapter, we introduced 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 showed that we can leverage this federated model to reduce the number of training repetitions required per physical action when training at new locations. We concluded 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 found that using WiFederated can reduce the number of local training epochs required compared to other methods. Finally, we evaluated 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.

# CHAPTER 6

# ADVERSARIAL OCCUPANCY MONITORING USING ONE-SIDED THROUGH-WALL WIFI SENSING

## 6.1 Introduction

Occupancy monitoring and crowdcounting offers the ability to collect analytics and insights into traffic within indoor spaces for use in intelligent energy efficient heating and air conditioning control systems [211], building security through intruder detection [212] and crowd safety [213]. Unfortunately, human target surveillance in public and private scenarios can also benefit from both occupancy monitoring and crowdcounting techniques. For private locations such as businesses or homes, typical surveillance devices such as cameras or microphones would require an adversary to have full access to the target areas. This of course is not always possible when considering private residences. Further, even when access is possible, the device payload will likely attract attention by the presence of features such as a camera lens.

This chapter proposes the use of WiFi sensing techniques to understand actions occurring in a physical environment through an adversarial or unauthorized manner. Because WiFi is designed to penetrate walls, device payloads no longer need to be placed directly in the target area. Instead, the WiFi receiving device can be placed on the outer perimeter of a room or building to then sense through the walls. Furthermore, because of the ubiquity of WiFi devices and routers in environments such as residential homes and commercial buildings, a WiFi sniffer device can also leverage the natural ambient radio traffic from the existing devices in the environment to detect human presence or activities. In this study, we consider the case where an adversary places a WiFi transmitter and a WiFi receiver in a NLOS location that is behind the wall of the target hallway area.

While there exist studies which consider through-wall crowdcounting such as in [214], in this chapter, we take these efforts further and investigate if it is possible to perform crowdcounting successfully when the access to the monitored area is much more limited. That is, in existing through wall research, it is assumed that both TX and RX can be placed across from one another as illustrated in Fig. 35a. In an adversarial scenario however, an attacker may not be able to place the devices in both areas to perform LOS sensing. Instead, an attacker may only have access to a single room in a building or to the exterior of the building resulting in a limited monitoring ability from only one side. However, this limits the attacker to NLOS conditions as shown in Fig. 35c. Indeed, WiFi sensing based recognition has been shown to be successful in NLOS scenarios [215] in a hallway environment, however the radio is considered to be in the center of the hallway rather than being hidden behind a wall. Also note that these methods require extensive training phases with labeled data for each new target environment before successful results are achieved. Thus, any person tasked with tracking targets must have full access to the environment and then must perform time-consuming setup training before each new deployment. Our proposed system in this study instead leverages signal features common to all environments for prediction. While there are several existing studies that look at the through-wall occupancy detection and crowdcounting problem through device-free WiFi sensing, to the best of our knowledge, this NLOS scenario has not been considered yet, while it is a more practical scenario for an adversary.

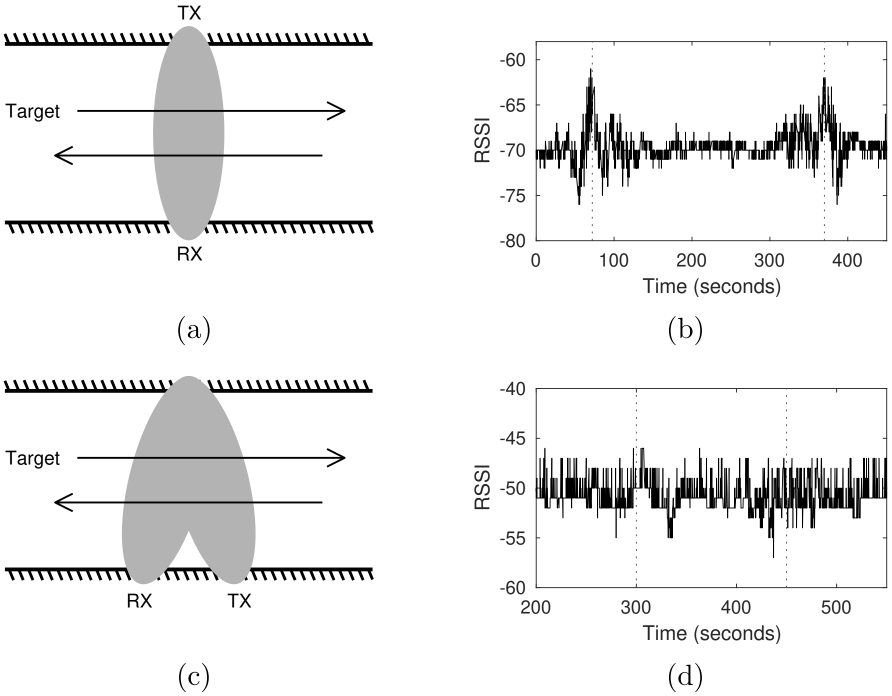


Fig. 35. Through-wall hallway experiment diagrams. Dark lines indicate the walls of the hallway while the gray areas indicate the multi-path propagation of the radio signals from TX to RX as the target walks through the hallway. (a) LOS experiment setup, (b) RSSI for LOS experiment, (c) NLOS experiment setup, (d) RSSI for NLOS experiment.

## 6.2 Proposed Method

To begin explaining this through-wall occupancy monitoring system, we first consider some empirical experiments performed in a real-world hallway environment. For the first experiment, we record radio signal data in LOS conditions as illustrated in Fig. 35a as performed in previous works [216, 217]. Then we move transmitter and receiver into NLOS positions shown in Fig. 35c.

RSSI has previously been used as a simple and more easily obtained signal metric because of its immediate availability on smartphones and other consumer radio-enabled devices. In LOS, RSSI works well to recognize targets as demonstrated in our experiment result in Fig. 35b where the vertical dotted lines indicate the time when the target is passing the LOS. We can see directly that as the target passes, more RSSI variation occurs. However, if we perform a NLOS experiment as illustrated in Fig. 35c, we find that RSSI no longer reveals any signal variations when the target passes as we can see in our NLOS experiment result in Fig. 35d. Instead, in this work, we evaluate the use of the CSI signal metric in similar situations which gives much more fine grained details compared to RSSI.

### 6.2.1 CSI Pre-Processing

Channel State Information varies in new environments because of the unique multi-path conditions of each location. Thus, received $A^{(i)}$ and the change of $A^{(i)}$ over time will vary uniquely when similar actions are performed in different environments. To combat this for our occupancy monitoring problem, we suggest the following signal pre-processing steps to be applied to $A^{(i)}$. We begin the pre-processing by applying a windowed outlier filter. That is, we find

$$\bar{A}_t^{(i)} = \begin{cases} A_t^{(i)}, & \frac{\left|A_t^{(i)} - \mu\left(A^{(i)}\{t-w_1:t\}\right)\right|}{\sigma\left(A^{(i)}\{t-w_1:t\}\right)} < \lambda \\ A_{t-1}^{(i)}, & \text{otherwise} \end{cases} \tag{6.1}$$

where

$$\mu(\mathbf{x}) = \frac{1}{|\mathbf{x}|} \sum_{j=1}^{|\mathbf{x}|} \mathbf{x}^{(j)} \tag{6.2}$$

is the mean function which is applied to the received signal from time $t-w_1$ until the current time $t$ on $A^{(i)}$, where $w_1$ represents the window length parameter. Similarly,

$$\sigma(\mathbf{x}) = \sqrt{\frac{\sum_{j=1}^{|\mathbf{x}|} \left(\mathbf{x}^{(j)} - \mu(\mathbf{x})\right)^2}{|\mathbf{x}|}} \tag{6.3}$$

is the standard deviation function applied to the same window of $A^{(i)}$. The goal of Equation (6.1) is to replace any outlier samples (those which are greater than $\lambda$ standard deviations from the mean) with the most recent valid/normal sample. Outlier filtering is applied to each subcarrier independent of one another.

After filtering outliers, we want to gather aggregated statistics across all subcarriers independently across another time window of size $w_2$. Here, while we can use different values for both $w_1$ and $w_2$, for simplicity, we keep $w_1 = w_2$. For this, we apply some windowed statistical aggregation function $\Phi(\mathbf{x})$ on each subcarrier independently,

$$\tilde{A_t}^{(i)} = \Phi\left(\bar{A}^{(i)}_{(t-w_2:t)}\right). \tag{6.4}$$

For our experiments, we consider $\Phi(\mathbf{x}) \equiv \sigma(\mathbf{x})$ because our goal is to understand how noisy each subcarrier is independently, however $\Phi(\mathbf{x})$ can be replaced with any other statistical function as needed.

After collecting a noise metric for all time instances on each subcarrier, we want to find if the noise present in one subcarrier is similar to the noise present in other subcarriers. Again, we apply a new statistical aggregation function $\Psi(\mathbf{x})$, this time on all subcarriers for a single time instance $t$,

$$A_{CSI,t} = \Psi\left(\tilde{A_t}^{(1:|A_t|)}\right). \tag{6.5}$$

For our purposes, $\Psi(\mathbf{x}) \equiv \mu(\mathbf{x})$ with the intuition that if all subcarriers are high in noise, then $A_{CSI,t}$ will be larger than it is when only a small subset of subcarriers are affected by noise. This is important because the noise resulting from the environment may cause subcarriers to randomly produce noise which is not represented across any other subcarriers. Instead, when a target is present, noise will be present across more subcarriers which will more consistently produce a higher value for $A_{CSI,t}$. On

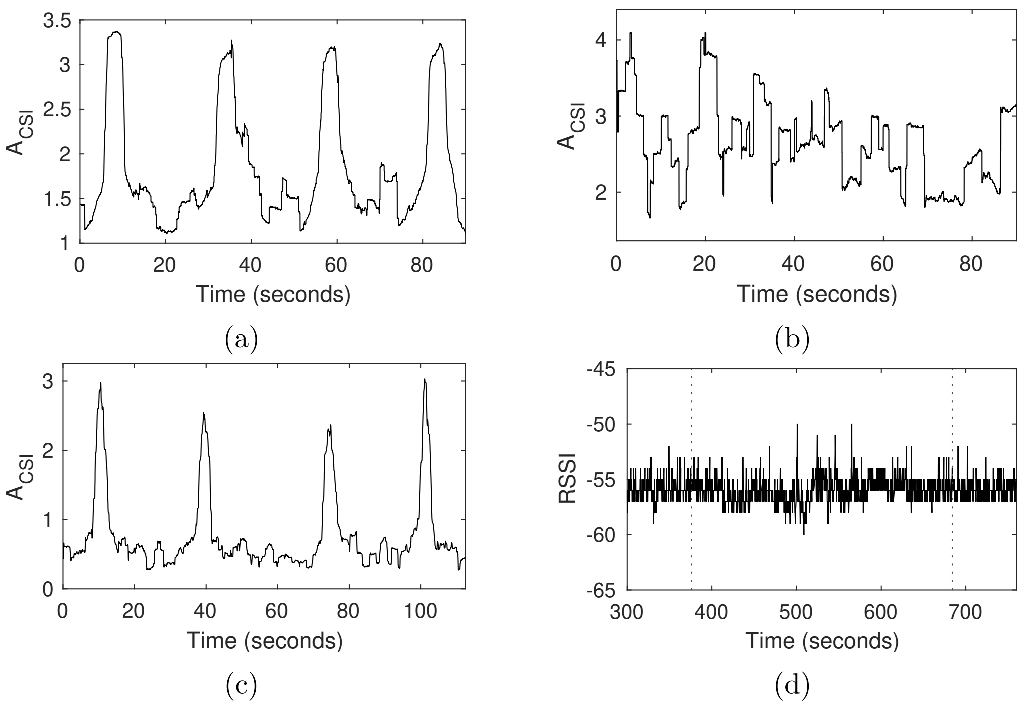


Fig. 36. $A_{CSI}$ for (a) LOS experiment setup, (b) NLOS experiment setup without directional shielding, (c) NLOS experiment with directional shielding, and (d) RSSI with directional shielding. Target is still not detectable with RSSI even with shielding.

the other hand, when no target is present, any noise anomalies present on a single subcarrier will be filtered out because of disagreement across subcarriers. For notation simplicity, we will denote $A_{CSI} \equiv A_{CSI,t}$ with the understanding that $A_{CSI}$ is a scalar metric for some time instance $t$.

### 6.2.2 Standard LOS Through-Wall

As shown in many previous experiments in WiFi sensing, recognizing targets as they pass through the LOS between a transmitter and a receiver is a trivial task and can then be used to estimate the number of targets in an area [214, 217]. We perform our first experiment with one target passing through the LOS. The results in Fig. 36a show when a target passes the LOS four separate times, $A_{CSI}$ gives distinct peaks, indicating that the target has passed by the receiver four separate times. In between these passing events, $A_{CSI}$ returns to some lower noise-floor level.

### 6.2.3 NLOS Through-Wall

As discussed, in certain environments it may not be possible to place a transmitter and a receiver to create such LOS conditions. For example, when rooms are not available on both sides of a hallway or if access is restricted for these adjacent rooms. In these cases, it would be most advantageous for both the transmitter and the receiver to be placed in a single room together against one wall. This is particularly useful in adversarial conditions where an attacker has access to only a single location because they can then keep an eye on the radios as they perform sensing tasks. Fig. 35c illustrates this setup. To the best of our knowledge, such adversarial placement of the transmitter and the receiver has not been attempted by any of the device-free WiFi sensing studies in the literature.

In this case, the target will no longer be in the LOS of the devices, thus a NLOS monitoring will be required. For our first experiment with this NLOS placement, we position both a transmitter and a receiver 6 meters apart, both 50 centimeters away from the wall. The resulting $A_{CSI}$ in Fig. 36b shows that the target passing times are not clearly visible. This is because the direct LOS between the transmitter and the receiver dominates the received signal path which travels through the wall and comes back. As a result, the targets passing through the hallway does not cause a distinguishable change on the received signal amplitude collected. Thus, an update to the setting is needed in order to make the effect of such through-wall NLOS signals prominent.

Typically, WiFi antennas are designed to transmit omnidirectionally, but unidirectional antennas allow for signals to be focused in more specific areas. Unidirectional antennas however must be aimed with great accuracy to ensure that signals are eventually received by the receiver. This may not be an easy task without knowing the characteristics of the environment on the other side of the wall. Our solution is to shield both transmitter and receiver against the wall to prevent the direct LOS signal from dominating the NLOS signal while still allowing for partial omnidirectional propagation in the target area. This will be additionally useful if multiple receivers are used for through-wall sensing with a single transmitter. After adding the directional shielding, we can see in Fig. 36c that we can again identify distinct peaks when the target passes through the hallway environment. Note that RSSI still cannot be used to recognize the passing target in this directional setting as shown in Fig. 36d.

## 6.3 Detection Framework and Evaluation

We now move on to defining our full framework for target detection. For all of the experiments in this work, we use our ESP32-CSI-Toolkit[1] [102] to collect CSI which uses two ESP32 WiFi-enabled microcontrollers for our transmitter and receiver, respectively. Using these small, low-cost microcontrollers demonstrates how an adversary could both implement and distribute large numbers of adversarial devices with less fear of discovery because of their small size and without fear of loss because of the low-cost of each standalone ESP32 module. The ESP32 devices are set to send and receive CSI at a packet rate of 100Hz. The entire framework is designed such that a low resource device such as the ESP32 can perform all tasks in real time without requiring additional external computation power such as a server of laptop as it is often required in WiFi sensing literature. From an adversarial perspective, this is important to ensure that the devices remain small and easy to conceal.

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

### 6.3.1 Human Presence

As shown in Section 6.2, we see that our $A_{CSI}$ metric can be used visually to detect activity whenever a target passes. For our model to predict the binary presence of a target, we designate a threshold parameter $\tau$. When $A_{CSI} \geq \tau$, then the model predicts the presence of the target. We perform our experiment with a target passing the monitored area five times. For each time instance, our model predicts whether a target is present. To evaluate how well different thresholds work in predicting the class of our samples, we define a class prediction probability metric $P^{(c)}_{samples}$ for samples of a given class $c \in \{\text{'target'}, \text{'no target'}\}$. The class for a sample at time $t$ is denoted as $\mathbf{C}^{(t)}$. We thus define $T^{(c)} = \{t \in \boldsymbol{T} \ s.t. \ \mathbf{C}^{(t)} = c\}$ to be the set of time instances labeled as class $c$, where $\boldsymbol{T}$ is the set of all time instances. Further, $N^{(c)} = \left|T^{(c)}\right|$ is the number of CSI frame samples marked as class $c$. From this, we define $P^{(c)}_{samples}$ as:

$$P^{(\mathrm{c})}_{samples} = \frac{1}{N^{(\mathrm{c})}} \sum_{t \in \boldsymbol{T}^{(\mathrm{c})}} Y(t, c) \tag{6.6}$$

where

$$Y(t, c) = \begin{cases} 1 & \textit{if } A_{CSI,t} \geq \tau \text{ and } c = \text{'target'} \\ 1 & \textit{if } A_{CSI,t} < \tau \text{ and } c = \text{'no target'} \\ 0 & \text{otherwise} \end{cases} \tag{6.7}$$

We can interpret $P^{(\mathrm{c})}_{samples}$ as the percentage of CSI frame samples which are truly class $c$ and are predicted as class $c$; or put simply, the true-positive and true-negative rates. In Fig. 37a we see the results of our model. As it is expected, as $\tau$ increases, $P^{(\text{'no target'})}_{samples}$ increases and $P^{(\text{'target'})}_{samples}$ decreases. Specifically, when $\tau < 1.0$, $P^{(\text{'no target'})}_{\text{samples}} = 0.0$ and $P^{(\text{'target'})}_{samples} = 1.0$, this is because there are no samples where $A_{CSI} < 1.0$. In addition to this, we can see that there is no value for $\tau$ where we are able to achieve perfect accuracy on predicting both the true positives and true negatives. However,

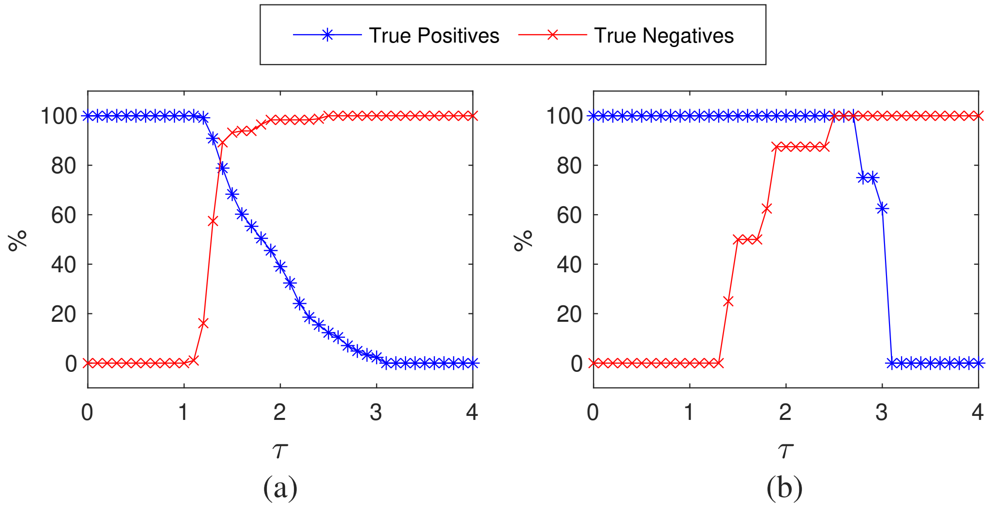


Fig. 37. Prediction accuracy as threshold parameter $\tau$ changes. (a) With all recorded samples using $P_{samples}$. (b) With all independent action segments using $P^{(c)}_{segments}$.

our goal is not to predict the action for all time instances individually. Instead, we are only interested in correctly classifying each action segment overall.

To define action segments, we first collect a set of time instances ($\mathbf{I}$) which indicate the beginning and ending of different actions. To determine these indices, we apply the following:

$$\mathbf{I} = \{0\} \cup \left\{t \in \{2 : \mathbf{T}\} \textit{ where } C^{(t-1)} \neq C^{(t)}\right\} \cup \{\mathbf{T}\}. \tag{6.8}$$

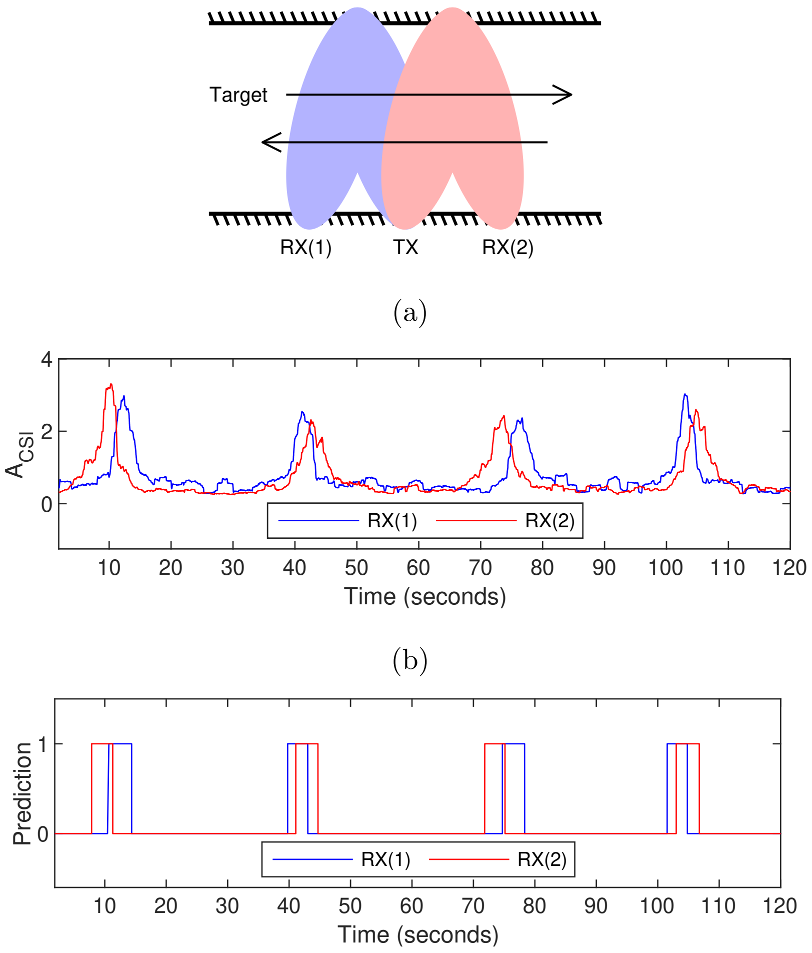
With this, we can describe the number of action segments recorded $N_{seg} = |\mathbf{I}| - 1$. We say that the target is predicted as present ($P^{(i)}_{\text{`}target\text{'}}$) during some action segment $i$ if $\exists t \in \{\mathbf{I}^{(i)} : \mathbf{I}^{(i+1)}\}$ $s.t.$ $A_{CSI,t} \geq \tau$. Action segments containing no target on the other hand are denoted $P^{(i)}_{\text{`}no\ target\text{'}}$, which is simply the negation of $P^{(i)}_{\text{`}target\text{'}}$. The number of segments for a given class is described as $N^{(c)}_{seg}$. To evaluate our predictions on all segments, we define:

$$P^{(c)}_{segments} = \frac{1}{N^{(c)}_{seg}} \sum_{i=1}^{N_{seg}} \begin{cases} 1 & \textit{if } P^{(i)}_{(c)} \text{ and } \mathbf{C}^{(t)} = c \\ 0 & \text{otherwise} \end{cases} \tag{6.9}$$

which is described as the percentage of segments correctly labeled as class $c$. With this, our final goal is to find a value for $\tau$ such that we maximize the number of segments where a target was present and minimize the number of time segments predicted as containing a target when no target was present. When considering this segmented approach, we see in Fig. 37b that when $\tau \in [2.5, 2.7]$ both the true positive and true negative rate reach 1.0, indicating a range of perfect predictions.

### 6.3.2 Human Direction

Our next task is to recognize the moving direction of the target in the hallway environment. While we show in Section 6.2 that $A_{CSI}$ reveals human presence, moving direction of the target is not directly revealed by the metric. To address this, we use two receiving devices, one located to the left of the transmitter and the other to the right as shown in Fig. 38a. Both receivers again use directional shielding so that when the central transmitter sends radio signals, they are both able to receive the signals for different areas within the hallway environment. The expectation is that when a target moves from left to right, the device located to the left-most side will recognize the target first, then later on, the right-most device will recognize the target. Afterwards, we would expect the left-most device will stop recognizing the target before the right-most device. In Fig. 38b we see $A_{CSI}$ for both RX(1), which is placed to the left-hand side of the TX, and RX(2), which is placed to the right. As the target moves back and forth through the hallway environment, $A_{CSI}$ for RX(2) increases before $A_{CSI}$ for RX(1) indicating that the user moved in the direction of right-to-left. In the second



(a)

(b)

(c)

Fig. 38. Using two receivers we are able to identify the directional movement of the human target based on which receiver sees an increase in $A_{CSI}$ first. (a) Experiment setup with all adversarial ESP32 devices on one side of the wall: TX at the center, RX(1) to the left of TX and RX(2) to the right. (b) Raw $A_{CSI}$ showing four peaks when the target moves back and forth within the hallway environment, (c) After applying binary human detection algorithm, we can even more clearly identify the human target direction.

pair of peaks at around 40 seconds, $A_{CSI}$ for RX(1) increases first, indicating that the target moved back to the starting point from left-to-right. By applying the binary human detection algorithm from Section 6.3.1, we can see this relationship even more clearly as shown in Fig. 38c.

## 6.4 Chapter Contributions and Summary

In this chapter, we studied the use of Channel State Information for adversarial through-wall occupancy monitoring in hallway environments. We demonstrated through real world experiments how an attacker could perform surveillance of a building if given access to a single room or even from a single exterior wall. Through the use of our previously developed WiFi sensing toolkit [102], we demonstrated how this sort of attack is very low cost and much easier to conceal compared to camera-based surveillance methods. Using the signal pre-processing steps proposed in this work, we were able to demonstrate that the two components required for tracking humans, namely, presence and moving direction, can be successfully predicted even in one-sided through-wall scenarios which can then be used for crowdcounting by adversaries.

# CHAPTER 7

# SPATIAL ANTENNA DEFENSE AGAINST WIFI SENSING EAVESDROPPERS

## 7.1 Introduction

Despite the benefits of using ubiquitous WiFi sensing for several diverse applications, there is still an inevitable security and privacy risk of WiFi sensing. That is, an adversary (e.g., Eve in Fig. 39) sniffing the WiFi signals in the environment can track private information about the users (e.g., if they are at home or not, or even which room they are in [87], their walking direction behind the wall [102]) and leverage this information for malicious purposes. Recognizing this risk, in some WiFi sensing studies [218], the machine learning model used in WiFi sensing is trained in a way such that only the allowed behaviors (e.g., falling of a senior) can be sensed properly while private activities (e.g., bathing) are prevented. However, such solutions provide only partial protection as it assumes that the trained model is the source of potential privacy leakage only. However, ambient WiFi signals can be sniffed by an eavesdropper and CSI data can be used for detection of activities using a pretrained environment-independent model [196, 148, 191] (i.e., a model trained using CSI data collected from different environment(s) but can perform accurate predictions in a totally new environment).

There are some recent efforts that aim to protect CSI signals from adversaries and thus invalidate their proper WiFi sensing capability. However, they are either more complicated as they use specialized hardware (e.g., using USRP [224], IRS[225]), and are not easy to implement in practice. Moreover, some of the solutions aim to totally

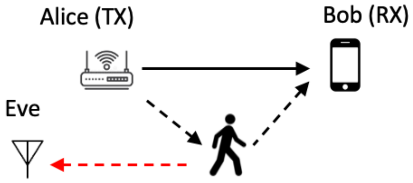


Fig. 39. A malicious eavesdropper (i.e., Eve) can obtain CSI data to perform adversarial WiFi sensing with a pretrained environment-independent ML model.

avoid WiFi sensing even for legitimate devices, thus these solutions are not desirable. Our goal is to allow legitimate WiFi sensing with allowed receiver (RX) devices but prevent illegitimate RX devices or eavesdroppers from performing adversarial WiFi sensing. To this end, we propose a WiFi sensing solution where multiple spatially distributed transmitter (TX) antennas are used to transmit WiFi packets to the RX.

The rest of the chapter is organized as follows. In Section 7.2, we provide a background on WiFi sensing and discuss the literature in particular in adversarial WiFi sensing and solutions to avoid it. In Section 7.3, we present our system model together with the assumptions made and attacker and defense models. We then present our motivation for this work in Section 7.4 and evaluate how our method can prevent eavesdroppers from performing WiFi sensing through experiments in Section 7.5. Finally, we provide additional discussion about our method in Section 7.6 and make our concluding remarks in Section 7.7.

## 7.2 Preliminaries

### 7.2.1 Related Work

With the growing number of studies (e.g., [87, 230]) showing various levels of activity information and location leakage through adversarial WiFi sensing systems,

Table 25. Comparison of Existing Defense Methods

| References | Method | Issues |
|---|---|---|
| [219],[220],[221] | Transmitter altered signals | By altering the signals, the data is no longer valid WiFi frames. |
| [222] | Signal strength variations | Reduces communication capacity of the network. |
| [223],[224] | External obfuscator node | Requires an additional device used solely for noisy transmissions. |
| [225] | Intelligent reflecting surface (IRS) | Requires hardware with low consumer usage. |
| [226],[227] | Omnidirectional jammer | Prevents all legitimate sensing and communication. |
| [228] | Directional jammer | Requires additional physically moving devices. |
| [229] | Alters signals to emulate activities | Requires specialized USRP equipment and non-standard WiFi frames. |

developing counter mechanisms has become a necessity. Thus, several recent studies have looked at this problem and proposed different solutions. Table 25 provides a summary of existing defense mechanisms against eavesdropping with WiFi sensing. In [219], an obfuscation based solution is proposed which captures ambient wireless signals and relays them back into the environment with randomized modifications. However, the proposed solution uses full-duplex radio which requires specialized and costly hardware. A similar approach without using full-duplex is studied in [228], but it uses a motorized component to change the orientation of the antenna and introduces randomized delay. In [222], a solution is proposed which varies signal strengths of the transmitters and a game-theoretical model is studied between the attacker and defender considering the trade-off between privacy and utility in the system. This can however reduce the communication capacity between the devices.

Jammer-based solutions [227, 226], introduce randomized signal noise to prevent proper sensing. However, these solutions hamper the communication, thus they may not be practical in most of the real-life scenarios. Instead, in [224], a selective obfuscating solution is proposed to avoid extraction of location information from CSI. The solution superimposes a duplicated copy of the signal on each frame which does not affect the reception but does hinder the location-relevant information. However, this is mainly for protection of location and not applicable to activity detection use cases.

In [229], a modification to the radio training system is proposed to change the transmitted symbols over time, space and frequency as if they are affected due to human activities in the environment. While this approach prevents eavesdroppers from distinguishing real and fake human gestures, due to the requirement of specialized hardware (e.g., USRPs), it incurs a high cost and will not be practical. Note that our work also differs from the studies (e.g., [221]) that look at solutions against malicious radiometric fingerprinting of devices. These studies focus on the device-specific fingerprinting which could be used for impersonation attacks.

## 7.3 System Model

### 7.3.1 Assumptions

In our proposed system, we assume multiple TX antennas that are spatially distributed in a target sensing area as illustrated in Fig. 40. There is a single source device ($\mathcal{D}$) that is equipped with an antenna switch to automatically select the transmitting antenna at a per-packet level. This ensures that low layer attributes (e.g., MAC address and sequence number) will not directly reveal antenna changes to the eavesdropper. Usage of TX antennas are determined by a predefined schedule model ($\mathcal{S}$) which is shared between $\mathcal{D}$ and any legitimate RX devices. When evaluating our proposed system, only a single TX antenna communicates at any given time, however, $\mathcal{S}$ may be extended to allow multiple antennas to communicate simultaneously.

### 7.3.2 Experiment Setup

In our experiments, we consider a home environment where ESP32 microcontrollers are used as both TX and RX devices using the 802.11n protocol and 2.4GHz frequency band for CSI collection. Five TX devices that are placed 70 centimeters apart in a room of size 2.8 meters × 3.2 meters and 1 RX device are used as illustrated in Fig. 40. The RX is placed in an adjacent room to emulate an attacker which does not have direct access to the targeted sensing area. Each of the five TXs transmit WiFi frames to the RX at 20Hz concurrently resulting in CSI samples arriving at the RX at an overall rate of 100Hz. During our experimental data collection we allow all TXs to transmit concurrently so that we can emulate different transmission schedules, however in a real world system, only selected TXs will transmit at any given point of time. We collect CSI data to a Raspberry Pi single-board computer for processing.

For our dataset[1], we perform the following 5 activities:

- **Door:** Opening/closing main door
- **Sit:** Sitting and swiveling on a chair at a desk
- **Stand:** Standing at a desk and writing in a book

[1] Dataset and codebase for this project can be found in https://github.com/MoWiNG-Lab/AntiEave-WiFi-Sensing.

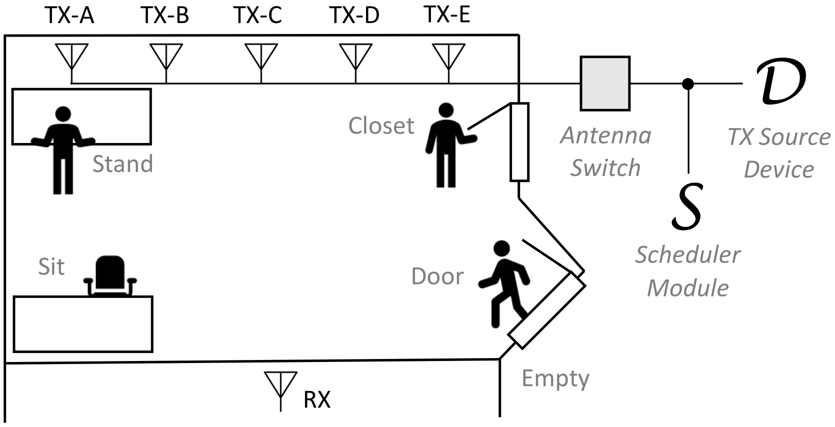


Fig. 40. Experimental setup with 5 TXs and 1 RX and 5 activities to be sensed. The scheduler ($\mathcal{S}$) decides which of the TXs that are wired connected to the same source device ($\mathcal{D}$) through an antenna switch needs to transmit.

- **Closet:** Opening/closing closet door
- **Empty:** No movement within the room

These activities are performed in a round-robin fashion 6 distinct times. The first set of 3 repetitions are used for training our model while the final 3 are used for evaluation. Note that while the activities considered in this dataset are performed in diverse locations, WiFi sensing techniques are also applicable when multiple activities are performed at the same location [37, 231]. Additionally, while we use multiple ESP32s to act as TX antennas during our data collection phase, a similar system can also be achieved with a single WiFi device (e.g., one ESP32) and an antenna switch as illustrated in Fig. 40.

### 7.3.3 Tree-structured Parzen Estimator (TPE)

For our evaluations, we use the Tree-structured Parzen Estimator (TPE) [232] which is a hyperparameter optimization technique which selects some set of hyperparameters ($\theta$) in an attempt to decrease some loss function $\mathcal{L}$ through the use of an expected improvement (EI) function:

$$EI(\theta) = \frac{p\big(\theta|\mathcal{L}'(\theta) > \mathcal{L}^*\big)}{p\big(\theta|\mathcal{L}'(\theta) \leq \mathcal{L}^*\big)}, \tag{7.1}$$

where $\mathcal{L}^*$ is the average loss of the previously evaluated hyperparameter values for a given hyperparameter and $\mathcal{L}'(\theta)$ is formed based on previously observed hyperparameter values.

### 7.3.4 Attack Model

We consider a scenario where an attacker aims to sense the activities performed by an individual and subsequently localize the individual using the temporal CSI data obtained from the sniffed ambient WiFi signals. Fig. 39 illustrates our scenario where Alice transmits a signal to Bob. As the signals propagate through the environment, some of them reflect off of the human within the environment before continuing to propagate to Bob, thus allowing Bob to perform WiFi sensing. However, a malicious eavesdropper (Eve) can also receive the reflected signal which then allows Eve to perform WiFi sensing and thus Eve can achieve covert surveillance.

We assume that the attacker knows the set of localized activities and has a pretrained ML model for these activities. We also assume that this model is obtained through the solutions in the literature that offer environment-independent ML models [196, 148] or generic models that are obtained through a federated learning process [191]. However, we consider CSI data generated from both a single TX and multiple TX devices for training the attacker's model. Additionally, we assume that the attacker has a device and a tool that can extract CSI data from sniffed signals, which can be easily achieved through recent low-cost off-the-shelf solutions [102]. Attacker then uses this extracted CSI for predictions using the pretrained model. Similar

Table 26. Scenarios considered during training and evaluation.

| Scenario | Train on CSI from | Evaluate on CSI from | Section |
|---|---|---|---|
| Normal Sensing | Single TX | Single TX | Section 7.4 |
| Naive Eve | Single TX | Multiple TX | Section 7.5.1 |
| Advanced Eve | Multiple TX | Multiple TX | Section 7.5.2 |

to the training scenario, we look at the predictions when the attacker uses CSI data received from (i) a single TX and (ii) multiple TXs. These scenarios and the sections looking at the evaluation of each scenario are given in Table 26.

### 7.3.5 Defense Model

To prevent eavesdroppers from sensing physical activities using WiFi sensing without also hindering allowed RX devices from sensing physical activities, we leverage a multi-TX setup as illustrated in Fig. 40. We define a scheduler $\mathcal{S}$ which pseudorandomly decides which TX should transmit at a given time instance, $t$, such that $\mathcal{S}(t) \in \{1, 2, \ldots, |TX|\}$. Allowed RX devices are given access to $\mathcal{S}$ which ensures that they are able to accurately identify which TX is transmitting at any given time while the disallowed eavesdropper is unable to make this distinction. To further obfuscate the physical activity and reduce the sensing capability of the eavesdropper, we define specific probability values for each TX device to determine how often the TX is selected from our scheduler module ($\mathcal{S}$).

### 7.3.6 Allowed RX Emulation

To evaluate the proposed system, we begin by describing the CSI data as seen by the allowed RX. This RX can recognize which TX is transmitting at any given time (from scheduler information). In our evaluations, we begin with a 3-dimensional CSI tensor $\mathbb{H} \in \mathbb{R}^{|\mathcal{T}|\times|TX|\times|s|}$ where $|\mathcal{T}|$ is the number of time steps in our dataset, $|TX|$ is the number of transmitters, and $|s|$ is the number of subcarriers per CSI frame. We apply a transformation to $\mathbb{H}$ based on our scheduler model $\mathcal{S}$ as so:

$$\big(\mathbb{H} \oplus \mathcal{S}\big)[t, i, :] = \mathbb{H}[t, i, :] * \mathit{soft\_equals}(i, \mathcal{S}(t)), \tag{7.2}$$

where $\mathbb{H}[t, i, :]$ is a tensor slice of all subcarriers for station $i$ collected at time $t$ and

$$\mathit{soft\_equals}(a, b) = 1 - \mathit{tanh}(|a - b|\beta), \tag{7.3}$$

which has an output approaching 1 when $a = b$, and 0 when $a \neq b$ and when $\beta$ is some large value (i.e., $\beta = 1e4$). Through this, the allowed RX receives a tensor $\big(\mathbb{H} \oplus \mathcal{S}\big) \in \mathbb{R}^{|\mathcal{T}|\times|TX|\times|s|}$, however for each $i \in \{1, 2, \ldots, |TX|\}$ which is not selected at time $t$, the values for $\big(\mathbb{H} \oplus \mathcal{S}\big)[t, i, :] = \mathbf{0}$ because the model being trained would not be able to witness the CSI for the $i$-th TX.

### 7.3.7 Disallowed (Eavesdropper) RX Emulation

Now that we have reviewed the CSI data as seen by an allowed RX, next we review the CSI data as seen by a disallowed RX (i.e., an eavesdropper). The only difference in the allowed RX versus the disallowed RX is that the disallowed RX is not able to directly identify the difference between which TX is actively transmitting at any given time. As such, we define:

$$\begin{aligned}\big(\mathbb{H} \,\check{\oplus}\, \mathcal{S}\big)[t, :] &= \sum_{i=1}^{|TX|} \Big(\big(\mathbb{H} \oplus \mathcal{S}\big)[t, i, :]\Big) \\ &= \big(\mathbb{H} \oplus \mathcal{S}\big)[t, \mathcal{S}(t), :],\end{aligned} \tag{7.4}$$

where $\left(\mathbb{H} \check{\oplus} \mathcal{S}\right) \in \mathbb{R}^{|\mathcal{T}| \times |s|}$. It is important to note that while $\left(\mathbb{H} \oplus \mathcal{S}\right)$ and $\left(\mathbb{H} \check{\oplus} \mathcal{S}\right)$ have different tensor shapes, they both contain the same amount of CSI amplitude information, meaning that they both have the same number of non-zero entries within the tensors. However, $\left(\mathbb{H} \oplus \mathcal{S}\right)$ encodes slightly more information due to the structure itself which is derived due to the knowledge of the transmission schedule shared between the TXs and RX.

## 7.4 Motivation

In our initial efforts to motivate the multi-TX based proposed solution, we begin by presenting our experiment results for a human activity detection and localization scenario. We train a Dense Neural Network (DNN) machine learning classifier model $\mathcal{M}_{\text{TX-}m}$ per TX-$m$ with one input dense layer, two hidden dense layers and one dense output layer. We apply $L_2$ kernel regularization across each dense layer and apply a dropout layer between each dense layer to prevent the model from overfitting. Finally, we use Stochastic Gradient Descent (SGD) to optimize the loss function

$$\mathcal{L}(x, y) = -\frac{1}{|x|} \sum_{i=1}^{|x|} \sum_{c=1}^{|C|} y_{i,c} \log \mathcal{M}_{\text{TX-}m}(x_{i,c}), \tag{7.5}$$

where $\mathcal{M}_{\text{TX-}m}(x_{i,c})$ is the model prediction for input CSI $x_{i,c}$ and $y_{i,c}$ is the true class for the $i$-th CSI measurement. We apply a preprocessing step to transform the raw CSI through Principal Component Analysis (PCA), which is shown to be one of the most effective preprocessing methods for increasing prediction accuracy in WiFi sensing scenarios [233].

We begin by showing how the accuracy of a WiFi sensing system is affected by the different physical positions of the five TXs relative to the RX as well as relative to the actions being performed. To this end, we train an ML model on training data captured by each TX and then evaluate the models on the testing data from the

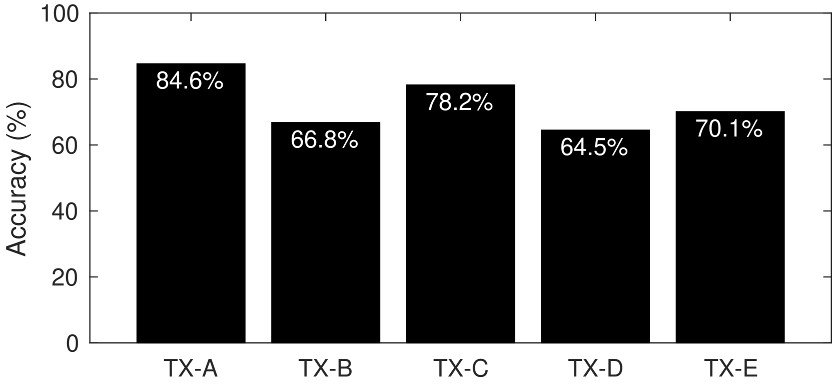
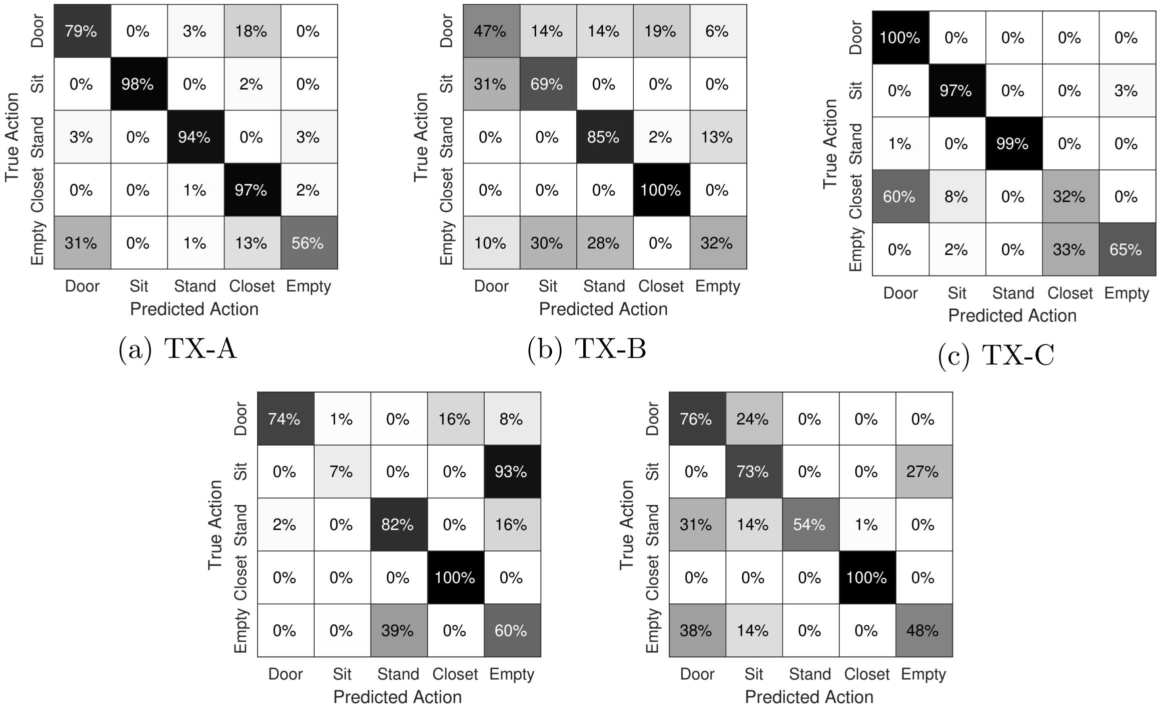


Fig. 41. Accuracy with ML models developed by CSI data coming from each TX.

same TX. The accuracy for the models trained at each TX is shown in Fig. 41. We can see that TX-A achieves the highest accuracy at 84.59% and TX-D achieves the lowest accuracy at 64.47%. This demonstrates that the accuracy possible from each TX varies due to the unique physical positions of the TXs within the environment.

The confusion matrices in Fig. 42 show which classes of actions are accurately predicted and which classes are commonly predicted incorrectly per TX. From this, we can see that each TX is better at distinguishing different sets of activities due to the spatially distributed nature of the TXs in the environment as well as the unique physical locations where each physical activity is performed. For example, TX-A achieves high classification accuracy on classes *sit, stand, closet*, TX-C achieves high classification accuracy on classes *door, sit, stand* and TX-B, TX-D, TX-E can each distinguish the *closet* action with high accuracy. This means that each of the TXs has unique strengths as well as unique weaknesses in our experiment scenario. In the next section, we will evaluate how we can leverage these differences due to TX positioning against a malicious eavesdropper.



(a) TX-A

| | Door | Sit | Stand | Closet | Empty |
|---|---|---|---|---|---|
| Door | 79% | 0% | 3% | 18% | 0% |
| Sit | 0% | 98% | 0% | 2% | 0% |
| Stand | 3% | 0% | 94% | 0% | 3% |
| Closet | 0% | 0% | 1% | 97% | 2% |
| Empty | 31% | 0% | 1% | 13% | 56% |

(b) TX-B

| | Door | Sit | Stand | Closet | Empty |
|---|---|---|---|---|---|
| Door | 47% | 14% | 14% | 19% | 6% |
| Sit | 31% | 69% | 0% | 0% | 0% |
| Stand | 0% | 0% | 85% | 2% | 13% |
| Closet | 0% | 0% | 0% | 100% | 0% |
| Empty | 10% | 30% | 28% | 0% | 32% |

(c) TX-C

| | Door | Sit | Stand | Closet | Empty |
|---|---|---|---|---|---|
| Door | 100% | 0% | 0% | 0% | 0% |
| Sit | 0% | 97% | 0% | 0% | 3% |
| Stand | 1% | 0% | 99% | 0% | 0% |
| Closet | 60% | 8% | 0% | 32% | 0% |
| Empty | 0% | 2% | 0% | 33% | 65% |

(d) TX-D

| | Door | Sit | Stand | Closet | Empty |
|---|---|---|---|---|---|
| Door | 74% | 1% | 0% | 16% | 8% |
| Sit | 0% | 7% | 0% | 0% | 93% |
| Stand | 2% | 0% | 82% | 0% | 16% |
| Closet | 0% | 0% | 0% | 100% | 0% |
| Empty | 0% | 0% | 39% | 0% | 60% |

(e) TX-E

| | Door | Sit | Stand | Closet | Empty |
|---|---|---|---|---|---|
| Door | 76% | 24% | 0% | 0% | 0% |
| Sit | 0% | 73% | 0% | 0% | 27% |
| Stand | 31% | 14% | 54% | 1% | 0% |
| Closet | 0% | 0% | 0% | 100% | 0% |
| Empty | 38% | 14% | 0% | 0% | 48% |

Fig. 42. Confusion matrix for each model in Fig. 41.

## 7.5 Evaluation

We demonstrated how CSI captured from a single TX can be used to predict the localized physical activity of humans in an environment. However, achieving high accuracy in the previous scenario not only means that legitimate RXs can sense actions being performed, but it also means that malicious eavesdroppers can also covertly perform surveillance on the human target by sniffing these same signals.

To obfuscate the physical actions being performed in the environment, we allow the TXs to transmit one at a time on a random schedule every 50ms as illustrated in Fig. 43. This random schedule is emulated during our evaluations using the data collected and described in the previous section, but in a real-world deployment, we can assume that each TX adheres to the random schedule.

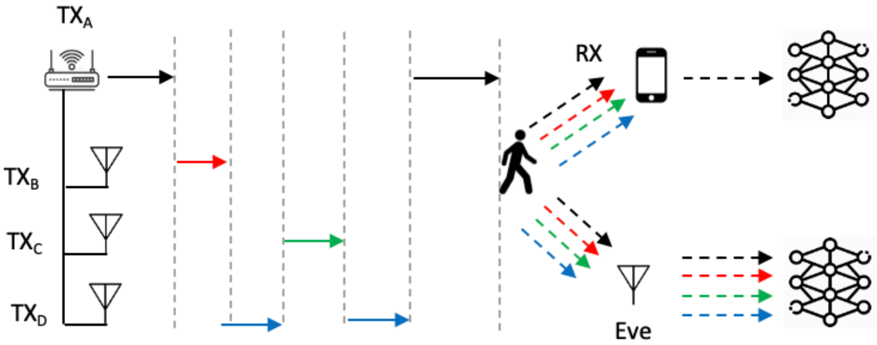


Fig. 43. Multiple TX antennas are used to transmit the WiFi signals at different times based on a predefined schedule known by a legitimate RX device, which then can filter the necessary CSI data for use in the prediction model, while eavesdropper uses all CSI and obtains inaccurate results.

We study two scenarios: (i) a naive attacker that is not aware of the multiple TX antennas thus uses a sensing model trained with CSI data from one TX, and (ii) a more intelligent and advanced attacker which trains a sensing model using CSI from multiple TXs which are generated based on the scheduler. In the latter, however, we still assume that the attacker does not know which packet comes from which TX device.

Note that it is not trivial for an advanced attacker to generate a pretrained environment independent model using CSI data from multiple TX locations (as done in the single source with a single antenna scenario [196, 148, 191]). This is because different spatial distribution of TX devices with respect to a receiver device can generate different results. However, to explore the extent to which an attacker can achieve sensing, we assume that the attacker is able to acquire CSI data from the same spatial distribution of TX devices as in the environment of interest along with the corresponding labels for each activity.

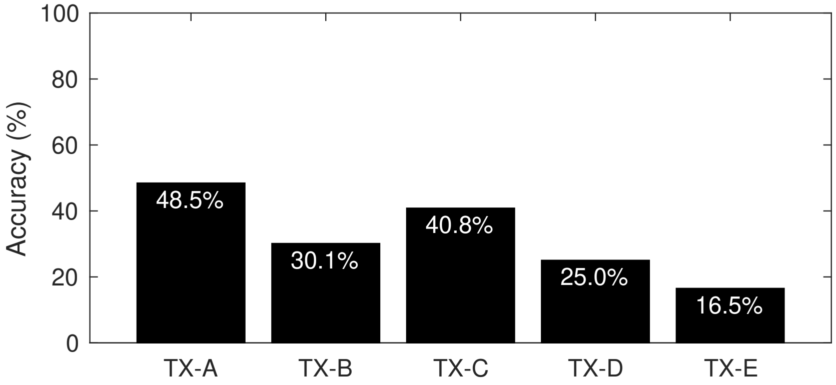


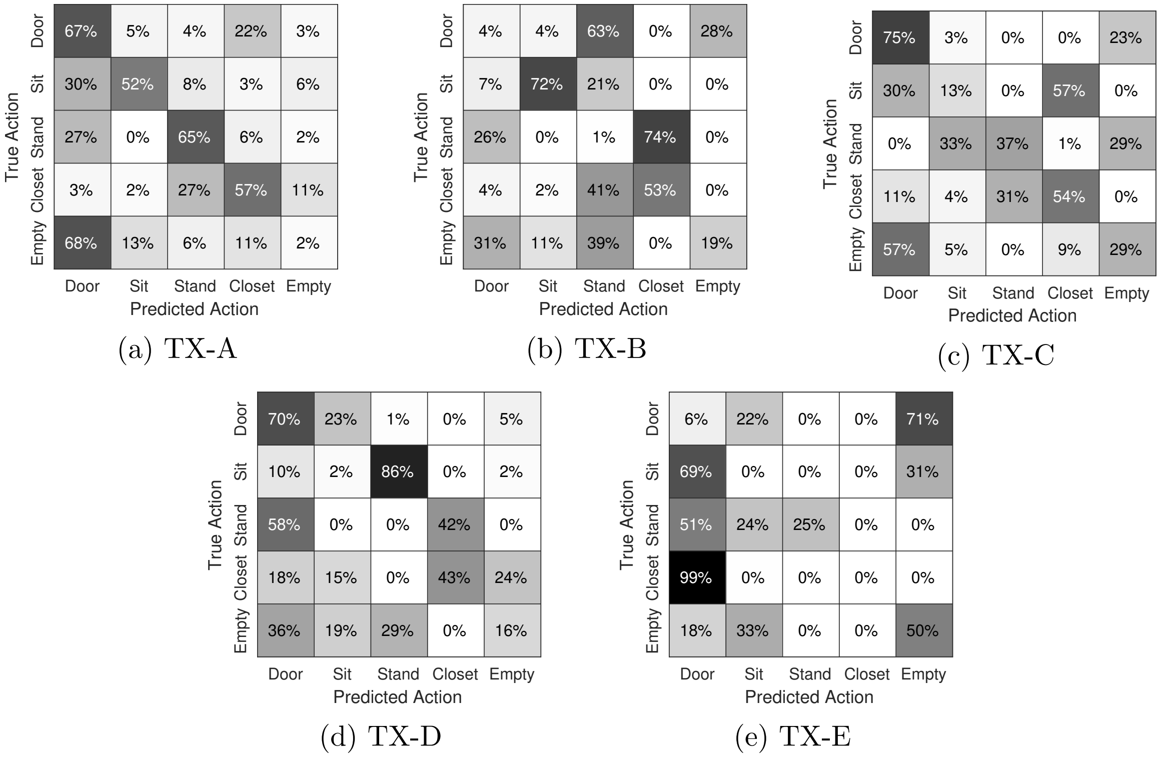
Fig. 44. Accuracy of eavesdropper's model trained on a single TX CSI data and used in obfuscated CSI data from all 5 TXs on a random schedule.

### 7.5.1 Naive Attacker

We begin by evaluating the naive attacker which considers that there is only one TX in the environment communicating with an RX device to generate the necessary signaling for WiFi sensing. As such, the model that is trained by this naive attacker will likely be confused by the CSI data coming from multiple TX antennas located in unique physical positions.

In Fig. 44, we can see the accuracy of the eavesdropper model when trained on CSI from a single TX and then applied to our obfuscation scenario where 5 TXs transmit on a random schedule. Since the eavesdropper does not know the random order of the transmitting devices, the eavesdropper must assume the use of all incoming CSI frames. We can see that the accuracy for each of the TX models has decreased significantly by as much as 53.5% for TX-E and a decrease of accuracy more than 35% for all other TXs (compared to the results in Fig. 41). Overall, this suggests that increasing the number of TXs even beyond five will allow for an ever lower accuracy for the naive attacker.

Fig. 45 shows the confusion matrix for each of the eavesdropper models in this



(a) TX-A

(b) TX-B

(c) TX-C

(d) TX-D

(e) TX-E

Fig. 45. Confusion matrix for each model in Fig. 44.

same scenario. These figures show that our random scheduling method causes the eavesdropper model to randomly and incorrectly guess the current action being performed in the environment. Unlike Fig. 42 where each TX was able to achieve greater than 80% accuracy for more than one class, with our random scheduling method, the eavesdropper is unable to predict any of the individual classes with an accuracy greater than 80% for any of the TX models. The class that is most accurately predicted for the eavesdropper would be the *door* class using the model trained at TX-C. However, because the remaining predictions are so poor, it is not reasonable for an eavesdropper to believe that these predictions are correct. For example, while *door* is correctly predicted 75% of the time, *empty* is incorrectly predicted to be the *door* class 57% of the time and similarly, *sit* is incorrectly predicted to be the *door* class 30% of

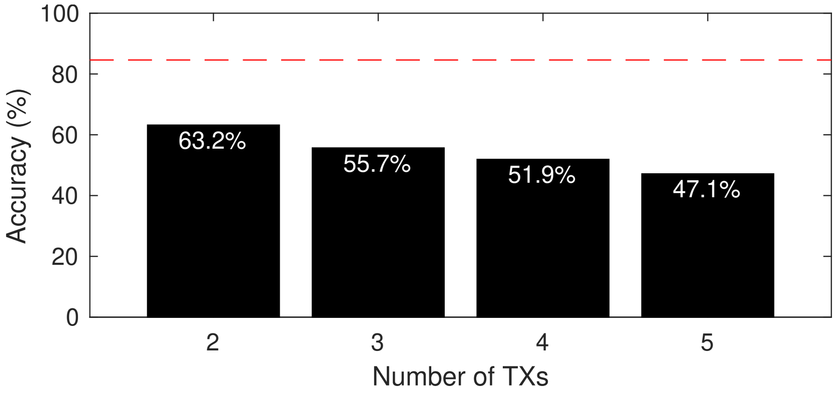


Fig. 46. Accuracy of an eavesdropper with different number of TXs communicating on a random schedule. Red dashed line shows the accuracy if random scheduling was not used.

the time. Thus, because the accuracy is so poor for most of the action classes, any accurately predicted class cannot be distinguished from incorrectly predicted classes by the eavesdropper, thus rendering the predictions useless.

Another fascinating observation is that the TX-E model incorrectly predicts *sit, stand, closet* classes most often to be the *door* class, yet the *door* action is rarely ever predicted correctly. This suggests that our method can be used to deceive the eavesdropper such that the eavesdropper will have a high propensity for predicting one given class while also concealing the action when it actually occurs in the environment.

Now that we have evaluated the eavesdropper model when 5 TXs are used in our random schedule, we next look at the accuracy of our system when different numbers of TXs are used during evaluation. In Fig. 46, we evaluate the accuracy for a model trained at TX-A and then evaluate when multiple TXs are used in the random schedule including TX-A and some number of other TXs. The red dashed line shows the accuracy (84.59%) of the model when the random schedule was not applied. We can see that the accuracy decreases as more TXs are added to our random schedule, however, even when the number of TXs is 2 (i.e., TX-A and one other TX), the accuracy is 63.2% which is far lower than the 84.59% that could be achieved without the random schedule.

### 7.5.2 Advanced Attacker

In the previous scenario, we assumed that the eavesdropper naively trains a model using CSI collected from a single TX and then applies this model in a randomly scheduled multi-TX setting. However, a more advanced eavesdropper may train their model using CSI collected from all TXs as they actively communicate in the environment. As such, in this section, we begin by evaluating the advanced attacker in a random station schedule scenario. After this, our goal is to identify a transmission schedule which reduces the ability of the eavesdropper to perform sensing.

#### 7.5.2.1 Random Schedule

In order to test the accuracy of models generated by Eve using the multi-TX data, we initially consider a random schedule of TXs in the system. Eve trains a model based on the data from all TXs using this random schedule, then the model is also used for predictions again using the CSI data from all TXs involved. In Table 27, we review two forms of random TX scheduling, namely: periodic and non-periodic. In the periodic case, we create a pseudo-random schedule of size $w$ which is repeated across the entire dataset. For the non-periodic case, we create a pseudo-random schedule across all timesteps within our dataset without actively ensuring periodicity. From this, we can observe that the periodic case allows Eve to achieve an accuracy of +29.62% greater than the non-periodic case. This demonstrates that any repeating patterns in the transmission schedule will actually improve the accuracy

Table 27. Eavesdropper accuracy with periodic and non-periodic random schedulers ($N = 50$ each).

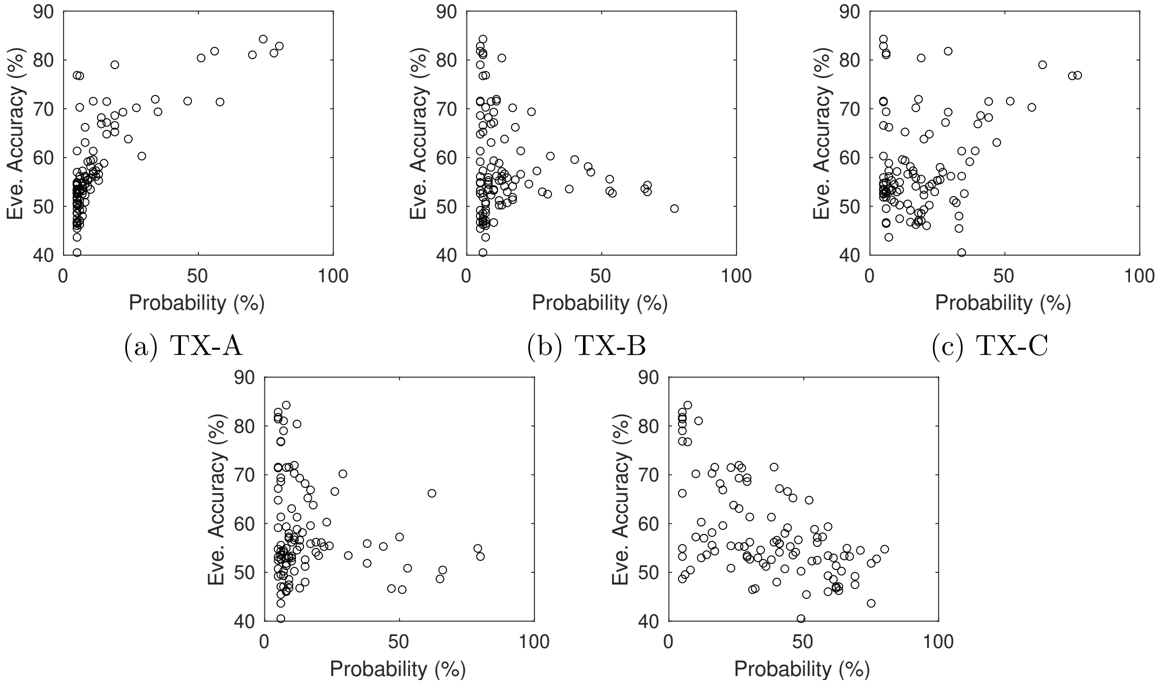
| Type | Avg. Accuracy (Std. Dev.) |
|---|---|
| Non-Periodic | 56.58% (±9.90%) |
| Periodic | 86.20% (±1.75%) |

of Eve compared to a single-TX system (i.e., 86.20% is greater than all accuracy values in Fig. 41). This also demonstrates that an advanced attacker can achieve greater prediction accuracy (i.e., 56.58%) compared to a naive attacker (i.e., 48.5% with TX-A in Fig. 44) but still less accuracy than if only a single TX was used in the environment (i.e., 64.5% worst-case with TX-D in Fig. 41).

#### 7.5.2.2 Probabilistic Schedule

In our previous experiments, we observed that each TX can achieve different levels of accuracy. For example, TX-A achieves the greatest accuracy in Fig. 41 at 84.6% while TX-D only achieves the lowest accuracy of 64.5%. We propose that we can leverage this knowledge to determine a schedule by setting pseudo-random probabilities uniquely per-station. Since different environments will have different TXs which achieve the best and worst sensing accuracy values, thus, we propose a learning approach to determine these pseudo-random per-station probabilities. Specifically, TPE determines optimal hyperparameter values for the probability for each station.

Towards this, when selecting the per-station probabilities, we define $0 \leq m \leq \frac{100}{|TX|}$, the minimum probability that all TXs are selected. In our experiments, since we have 5 TXs, the maximum value for $m$ is 20%. The order in which hyperparameters are selected is important to ensure that the entire search space is explored by TPE.



(a) TX-A

(b) TX-B

(c) TX-C

(d) TX-D

(e) TX-E

Fig. 47. Eavesdropper accuracy for different per-station probabilities when using TPE ($N = 100$, minimum per-station probability: 5%).

We find that if we use TPE to select a station probability in order for TX-A, TX-B, . . . , TX-E, then TX-E will inevitably result in only very low probability values being explored due to it being the last selected probability value. As such, we instead select the probabilities of each TX in a random order for each TPE trial, thus allowing the full search space to be explored.

In Fig. 47, we illustrate the results of TPE when $N = 100$ TPE trials are performed and the minimum per-station probability $m = 5\%$. In this figure, we can see that as the probability for TX-A increases, the eavesdropper accuracy increases as well, thus TPE is able to recognize that low-values are more useful for our experiment environment. TX-C shows a similar upward trend while TX-B, TX-D, and TX-E show a negative trend as probability increases for each station. This is under-

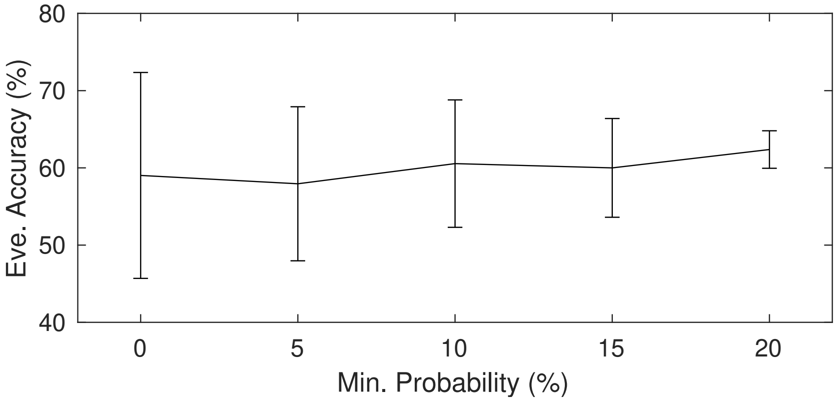


Fig. 48. Effect of minimum per-station probability on eavesdropper accuracy ($N = 100$ each).

standable considering that these TXs achieve the lowest accuracy values in Fig. 41 when evaluated on their own. While high probability values for TX-E appear to achieve the lowest accuracy for Eve, the achievable accuracy distribution range is wide, demonstrating that high probability values for TX-E do not always translate to the same low accuracy for Eve. This may be due to other TXs like TX-A or TX-C being selected along with TX-E in those trials.

Next we consider how applying different minimum probability values for $m$ affects accuracy of Eve. By applying a minimum probability for all TXs, we can ensure that we leverage all of the available hardware which is deployed in the environment. Since our experimental design uses five TXs, when $m = 20$, each TX is selected equally by $\mathcal{S}$, however, with $m = 0$, it is possible that some TXs are unused for communication and sensing. In Fig. 48, we show the mean and standard deviation of $N = 100$ TPE trials when $m \in \{0, 5, 10, 15, 20\}$. The average accuracy of Eve decreases slightly as $m$ decreases from 62.38% when $m = 20$ down to 59.02% when $m = 0$. However, the standard deviation increases from 2.43% when $m = 20$ up to 13.33% when $m = 0$. This is because with low values of $m$, there are more possibilities for better Eve accuracy as well as lower Eve accuracy values. This demonstrates that allowing some

Table 28. Average accuracy ($N = 25$ each) for different per-station probabilities.

| Station Probabilities | Eavesdropper Accuracy | | Allowed-RX |
|---|---|---|---|
| TX-A, B, C, D, E | TPE | Avg. ($N = 25$) | Avg. Accuracy ($N = 25$) |
| 4%, 34%, 23%, 39%, 0% | 40.26% | 40.60% | 86.12% |
| 6%, 29%, 25%, 36%, 4% | 40.06% | 40.94% | 87.61% |
| 6%, 26%, 21%, 40%, 7% | 39.89% | 40.85% | 88.19% |

stations to be selected with a minimum probability $m < \frac{100\%}{|TX|}$ ensures that we can further decrease the achievable accuracy of even an advanced attacker.

The three best performing station probability values found through TPE are shown in Table 28 along with the accuracy achieved during TPE optimization. We can see that TX-D is given the highest probability values. This is a reasonable choice considering that TX-D achieves the lowest accuracy (i.e., in Fig. 41) when evaluated alone. When TPE is used to optimize the per-station probability values, only a single training repetition is performed. As such, it is possible that the accuracy achieved is artificially low. To ensure that the accuracy values found through TPE are legitimate, we repeat the experiment with the same per-station probability hyperparameter values over $N = 25$ repetitions and calculate the average and the difference from the TPE accuracy. We can see that for most of the best-selected per-station probabilities, the TPE accuracy and the average after 25 repetitions is within 1%. This demonstrates that the per-station probabilities selected by TPE are generalizable and not due to random chance. From this, we can observe that by using unique selection probabilities for each TX allows us to reduce the expected accuracy of the eavesdropper from 56.58% (i.e., non-periodic in Table 27) down to approximately 40% accuracy.

Now that we have demonstrated that these per-station probability values can successfully decrease the accuracy of the advanced attacker, next we look at how these random station probabilities affect any legitimate WiFi sensing RX device. A legitimate RX knows the exact random schedule of the TXs while the eavesdropper does not and as such, our allowed TX can actually leverage the CSI coming from more than one TX when making predictions. In Table 28, we identified the station probabilities which achieve lowest accuracy for Eve through TPE. Using these same station probabilities, we trained an allowed RX model by replacing eq. (7.4) with eq. (7.2). By doing this, we encode some additional structure in the CSI tensor without including any additional CSI amplitude data. Through this, we find that all station probabilities achieve between 86.12% and 88.19% accuracy for the allowed RX. In fact, these accuracy values are similar and even greater than the best single TX in Fig. 41 (i.e., TX-A with 84.6% accuracy). As such, we can say that while applying the pseudo-random schedule reduces the effectiveness of disallowed eavesdropper devices in performing sensing, the same system does not affect and may even improve the performance of allowed sensing device. Note that these accuracy values for Eve are based on the assumption that the eavesdropper can obtain training CSI data from the TX devices in the environment, which could be challenging. Any missing information during such training process (e.g., wrong labels, missing CSI from some time frames or from some TXs temporarily) will potentially lower the accuracy even further.

## 7.6 Discussion

### 7.6.1 Effect on Communication

WiFi sensing combines RF sensing into preexisting pervasive communications systems (i.e., WiFi). As such, it is important that a scheme which decreases the sensing ability of a system does not also decrease the communication ability of the system. For example, a signal jammer may be an efficient method for adding random signal noise into WiFi sensing measurements, however it also hinders the ability for legitimate WiFi devices to communicate while jamming is in progress. Our proposed method achieves the following regarding both sensing and communication:

1. Sensing is still possible and even improved for legitimate RXs through the use of multiple TXs.

2. Sensing is falsified and obscured for illegitimate eavesdropper RXs.

3. Communication packets are captured like normal for legitimate RXs.

4. Communication packets are captured like normal for eavesdropper RXs.

Notice, that our method does not worry about the content of the communication and even allows both legitimate and eavesdropper RXs to still capture the packet data. If the data in the packets must be hidden from eavesdroppers, then the data can easily be encrypted before transmission, however this is unrelated to the privacy concerns discussed in this chapter.

### 7.6.2 Generalizability to New Environments

In this chapter, we demonstrated that we can confuse an eavesdropper device by transmitting over multiple TX antennas following a pseudo-random schedule rather than transmitting over just a single TX antenna. Due to the placement of these TX antennas and the physical locations of the activities being sensed, we showed that different TX antennas are better for recognizing different sets of activities. As such, the best per-station probabilities selected in this experimental environment will not necessarily be applicable to new environments, which may also have more or fewer TX antennas in the setup. As such, the proposed system is structured such that:

1. Per-station probabilities are learned through the TPE using real-world CSI data collected in the environment. Thus, the probability values can be selected automatically for each new environment.

2. The allowed RX (i.e., eq. (7.2)) and disallowed RX (i.e., eq. (7.4)) are designed as differentiable functions which allows for a machine learning model-based optimization of station probabilities. Thus, more complex station scheduling can be performed in new environments.

Furthermore, towards machine learning model-based station scheduling, we showed in Chapter 4 that even low level WiFi sensing devices such as the ESP32 used in this study can leverage machine learning models directly on-board. This means that such a system is possible even with low cost equipment, thus improving the scalability of such a system.

### 7.6.3 Future Work

In this chapter, we evaluated the effect of five TX antennas that are positioned at a constant distance apart, however different distributions of TX antennas will have different effects within each unique environment. As such, more work can be done in understanding how different TX antenna positions and different number of antennas affect the proposed system. Furthermore, because each environment has unique activities to be obfuscated, it may be possible to automatically determine optimal placement of these TX antennas through metrics such as the sensing-signal-to-noise-ratio (SSNR) [234] or through wireless sensing signal simulators [235].

In our experiments, the eavesdropper uses CSI exclusively to recognize and localize the activities performed. However, metrics such as the received signal strength indicator (RSSI), angle of arrival (AoA), or other signal metrics may also be available for the eavesdropper and may reveal the physical locations of the TXs [236]. While this chapter focuses directly on obfuscating physical activities, additional work into obfuscating antenna locations (e.g., [237]) will directly benefit our proposed system.

## 7.7 Chapter Contributions and Summary

In this chapter, we proposed a defense mechanism against adversarial WiFi sensing through the use of multiple spatially distributed TX antennas connected to the same source device. These antennas are utilized to transmit data based on a pseudo-random schedule which is known to legitimate RX devices but hidden from eavesdroppers. Legitimate RX devices can filter the received data per TX device based on the schedule used and use a specific ML model for predictions, while the eavesdropper uses the CSI data from all TX devices and uses them as input into its prediction model. Through various experiments, we showed that accuracy of the eavesdropper model is much lower than the accuracy of the legitimate RX model thanks to the obfuscation generated through the spatially distributed TX antennas. The accuracy of the eavesdropper also reduces as the number of TX devices increases. Additionally, we demonstrated that setting a per-station probability for our pseudo-random scheduler allows for a further decrease in the accuracy of an eavesdropper. We proposed a Tree-structured Parzen Estimator (TPE) approach to identify optimal per-station probability values which ensure that the system can be automatically adaptable in new environments. Finally, we also showed that accuracy for legitimate WiFi sensing RX devices can even be improved through the use of CSI from multiple TXs. As such, the proposed system is able to allow legitimate sensing to occur while reducing the feasibility of illegitimate eavesdropper-based sensing from occurring.

# CHAPTER 8

# CONCLUDING REMARKS

In this dissertation, we considered the use of WiFi sensing at the edge, where wireless WiFi signals captured throughout our everyday environments can be used to sense and track physical attributes of the environment such as the human movements and activities. This work emphasized the use of edge based microcontrollers (i.e., the ESP32 MCU) which allows WiFi sensing from much smaller devices than were possible using previously available tools. Thus, throughout the research work discussed in this dissertation, we moved forward towards more realistic and scalable WiFi sensing systems.

The novelty of WiFi sensing on edge devices produces new unseen challenges which are identified, anticipated, and solved through the course of this dissertation work. Namely, throughout this work, we discuss the introduction of a tool developed to ease the use and understanding of WiFi sensing on edge devices. Furthermore, we evaluate signal preprocessing steps as well as techniques for achieving machine learning directly on-board very low-resource edge devices. From here, we look forward towards the ability to allow for machine learning personalization on-board edge devices and demonstrate methods like federated learning which can be leveraged to increase the accuracy of WiFi sensing models in a collaborative model, thus allowing for a reduced reliance on central computation resources as well as reducing the amount of training data required for deploying new WiFi sensing systems into real-world environments. Beyond the edge, the work in this dissertation anticipates the broader impacts of the use of WiFi sensing to identify negative aspects such as adversarial surveillance. Finally, to defend against adversarial eavesdropping WiFi sensing nodes, we proposed an anti-eavesdropper method which prevents accurate sensing for eavesdropper devices while still allowing and even improving the sensing accuracy of approved WiFi sensing nodes in our system.

Overall, the goal of this dissertation is to further improve understanding of WiFi sensing and to introduce and emphasize the need for designing learning algorithms which will be reasonably deployed in edge scenarios. Pushing towards these ideals will ensure the feasibility of WiFi sensing as a viable sensing modality as well as allow for better user privacy by retaining personally identifiable sensing and recognition at the edge rather than at central computation and storage data centers.

## 8.1 Contributions

Throughout this dissertation, we made the following contributions:

1. Designed an ESP32-CSI-Tool which enables CSI collection from low-powered ESP32-MCU edge devices. By collecting CSI, this allows WiFi sensing to be performed on much lower-cost edge devices, thus allowing for real-world and scalable systems compared to previous CSI collection tools.

2. Produced a thorough survey of WiFi sensing research literature to identify signal processing techniques and machine learning techniques typically used for WiFi sensing. We then evaluate these techniques for:

   (a) Feasibility when performed on low-power edge devices like the ESP32-MCU.

   (b) Accuracy across different WiFi sensing tasks including small-scale hand gesture recognition, medium-scale human activity recognition, and large-scale human activity and localization.

(c) Other system properties such as computation time on-board an ESP32-MCU, inference rates for signal processing techniques and machine learning, and energy consumption.

3. Proposed the *WiFederated* framework, which introduces federated learning for the first time for WiFi sensing tasks. WiFederated allows multiple edge devices to collaboratively train machine learning models by sharing knowledge in the form of model weights rather than sharing raw CSI data. Through this, we can ensure raw data remains private, and also reduces the amount of communication required.

4. Proposed client selection methods for WiFederated which selects subsets of candidate devices to collaboratively train the federated model. Through this method, we can increase the accuracy of the model by only allowing quality devices to update the federated model.

5. Introduced the concept of *continuous annotation* which can leverage external sensors to provide accurate annotations for training federated models.

6. Demonstrated one adversarial use of WiFi sensing where an adversary can perform crowdcounting and human direction tracking through-wall in NLOS scenarios.

7. Proposed an anti-eavesdropper defense mechanism which prevents eavesdropper WiFi sensing devices from performing surveillance while still allowing and even improving the accuracy of allowed devices to perform WiFi sensing in a target area.

## 8.2 Future Work

Throughout this work, emphasis is placed on the use of WiFi signals as the source for achieving wireless sensing due to the high availability of WiFi signals continuously throughout our daily lives. However, many of the opportunities identified in this dissertation may similarly be applied to other wireless sensing modalities such as UWB which is increasingly present in areas such as smartphones [238] and automobiles [239], or mmWave which has a similar interest from smartphone manufacturers [4].

As edge devices continue to improve in computation ability and as energy requirements decrease, efforts towards achieving higher complexity edge machine learning will increase. This offers a number of important opportunities to not only push wireless sensing techniques further but also move other areas forward such as improved voice assistants [240], improved health and wellness tracking [241], and more. To allow these areas to prosper, further work is needed in improving model training efficiency, decreasing the amount of data required compared to current state-of-the-art data-intensive deep learning models, and finally, allowing for models to be trained and personalized per user or per location to allow for better specialized models rather than focusing on generalized models which fail for important everyday tasks.

# REFERENCES

[1] Youwei Zeng et al. “FarSense: Pushing the Range Limit of WiFi-based Respiration Sensing with CSI Ratio of Two Antennas”. In: *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* (2019).

[2] Khe Chai Sim, Petr Zadrazil, and Françoise Beaufays. “An Investigation Into On-device Personalization of End-to-end Automatic Speech Recognition Models”. In: *arXiv preprint arXiv:1909.06678* (2019).

[3] Andrew Hard et al. “Federated Learning for Mobile Keyboard Prediction”. In: *arXiv preprint arXiv:1811.03604* (2018).

[4] Google LLC. *Technology: Soli*. URL: https://atap.google.com/soli/technology/.

[5] *Linksys Aware Homepage*. URL: https://www.linksys.com/us/linksys-aware/.

[6] *Emerald Innovations Homepage*. URL: https://emeraldinno.com/.

[7] *Origin Wireless Homepage*. URL: https://originwirelessai.com/.

[8] Google LLC. *Google Cloud Edge TPU*. URL: https://cloud.google.com/edge-tpu.

[9] Canaan Inc. *Kendryte K210*. URL: https://canaan.io/product/kendryteai.

[10] Sipeed. *Maixduino Documentation*. URL: https://maixduino.sipeed.com/en/.

[11] Peizheng Li et al. “Deep Transfer Learning for WiFi Localization”. In: *Proceedings of the Radar Conference*. 2021.

[12] Rui Zhou et al. “Adaptive Device-Free Localization in Dynamic Environments Through Adaptive Neural Networks”. In: *Sensors Journal* (2020).

[13] Sheng Tan and Jie Yang. “Multi-User Activity Recognition and Tracking Using Commodity WiFi”. In: *arXiv preprint arXiv:2106.00865* (2021).

[14] Cong Shi et al. “WiFi-Enabled User Authentication through Deep Learning in Daily Activities”. In: *Transactions on Internet of Things* (2021).

[15] Yanchao Zhao et al. “Device-Free Secure Interaction with Hand Gestures in WiFi-enabled IoT Environment”. In: *Internet of Things Journal* (2020).

[16] Qirong Bu et al. “TransferSense: towards environment independent and one-shot wifi sensing”. In: *Personal and Ubiquitous Computing* (2021).

[17] Itsuki Shirakami and Takashi Sato. “Heart Rate Variability Extraction using Commodity Wi-Fi Devices via Time Domain Signal Processing”. In: *EMBS International Conference on Biomedical and Health Informatics.* IEEE. 2021.

[18] Wei Liu et al. “Wi-PSG: Detecting Rhythmic Movement Disorder Using COTS WiFi”. In: *Internet of Things Journal* (2020).

[19] Xi Chen et al. “PHCount: Passive Human Number Counting Using WiFi”. In: *International Conference in Communications, Signal Processing, and Systems.* Springer. 2020.

[20] Roshan Sandaruwan et al. “Device-free Pedestrian Count Estimation Using Wi-Fi Channel State Information”. In: *International Intelligent Transportation Systems Conference (ITSC).* IEEE, Sept. 2021.

[21] Feng Zhang et al. “WiDetect: Robust Motion Detection with a Statistical Electromagnetic Model”. In: *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* (2019).

[22] Fu Xiao et al. "Invisible Cloak Fails: CSI-based Passive Human Detection". In: *Proceedings of the Workshop on Context Sensing and Activity Recognition*. ACM, Nov. 2015.

[23] Steven M. Hernandez and Eyuphan Bulut. "TrinaryMC: Monte Carlo based anchorless relative positioning for indoor positioning". In: *Proceedings of the Consumer Communications & Networking Conference (CCNC)*. 2020.

[24] Steven M. Hernandez and Eyuphan Bulut. "Using perceived direction information for anchorless relative indoor localization". In: *Journal of Network and Computer Applications* (2020).

[25] Ninad Jadhav et al. "WSR: A WiFi Sensor for Collaborative Robotics". In: *arXiv preprint arXiv:2012.04174* (2020).

[26] Bohong Xiang et al. "UAV Assisted Localization Scheme of WSNs Using RSSI and CSI Information". In: *Proceedings of the International Conference on Computer and Communications*. IEEE. 2020.

[27] Jian-guo Jiang et al. "CS-Dict: Accurate Indoor Localization with CSI Selective Amplitude and Phase Based Regularized Dictionary Learning". In: *International Conference on Algorithms and Architectures for Parallel Processing*. Springer. 2020.

[28] Rui Zhou et al. "Device-free Localization Based on CSI Fingerprints and Deep Neural Networks". In: *Proceedings of the International Conference on Sensing, Communication, and Networking*. IEEE. 2018.

[29] Liuyi Yang, Tomio Kamada, and Chikara Ohta. "Indoor localization based on CSI in dynamic environments through domain adaptation". In: *Communications Express* (2021).

[30] Zhang Yong, Wu Cheng Bin, and Yang Chen. “A Low-overhead Indoor Positioning System Using CSI Fingerprint Based on Transfer Learning”. In: *Sensors Journal* (2021).

[31] Neena Damodaran et al. “Device free human activity and fall recognition using WiFi channel state information (CSI)”. In: *CCF Transactions on Pervasive Computing and Interaction* (2020).

[32] Jinyang Huang et al. “Towards Anti-interference WiFi-based Activity Recognition System Using Interference-Independent Phase Component”. In: *Conference on Computer Communications*. IEEE. 2020.

[33] Jianfei Yang et al. “Device-Free Occupant Activity Sensing Using WiFi-Enabled IoT Devices for Smart Homes”. In: *Internet of Things Journal* (2018).

[34] Md Tamzeed Islam and Shahriar Nirjon. “Wi-Fringe: Leveraging Text Semantics in WiFi CSI-Based Device-Free Named Gesture Recognition”. In: *Proceedings of the International Conference on Distributed Computing in Sensor Systems*. IEEE. 2020.

[35] Daqing Zhang et al. “Anti-Fall: A Non-intrusive and Real-Time Fall Detector Leveraging CSI from Commodity WiFi Devices”. In: *International Conference on Smart Homes and Health Telematics*. Springer. 2015.

[36] Yichao Zhou et al. “Human fall recognition based on WiFi CSI with dynamic subcarrier extraction of interference index”. In: *Journal of Physics: Conference Series*. IOP Publishing. 2021.

[37] Steven M. Hernandez et al. “Wi-PT: Wireless Sensing based Low-cost Physical Rehabilitation Tracking”. In: *Proceedings of IEEE International Conference on E-health Networking, Application & Services (HealthCom)*. 2022.

[38] Hong Cai et al. “Teaching RF to Sense without RF Training Measurements”. In: *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* (2020).

[39] Jinyang Huang et al. “WiLay: A Two-Layer Human Localization and Activity Recognition System Using WiFi”. In: *Proceedings of the Vehicular Technology Conference*. IEEE. 2021.

[40] Sheng Tan and Jie Yang. “WiFinger: Leveraging Commodity WiFi for Fine-grained Finger Gesture Recognition”. In: *Proceedings of the ACM International Symposium on Mobile Ad Hoc Networking and Computing*. 2016.

[41] Muneeba Raja, Viviane Ghaderi, and Stephan Sigg. “WiBot! In-Vehicle Behaviour and Gesture Recognition Using Wireless Network Edge”. In: *Proceedings of the International Conference on Distributed Computing Systems*. IEEE. 2018.

[42] Hong Li et al. “WiFinger: Talk to Your Smart Devices with Finger-grained Gesture”. In: *Proceedings of the International Joint Conference on Pervasive and Ubiquitous Computing*. ACM, 2016.

[43] Yong Zhang, Kangle Xu, and Yujie Wang. “WiNum: A WIFI Finger Gesture Recognition System Based on CSI”. In: *Proceedings of the International Conference on Information Technology: IoT and Smart City*. 2019.

[44] Zhanjun Hao et al. “Wi-SL: Contactless Fine-Grained Gesture Recognition Uses Channel State Information”. In: *Sensors* (2020).

[45] Chi Lin et al. “WiWrite: An Accurate Device-Free Handwriting Recognition System with COTS WiFi”. In: *Proceedings of the International Conference on Distributed Computing Systems*. IEEE. 2020.

[46] Ruiyang Gao et al. "Towards Position-Independent Sensing for Gesture Recognition with Wi-Fi". In: *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* (2021).

[47] Kamran Ali et al. "Keystroke Recognition Using WiFi Signals". In: *Proceedings of the International Conference on Mobile Computing and Networking.* 2015.

[48] Xingfa Shen et al. "WiPass: 1D-CNN-based smartphone keystroke recognition Using WiFi signals". In: *Pervasive and Mobile Computing* (2021).

[49] Jialai Liu et al. "An Efficient CSI-Based Pedestrian Monitoring Approach via Single Pair of WiFi Transceivers". In: *International Conference on Neural Computing for Advanced Applications.* Springer. 2021.

[50] Xingang Wang, Yufei Wang, and Dong Wang. "A Real-time CSI-based Passive Intrusion Detection Method". In: *International Conference on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking.* IEEE. 2020.

[51] Jikun Guo and Huihui Li. "RSWI: a rescue system with WiFi sensing and image recognition". In: *Proceedings of the Turing Celebration Conference-China.* ACM, 2019.

[52] Belal Korany and Yasamin Mostofi. "Counting a Stationary Crowd Using Off-the-Shelf WiFi". In: *Proceedings of the International Conference on Mobile Systems, Applications, and Services.* 2021.

[53] Daniel Konings and Fakhrul Alam. “LifeCount: A Device-free CSI-based Human Counting Solution for Emergency Building Evacuations”. In: *Sensors Applications Symposium*. IEEE. 2020.

[54] Steven M. Hernandez and Eyuphan Bulut. “Adversarial Occupancy Monitoring using One-Sided Through-Wall WiFi Sensing”. In: *International Conference on Communications*. IEEE. 2021.

[55] Youwei Zeng et al. “FullBreathe: Full Human Respiration Detection Exploiting Complementarity of CSI Phase and Amplitude of WiFi Signals”. In: *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* (2018).

[56] Youwei Zeng et al. “MultiSense: Enabling Multi-person Respiration Sensing with Commodity WiFi”. In: *Proceedings of ACM Interactive Mobile Wearable Ubiquitous Technologies* (2020).

[57] Jinyi Liu et al. “WiPhone: Smartphone-based Respiration Monitoring Using Ambient Reflected WiFi Signals”. In: *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* (2021).

[58] Belal Korany and Yasamin Mostofi. “Nocturnal Seizure Detection Using Off-the-Shelf WiFi”. In: *arXiv preprint arXiv:2103.13556* (2021).

[59] Xuyu Wang, Chao Yang, and Shiwen Mao. “PhaseBeat: Exploiting CSI Phase Data for Vital Sign Monitoring with Commodity WiFi Devices”. In: *Proceedings of the International Conference on Distributed Computing Systems*. IEEE. 2017.

[60] Yu Gu et al. “EmoSense: Data-Driven Emotion Sensing via Off-the-Shelf WiFi Devices”. In: *International Conference on Communications (ICC)*. IEEE. 2018.

[61] Zhenzhe Lin et al. “WiEat: Fine-grained Device-free Eating Monitoring Leveraging Wi-Fi Signals”. In: *Proceedings of the International Conference on Computer Communications and Networks*. IEEE. 2020.

[62] Sheng Tan and Jie Yang. “Object Sensing for Fruit Ripeness Detection Using WiFi Signals”. In: *arXiv preprint arXiv:2106.00860* (2021).

[63] Weidong Yang et al. “Wi-Wheat: Contact-Free Wheat Moisture Detection with Commodity WiFi”. In: *International Conference on Communications*. IEEE. 2018.

[64] Steven M. Hernandez, Deniz Erdag, and Eyuphan Bulut. “Towards Dense and Scalable Soil Sensing Through Low-Cost WiFi Sensing Networks”. In: *Proceedings of the Conference on Local Computer Networks (LCN)*. IEEE. 2021.

[65] Yili Ren et al. “Liquid Level Sensing Using Commodity WiFi in a Smart Home Environment”. In: *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* (2020).

[66] Hang Song et al. “WiEps: Measurement of Dielectric Property With Commodity WiFi Device—An Application to Ethanol/Water Mixture”. In: *Internet of Things Journal* (2020).

[67] Sirui Jian, Shigemi Ishida, and Yutaka Arakawa. “Initial Attempt on Wi-Fi CSI Based Vibration Sensing for Factory Equipment Fault Detection”.

In: *Adjunct Proceedings of the 2021 International Conference on Distributed Computing and Networking*. 2021.

[68] Feng Zhang et al. “WiSpeed: A Statistical Electromagnetic Approach for Device-Free Indoor Speed Estimation”. In: *Internet of Things Journal* (2018).

[69] Ramjee Prasad. *OFDM for Wireless Communications Systems*. Artech House, 2004.

[70] Yushi Shen and Ed Martinez. “Channel Estimation in OFDM Systems”. In: *Freescale semiconductor application note* (2006).

[71] Kamran Ali et al. “On Goodness of WiFi based Monitoring of Vital Signs in the Wild”. In: *arXiv preprint arXiv:2003.09386* (2020).

[72] Zhe Chen et al. “RF-Based Human Activity Recognition Using Signal Adapted Convolutional Neural Network”. In: *IEEE Transactions on Mobile Computing* (2021).

[73] Marwan Yusuf et al. “Human Sensing in Reverberant Environments: RF-Based Occupancy and Fall Detection in Ships”. In: *IEEE Transactions on Vehicular Technology* (May 2021).

[74] Mohammud J. Bocus, Kevin Chetty, and Robert J. Piechocki. “UWB and WiFi Systems as Passive Opportunistic Activity Sensing Radars”. In: *Proceedings of the Radar Conference*. IEEE, May 2021.

[75] Jaime Lien et al. “Soli: Ubiquitous Gesture Sensing with Millimeter Wave Radar”. In: *ACM Transactions on Graphics (TOG)* (2016).

[76] Ozturk, Muhammed Zahid and Wu, Chenshu and Wang, Beibei and Liu, KJ. “RadioMic: Sound Sensing via mmWave Signals”. In: *arXiv preprint arXiv:2108.03164* (2021).

[77] Jie Wang et al. “Device-Free Human Gesture Recognition With Generative Adversarial Networks”. In: *IEEE Internet of Things Journal* (Aug. 2020).

[78] Stephan Sigg, Ulf Blanke, and Gerhard Troster. “The telepathic phone: Frictionless activity recognition from WiFi-RSSI”. In: *International Conference on Pervasive Computing and Communications (PerCom)*. IEEE, Mar. 2014.

[79] Saandeep Depatla and Yasamin Mostofi. “Passive Crowd Speed Estimation and Head Counting Using WiFi”. In: *International Conference on Sensing, Communication, and Networking (SECON)*. IEEE, June 2018.

[80] Taiyu Zhu et al. “IoMT-Enabled Real-time Blood Glucose Prediction with Deep Learning and Edge Computing”. In: *Internet of Things Journal* (2022).

[81] A. Navaas Roshan et al. “Adaptive Traffic Control With TinyML”. In: *International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET)*. IEEE, Mar. 2021.

[82] Wamiq Raza et al. “Energy-Efficient Inference on the Edge Exploiting TinyML Capabilities for UAVs”. In: *Drones* (Oct. 2021).

[83] Haoyu Ren, Darko Anicic, and Thomas A Runkler. “TinyOL: TinyML with Online-Learning on Microcontrollers”. In: *International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2021.

[84] Kavya Kopparapu and Eric Lin. “TinyFedTL: Federated Transfer Learning on Tiny Devices”. In: *arXiv preprint arXiv:2110.01107* (2021).

[85] Han Cai et al. “TinyTL: Reduce Activations, Not Trainable Parameters for Efficient On-Device Learning”. In: *arXiv preprint arXiv:2007.11622* (2020).

[86] Ju Wang et al. “LiFS: low human-effort, device-free localization with fine-grained subcarrier information”. In: *Proceedings of the International Conference on Mobile Computing and Networking*. ACM. 2016.

[87] Yanzi Zhu et al. “Adversarial WiFi Sensing”. In: *arXiv preprint arXiv:1810.10109* (2018).

[88] Claude Elwood Shannon. “Communication in the Presence of Noise”. In: *Proceedings of the IRE* (1949).

[89] Jiang Xiao, Huichuwu Li, and Hai Jin. “Transtrack: Online meta-transfer learning and Otsu segmentation enabled wireless gesture tracking”. In: *Pattern Recognition* (2022).

[90] Gongzhui Zhang et al. “Gait Cycle Detection Using Commercial WiFi Device”. In: *International Conference in Communications, Signal Processing, and Systems*. Springer. 2020.

[91] Hua Kang, Qian Zhang, and Qianyi Huang. “Context-Aware Wireless Based Cross Domain Gesture Recognition”. In: *Internet of Things Journal* (2021).

[92] Lingchao Guo et al. “Emergency Semantic Feature Vector Extraction From WiFi Signals for In-Home Monitoring of Elderly”. In: *Journal of Selected Topics in Signal Processing* (2021).

[93] Hongbo Jiang et al. “Smart home based on WiFi sensing: A survey”. In: *IEEE Access* (2018).

[94] Yongsen Ma, Gang Zhou, and Shuangquan Wang. “WiFi Sensing with Channel State Information: A Survey”. In: *Computing Surveys* (2019).

[95] Jian Liu et al. “Wireless Sensing for Human Activity: A Survey”. In: *Communications Surveys & Tutorials* (2019).

[96] Ying He et al. “WiFi Vision: Sensing, Recognition, and Detection With Commodity MIMO-OFDM WiFi”. In: *Internet of Things Journal* (2020).

[97] Jiao Liu, Guanlong Teng, and Feng Hong. “Human Activity Sensing with Wireless Signals: A Survey”. In: *Sensors* (2020).

[98] Chenning Li, Zhichao Cao, and Yunhao Liu. “Deep AI Enabled Ubiquitous Wireless Sensing: A Survey”. In: *Computing Surveys* (2021).

[99] Isura Nirmal et al. “Deep Learning for Radio-based Human Sensing: Recent Advances and Future Directions”. In: *Communications Surveys & Tutorials* (2021).

[100] Daniel Halperin et al. “Tool Release: Gathering 802.11n Traces with Channel State Information”. In: *ACM SIGCOMM CCR* (Jan. 2011).

[101] Yaxiong Xie, Zhenjiang Li, and Mo Li. “Precise Power Delay Profiling with Commodity WiFi”. In: *Proceedings of the International Conference on Mobile Computing and Networking*. MobiCom ’15. Paris, France: ACM, 2015.

[102] Steven M. Hernandez and Eyuphan Bulut. “Lightweight and Standalone IoT based WiFi Sensing for Active Repositioning and Mobility”. In: *International Symposium on “A World of Wireless, Mobile and Multimedia Networks” (WoWMoM)*. Cork, Ireland, June 2020.

[103] Zimu Zhou et al. “LiFi: Line-Of-Sight Identification with WiFi”. In: *Conference on Computer Communications*. IEEE. 2014.

[104] Ramin Ramezani, Yubin Xiao, and Arash Naeim. “Sensing-Fi: Wi-Fi CSI and Accelerometer Fusion System for Fall Detection”. In: *EMBS International Conference on Biomedical & Health Informatics*. IEEE. 2018.

[105] Xuyu Wang, Chao Yang, and Shiwen Mao. "On CSI-Based Vital Sign Monitoring Using Commodity WiFi". In: *ACM Transactions on Computing for Healthcare* (2020).

[106] Enjie Ding et al. "A Robust Passive Intrusion Detection System with Commodity WiFi Devices". In: *Journal of Sensors* (2018).

[107] Lingyan Zhang and Hongyu Wang. "3D-WiFi: 3D Localization With Commodity WiFi". In: *IEEE Sensors Journal* (July 2019).

[108] Xue Ding et al. "A New Method of Human Gesture Recognition Using Wi-Fi Signals Based on XGBoost". In: *International Conference on Communications in China*. IEEE. 2020.

[109] Yi Tian Xu et al. "PresSense: Passive Respiration Sensing via Ambient WiFi Signals in Noisy Environments". In: *International Conference on Intelligent Robots and Systems*. IEEE. 2020.

[110] Jiancun Zuo et al. "A New Method of Posture Recognition Based on WiFi Signal". In: *Communications Letters* (2021).

[111] Qizhen Zhou, Jianchun Xing, and Qiliang Yang. "Device-free occupant activity recognition in smart offices using intrinsic Wi-Fi components". In: *Building and Environment* (2020).

[112] Lingchao Guo et al. "When Healthcare Meets Off-the-Shelf WiFi: A Non-Wearable and Low-Costs Approach for In-Home Monitoring". In: *arXiv preprint arXiv:2009.09715* (2020).

[113] Yanling Hao, Zhiyuan Shi, and Yuanwei Liu. "A Wireless-Vision Dataset for Privacy Preserving Human Activity Recognition". In: *International Conference on Multimedia Computing, Networking and Applications*. IEEE. 2020.

[114] Zhenguo Shi et al. “Towards Environment-independent Human Activity Recognition using Deep Learning and Enhanced CSI”. In: *Global Communications Conference*. IEEE. 2020.

[115] Dan Wu et al. “WiDir: Walking Direction Estimation Using Wireless Signals”. In: *Proceedings of the International Joint Conference on Pervasive and Ubiquitous Computing*. UbiComp ’16. Heidelberg, Germany: ACM, 2016.

[116] Mengyuan Li et al. “When CSI Meets Public WiFi: Inferring Your Mobile Phone Password via WiFi Signals”. In: *Proceedings of the SIGSAC Conference on Computer and Communications Security*. 2016.

[117] Lokesh Sharma et al. “High Accuracy WiFi-Based Human Activity Classification System with Time-Frequency Diagram CNN Method for Different Places”. In: *Sensors* (2021).

[118] Heba Abdelnasser, Moustafa Youssef, and Khaled A Harras. “WiGest: A Ubiquitous WiFi-based Gesture Recognition System”. In: *Conference on Computer Communications*. IEEE. 2015.

[119] Wei Xi et al. “Device-free Human Activity Recognition using CSI”. In: *Proceedings of the Workshop on Context Sensing and Activity Recognition*. 2015.

[120] Xuefeng Liu et al. “Wi-Sleep: Contactless Sleep Monitoring via WiFi Signals”. In: *Real-Time Systems Symposium*. IEEE. 2014.

[121] Dehao Jiang, Mingqi Li, and Chunling Xu. “WiGAN: A WiFi Based Gesture Recognition System with GANs”. In: *Sensors* (2020).

[122] Jinyang Huang et al. “WiAnti: an Anti-Interference Activity Recognition System Based on WiFi CSI”. In: *International Conference on Internet of Things,*

*Green Computing and Communications, Cyber, Physical and Social Computing, and Smart Data*. IEEE. 2018.

[123] Muhammad Muaaz et al. "Wi-Sense: A passive human activity recognition system using Wi-Fi and convolutional neural network and its integration in health information systems". In: *Annals of Telecommunications* (2021).

[124] Wei Nie et al. "UAV Detection and Identification Based on WiFi Signal and RF Fingerprint". In: *Sensors Journal* (2021).

[125] Yunhao Bai and Xiaorui Wang. "CARIN: Wireless CSI-based Driver Activity Recognition under the Interference of Passengers". In: *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* (2020).

[126] Xiangyu Xu et al. "BreathListener: Fine-grained Breathing Monitoring in Driving Environments Utilizing Acoustic Signals". In: *Proceedings of the International Conference on Mobile Systems, Applications, and Services*. 2019.

[127] Stéphane Mallat. *A Wavelet Tour of Signal Processing*. Third. Elsevier, 2009.

[128] Mohan Vishwanath. "The Recursive Pyramid Algorithm for the Discrete Wavelet Transform". In: *Transactions on Signal Processing* (1994).

[129] Abraham Savitzky and Marcel JE Golay. "Smoothing and Differentiation of Data by Simplified Least Squares Procedures." In: *Analytical chemistry* (1964).

[130] Abdelwahed Khamis et al. "WiRelax: Towards real-time respiratory biofeedback during meditation using WiFi". In: *Ad Hoc Networks* (2020).

[131] Julius Orion Smith. *Introduction to Digital Filters: With Audio Applications*. Julius Smith, 2007.

[132] Sheng Tan, Jie Yang, and Yingying Chen. “Enabling Fine-grained Finger Gesture Recognition on Commodity WiFi Devices”. In: *Transactions on Mobile Computing* (2020).

[133] Xiang Zhang et al. “Wital: A COTS WiFi Devices Based Vital Signs Monitoring System Using NLOS Sensing Model”. In: *Transactions on Human-Machine Systems* (2023).

[134] Wei Zhuang et al. “Develop an Adaptive Real-Time Indoor Intrusion Detection System Based on Empirical Analysis of OFDM Subcarriers”. In: *Sensors* (2021).

[135] Wei Wang et al. “Understanding and Modeling of WiFi Signal Based Human Activity Recognition”. In: *Proceedings of the International Conference on Mobile Computing and Networking*. ACM. 2015.

[136] Jie Zhang et al. “CrossSense: Towards Cross-Site and Large-Scale WiFi Sensing”. In: *Proceedings of the International Conference on Mobile Computing and Networking*. ACM. 2018.

[137] Bohan Yu et al. “WiFi-Sleep: Sleep Stage Monitoring Using Commodity Wi-Fi Devices”. In: *Internet of Things Journal* (2021).

[138] Chenshu Wu et al. “GaitWay: Monitoring and Recognizing Gait Speed Through the Walls”. In: *Transactions on Mobile Computing* (2020).

[139] Yanan Li et al. “Location-Free CSI Based Activity Recognition With Angle Difference of Arrival”. In: *Wireless Communications and Networking Conference*. IEEE. 2020.

[140] Dan Wu et al. “FingerDraw: Sub-wavelength Level Finger Motion Tracking with WiFi Signals”. In: *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* (2020).

[141] Tao Wang et al. “Wi-Alarm: Low-cost passive intrusion detection using WiFi”. In: *Sensors* (2019).

[142] Wenjun Jiang et al. “Towards 3D Human Pose Construction Using WiFi”. In: *Proceedings of the International Conference on Mobile Computing and Networking*. 2020.

[143] Rui Xiao et al. “OneFi: One-Shot Recognition for Unseen Gesture via COTS WiFi”. In: *Proceedings of the ACM Conference on Embedded Networked Sensor Systems*. 2021.

[144] Yong Lu, Shaohe Lv, and Xiaodong Wang. “Towards Location Independent Gesture Recognition with Commodity WiFi Devices”. In: *Electronics* (2019).

[145] Mohammed Abdulaziz Aide Al-qaness and Fangmin Li. “WiGeR: WiFi-Based Gesture Recognition System”. In: *ISPRS International Journal of Geo-Information* (2016).

[146] Tao Li et al. “A Novel Gesture Recognition System Based on CSI Extracted from a Smartphone with Nexmon Firmware”. In: *Sensors* (2021).

[147] Fang-Yu Chu et al. “WiFi CSI-Based Device-free Multi-room Presence Detection using Conditional Recurrent Network”. In: *Proceedings of the Vehicular Technology Conference*. IEEE. 2021.

[148] Yongsen Ma et al. “Location-and Person-Independent Activity Recognition with WiFi, Deep Neural Networks, and Reinforcement Learning”. In: *Transactions on Internet of Things* (2021).

[149] Xinbin Shen et al. “WiAgent: Link Selection for CSI-Based Activity Recognition in Densely Deployed Wi-Fi Environments”. In: *Wireless Communications and Networking Conference*. IEEE. 2021.

[150] Ye Sun et al. “Enabling Lightweight Device-Free Wireless Sensing with Network Pruning and Quantization”. In: *Sensors Journal* (2021).

[151] Rajalakshmi Nandakumar, Bryce Kellogg, and Shyamnath Gollakota. “Wi-Fi Gesture Recognition on Existing Devices”. In: *arXiv preprint arXiv:1411.5394* (2014).

[152] Jijun Zhao et al. “R-DEHM: CSI-Based Robust Duration Estimation of Human Motion with WiFi”. In: *Sensors* (2019).

[153] Zhenguo Shi et al. “WiFi-Based Activity Recognition using Activity Filter and Enhanced Correlation with Deep Learning”. In: *International Conference on Communications Workshops*. IEEE. 2020.

[154] Yan Wang et al. “E-eyes: Device-free Location-oriented Activity Identification Using Fine-grained WiFi Signatures”. In: *Proceedings of the International Conference on Mobile Computing and Networking*. 2014.

[155] Pierre-Emmanuel Novac et al. “Quantization and Deployment of Deep Neural Networks on Microcontrollers”. In: *Sensors* (2021).

[156] Forrest N Iandola et al. “SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and 0.5 MB model size”. In: *arXiv preprint arXiv:1602.07360* (2016).

[157] Andrew G Howard et al. “MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications”. In: *arXiv preprint arXiv:1704.04861* (2017).

[158] Mingxing Tan and Quoc Le. "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks". In: *International Conference on Machine Learning*. PMLR. 2019.

[159] Song Han et al. "Learning both Weights and Connections for Efficient Neural Networks". In: *arXiv preprint arXiv:1506.02626* (2015).

[160] Ze Yang et al. "Model compression with two-stage multi-teacher knowledge distillation for web question answering system". In: *Proceedings of the International Conference on Web Search and Data Mining*. 2020.

[161] Steven M. Hernandez et al. "Sharing Low Rank Conformer Weights for Tiny Always-On Ambient Speech Recognition Models". In: *Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. Rhodes, Greece, June 2023.

[162] Itay Hubara et al. "Binarized Neural Networks". In: *Advances in neural information processing systems* (2016).

[163] Zhuang Liu et al. "Rethinking the Value of Network Pruning". In: *arXiv preprint arXiv:1810.05270* (2018).

[164] Lennart Heim et al. "Measuring what Really Matters: Optimizing Neural Networks for TinyML". In: *arXiv preprint arXiv:2104.10645* (2021).

[165] Edgar Liberis, Łukasz Dudziak, and Nicholas D Lane. "$\mu$NAS: Constrained Neural Architecture Search for Microcontrollers". In: *Proceedings of the Workshop on Machine Learning and Systems*. 2021.

[166] Chenshu Wu and KJ Ray Liu. "Accurate Stride Length Estimation via Fused Radio and Inertial Sensing". In: *Proceedings of the World Forum on Internet of Things*. IEEE. 2020.

[167] Omotayo Oshiga et al. “Human Detection For Crowd Count Estimation Using CSI of WiFi Signals”. In: *International Conference on Electronics, Computer and Computation (ICECCO)*. IEEE, Dec. 2019.

[168] Alessandro Polo et al. “Real-Time CSI-Based Wireless Gesture Recognition for Human-Machine Interaction”. In: *International Conference on Modern Circuits and Systems Technologies (MOCAST)*. IEEE, July 2021.

[169] Shuo Li et al. “WiFi-based Device-free Vehicle Speed Measurement Using Fast Phase Correction MUSIC Algorithm”. In: *International Conferences on Internet of Things (iThings), Green Computing and Communications (GreenCom), Cyber, Physical and Social Computing (CPSCom), Smart Data (SmartData), and Congress on Cybermatics (Cybermatics)*. IEEE, Nov. 2020.

[170] Hua Xue et al. “Push the Limit of Multipath Profiling Using Commodity WiFi Devices With Limited Bandwidth”. In: *IEEE Transactions on Vehicular Technology* (Apr. 2020).

[171] Simon Tewes and Aydin Sezgin. “WS-WiFi: Wired Synchronization for CSI Extraction on COTS-WiFi-Transceivers”. In: *IEEE Internet of Things Journal* (June 2021).

[172] Francisco Tirado-Andrés and Alvaro Araujo. “Performance of clock sources and their influence on time synchronization in wireless sensor networks”. In: *International Journal of Distributed Sensor Networks* (2019).

[173] Wenda Li et al. “On CSI and Passive Wi-Fi Radar for Opportunistic Physical Activity Recognition”. In: *IEEE Transactions on Wireless Communications* (Jan. 2022).

[174] Elahe Soltanaghaei et al. "Robust and practical WiFi human sensing using on-device learning with a domain adaptive model". In: *Proceedings of the ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation*. 2020.

[175] Sorachi Kato et al. "CSI2Image: Image Reconstruction From Channel State Information Using Generative Adversarial Networks". In: *IEEE Access* (2021).

[176] Bo Wu et al. "Device-Free Human Activity Recognition With Identity-Based Transfer Mechanism". In: *Wireless Communications and Networking Conference (WCNC)*. IEEE, Mar. 2021.

[177] Dongheng Zhang, Xiong Li, and Yan Chen. "Pushing the Limit of Phase Offset for Contactless Sensing Using Commodity Wifi". In: *International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, June 2021.

[178] Hyuckjin Choi et al. "Wi-CaL: WiFi Sensing and Machine Learning Based Device-Free Crowd Counting and Localization". In: *IEEE Access* (2022).

[179] Marco Cominelli, Francesco Gringoli, and Renato Lo Cigno. "Passive Device-Free Multi-Point CSI Localization and Its Obfuscation with Randomized Filtering". In: *Mediterranean Communication and Computer Networking Conference (MedComNet)*. IEEE, June 2021.

[180] Steven M. Hernandez and Eyuphan Bulut. "Scheduled Spatial Sensing against Adversarial WiFi Sensing". In: *Proceedings of the International Conference on Pervasive Computing and Communications (PerCom)*. Atlanta, USA, Mar. 2023.

[181] Takuya Akiba et al. "Optuna: A Next-generation Hyperparameter Optimization Framework". In: *Proceedings of the SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 2019.

[182] Yiming Wang et al. "From Point to Space: 3D Moving Human Pose Estimation Using Commodity WiFi". In: *Communications Letters* (2021).

[183] Steven M. Hernandez and Eyuphan Bulut. "Online Stream Sampling for Low-Memory On-Device Edge Training for WiFi Sensing". In: *Proceedings of the ACM Workshop on Wireless Security and Machine Learning*. 2022.

[184] Muhammad Sulaiman, Syed Ali Hassan, and Haejoon Jung. "True Detect: Deep Learning-based Device-Free Activity Recognition using WiFi". In: *Wireless Communications and Networking Conference Workshops*. IEEE. 2020.

[185] Mohammud J Bocus et al. "Translation Resilient Opportunistic WiFi Sensing". In: *Proceedings of the International Conference on Pattern Recognition*. IEEE. 2021.

[186] Chong Tang et al. "Occupancy Detection and People Counting Using WiFi Passive Radar". In: *Proceedings of the Radar Conference*. IEEE. 2020.

[187] Nastaran Alishahi, Mazdak Nik-Bakht, and Mohamed M Ouf. "A framework to identify key occupancy indicators for optimizing building operation using WiFi connection count data". In: *Building and Environment* (2021).

[188] Rui Hu et al. "An Unsupervised Behavioral Modeling and Alerting System Based on Passive Sensing for Elderly Care". In: *Future Internet* (2021).

[189] Michel Allegue, N Ghouechian, and N Rozon. "WiFi Motion Intelligence: The Fundamentals". In: *WiFi Motion Intelligence: The Fundamentals* (2020).

[190] Sathiya Kumaran Mani et al. “An Architecture for IoT Clock Synchronization”. In: *Proceedings of the International Conference on the Internet of Things*. 2018.

[191] Steven M. Hernandez and Eyuphan Bulut. “WiFederated: Scalable WiFi Sensing using Edge Based Federated Learning”. In: *Internet of Things Journal* (2021).

[192] Shangqing Liu et al. “DeepCount: Crowd Counting with WiFi via Deep Learning”. In: *arXiv preprint arXiv:1903.05316* (2019).

[193] Fangxin Wang et al. “Channel Selective Activity Recognition with WiFi: A Deep Learning Approach Exploring Wideband Information”. In: *IEEE Transactions on Network Science and Engineering* (2020).

[194] Fei Wang et al. “Can WiFi Estimate Person Pose?” In: *arXiv preprint arXiv:1904.00277* (2019).

[195] Han Zou et al. “Towards occupant activity driven smart buildings via WiFi-enabled IoT devices and deep learning”. In: *Energy and Buildings* (2018).

[196] Wenjun Jiang et al. “Towards Environment Independent Device Free Human Activity Recognition”. In: *Proceedings of the International Conference on Mobile Computing and Networking*. 2018.

[197] Yuanrun Fang et al. “WiTransfer: A Cross-scene Transfer Activity Recognition System Using WiFi”. In: *Proceedings of the ACM Turing Celebration Conference-China*. 2020.

[198] Brendan McMahan et al. “Communication-Efficient Learning of Deep Networks from Decentralized Data”. In: *Artificial Intelligence and Statistics*. PMLR. 2017.

[199] Han Zou et al. “WiFi and Vision Multimodal Learning for Accurate and Robust Device-Free Human Activity Recognition”. In: *Conference on Computer Vision and Pattern Recognition Workshops, CVPR Workshops, Long Beach, CA, USA, June 16-20*. 2019.

[200] Lingchao Guo et al. “From Signal to Image: Capturing Fine-Grained Human Poses With Commodity Wi-Fi”. In: *IEEE Communications Letters* (2019).

[201] Fei Wang et al. “Person-in-WiFi: Fine-grained person perception using WiFi”. In: *Proceedings of the International Conference on Computer Vision*. 2019.

[202] Sinno Jialin Pan and Qiang Yang. “A Survey on Transfer Learning”. In: *IEEE Transactions on knowledge and data engineering* (2009).

[203] Mikhail Yurochkin et al. “Bayesian Nonparametric Federated Learning of Neural Networks”. In: *International Conference on Machine Learning*. PMLR. 2019.

[204] Hongyi Wang et al. “Federated Learning with Matched Averaging”. In: *International Conference on Learning Representations*. 2020.

[205] Sheheryar Arshad et al. “Leveraging transfer learning in multiple human activity recognition using WiFi signal”. In: *Proceedings of the International Symposium on “A World of Wireless, Mobile and Multimedia Networks” (WoWMoM)*. IEEE. 2019.

[206] Yae Jee Cho, Jianyu Wang, and Gauri Joshi. “Client Selection in Federated Learning: Convergence Analysis and Power-of-Choice Selection Strategies”. In: *arXiv preprint arXiv:2010.01243* (2020).

[207] Sauptik Dhar et al. “On-Device Machine Learning: An Algorithms and Learning Theory Perspective”. In: *arXiv preprint arXiv:1911.00623* (2019).

[208] Fan Liang et al. “Toward Edge-Based Deep Learning in Industrial Internet of Things”. In: *IEEE Internet of Things Journal* (2020).

[209] Jean-Christophe Bos. *MicroMLP*. 2018. URL: https://github.com/jczic/MicroMLP.

[210] João Gama et al. “A Survey on Concept Drift Adaptation”. In: *ACM computing surveys (CSUR)* (2014).

[211] Chenli Wang, Kaleb Pattawi, and Hohyun Lee. “Energy saving impact of occupancy-driven thermostat for residential buildings”. In: *Energy and Buildings* (2020).

[212] G. Aravamuthan et al. “Physical Intrusion Detection System Using Stereo Video Analytics”. In: *Proceedings of International Conference on Computer Vision and Image Processing*. Singapore: Springer Singapore, 2020.

[213] Zhaoxiang Zhang, Mo Wang, and Xin Geng. “Crowd counting in public video surveillance by label distribution learning”. In: *Neurocomputing* (2015).

[214] Saandeep Depatla and Yasamin Mostofi. “Crowd Counting Through Walls Using WiFi”. In: *International Conference on Pervasive Computing and Communications (PerCom)*. 2018.

[215] Shiwei Fang, Ron Alterovitz, and Shahriar Nirjon. “Non-Line-of-Sight Around the Corner Human Presence Detection Using Commodity WiFi Devices”. In: *Proceedings of the ACM International Workshop on Device-Free Human Sensing*. ACM. 2019.

[216] Osama Talaat Ibrahim, Walid Gomaa, and Moustafa Youssef. “CrossCount: A Deep Learning System for Device-Free Human Counting Using WiFi”. In: *IEEE Sensors Journal* (2019).

[217] Han Zou et al. “FreeCount: Device-Free Crowd Counting with Commodity WiFi”. In: *Global Communications Conference (GLOBECOM), Singapore, December 4-8, 2017*.

[218] Siwang Zhou et al. “Adversarial WiFi Sensing for Privacy Preservation of Human Behaviors”. In: *IEEE Communications Letters* (2019).

[219] Yue Qiao et al. “PhyCloak: Obfuscating Sensing from Communication Signals”. In: *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. 2016.

[220] Marco Cominelli et al. “An Experimental Study of CSI Management to Preserve Location Privacy”. In: *Proceedings of the ACM International Workshop on Wireless Network Testbeds, Experimental Evaluation & Characterization*. WiNTECH’20. New York, NY, USA, Sept. 2020.

[221] Luis Fernando Abanto-Leon et al. “Stay Connected, Leave no Trace: Enhancing Security and Privacy in WiFi via Obfuscating Radiometric Fingerprints”. In: *Proceedings of the ACM on Measurement and Analysis of Computing Systems* (2020).

[222] Pruthuvi Maheshakya Wijewardena et al. “A Plug-n-Play Game Theoretic Framework For Defending Against Radio Window Atacks”. In: *Proceedings of the ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec)*. 2020.

[223] Marco Cominelli, Francesco Gringoli, and Renato Lo Cigno. “Non Intrusive Wi-Fi CSI Obfuscation Against Active Localization Attacks”. In: *Proceedings of the Conference on Wireless On-demand Network Systems and Services Conference (WONS)*. Klosters, Switzerland, Mar. 2021.

[224] Marco Cominelli, Francesco Gringoli, and Renato Lo Cigno. “AntiSense: Standard-compliant CSI obfuscation against unauthorized Wi-Fi sensing”. In: *Computer Communications* (2021).

[225] Paul Staat et al. “IRShield: A Countermeasure Against Adversarial Physical-Layer Wireless Sensing”. In: *arXiv preprint arXiv:2112.01967* (2021).

[226] Jie Zhang et al. “Defeat Your Enemy Hiding behind Public WiFi: WiGuard Can Protect Your Sensitive Information from CSI-Based Attack”. In: *Applied Sciences* (2018).

[227] Qubeijian Wang. “A Novel Anti-Eavesdropping Scheme in Wireless Networks: Fri-UJ”. In: *Proceedings of the International Conference on Embedded Wireless Systems and Networks.* Junction Publishing. 2019.

[228] Yao Yao et al. “Aegis: An Interference-Negligible RF Sensing Shield”. In: *Proceedings of IEEE International Conference on Computer Communications (INFOCOM).* 2018.

[229] Syed Ayaz Mahmud, Neal Patwari, and Sneha K Kasera. “How to Get Away with MoRTr: MIMO Beam Altering for Radio Window Privacy”. In: *Proceedings of the IEEE International Conference on Mobile Ad Hoc and Smart Systems (MASS).* 2021.

[230] Yanzi Zhu et al. “Et Tu Alexa? When Commodity WiFi Devices Turn into Adversarial Motion Sensors”. In: *Proceedings of the Network and Distributed System Security Symposium, (NDSS), San Diego, California, USA, February 23-26.* 2020.

[231] Bo Tan et al. “Exploiting WiFi Channel State Information for Residential Healthcare Informatics”. In: *IEEE Communications Magazine* (2018).

[232] James Bergstra et al. "Algorithms for Hyper-Parameter Optimization". In: *Advances in Neural Information Processing Systems* (2011).

[233] Steven M. Hernandez and Eyuphan Bulut. "WiFi Sensing on the Edge: Signal Processing Techniques and Challenges for Real-World Systems". In: *IEEE Communications Surveys & Tutorials* (2022).

[234] Xuanzhi Wang et al. "Placement Matters: Understanding the Effects of Device Placement for WiFi Sensing". In: *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* (2022).

[235] Shelly Vishwakarma et al. "SimHumalator: An Open-Source End-to-End Radar Simulator for Human Activity Recognition". In: *IEEE Aerospace and Electronic Systems Magazine* (2021).

[236] Daniel Konings et al. "Do RSSI values reliably map to RSS in a localization system?" In: *Proceedings of Workshop on Recent Trends in Telecommunications Research (RTTR)*. 2017.

[237] Roshan Ayyalasomayajula et al. "Users are Closer than they Appear: Protecting User Location from WiFi APs". In: *arXiv preprint arXiv:2211.10014* (2022).

[238] Apple Inc. *Nearby interactions with U1*. URL: https://developer.apple.com/nearby-interaction/.

[239] Bosch GmbH. *Perfectly keyless – Precise wireless localization and secure key management*. URL: https://www.bosch-mobility-solutions.com/en/solutions/software-and-services/perfectly-keyless/.

[240] Zhaofeng Wu et al. “Dynamic Sparsity Neural Networks for Automatic Speech Recognition”. In: *nternational Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2021.

[241] Hemantha Krishna Bharadwaj et al. “A Review on the Role of Machine Learning in Enabling IoT Based Healthcare Applications”. In: *IEEE Access* (2021).

# VITA

Steven M. Hernandez received a B.S. degree in Computer Science from Virginia Commonwealth University in 2018. He is now completing his Ph.D. in the Computer Science Department of Virginia Commonwealth University with funding through the National Science Foundation Graduate Research Fellowship (2018-2023) and the Virginia Commonwealth University Graduate School Dissertation Assistantship (2023) under the supervision of Dr. Eyuphan Bulut. During his Ph.D., he had internship experiences at Facebook in the area of Spatial Computing (2020) and Google in Ambient Speech Recognition (2022). His research interests include WiFi sensing, TinyML, edge learning, and federated learning.

**Publications**

1. **S. M. Hernandez**, D. Zhao, S. Ding, A. Bruguier, R. Prabhavalkar, T. N. Sainath, Y. He, and I. McGraw, "Sharing Low Rank Conformer Weights for Tiny Always-On Ambient Speech Recognition Models," in Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Rhodes, Greece, Jun. 2023.

2. **S. M. Hernandez** and E. Bulut, "Scheduled Spatial Sensing against Adversarial WiFi Sensing," in Proceedings of the IEEE International Conference on Pervasive Computing and Communications (PerCom), Atlanta, USA, Mar. 2023.

3. **S. M. Hernandez** and E. Bulut, "WiFi Sensing on the Edge: Signal Processing Techniques and Challenges for Real-World Systems," in IEEE Communications Surveys & Tutorials (COMST), Vol. 25, 2023.

4. **S. M. Hernandez**, M. Touhiduzzaman, P. E. Pidcoe and E. Bulut, "Wi-PT:

Wireless Sensing based Low-cost Physical Rehabilitation Tracking," in Proceedings of the IEEE International Conference on E-health Networking, Application & Services (HealthCom), Genoa, Italy, Oct. 2022.

5. **S. M. Hernandez** and E. Bulut, "Online Stream Sampling for Low-Memory On-Device Edge Training for WiFi Sensing," in Proceedings of the ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiseML), Austin, Texas, USA, May 2022.

6. **S. M. Hernandez** and E. Bulut, "WiFederated: Scalable WiFi Sensing using Edge Based Federated Learning," in IEEE Internet of Things Journal (IoTJ), Vol. 9, 2022.

7. **S. M. Hernandez**, D. Erdag, and E. Bulut, "Towards Dense and Scalable Soil Sensing Through Low-Cost WiFi Sensing Networks," in Proceedings of the IEEE Conference on Local Computer Networks (LCN), Edmonton, Canada, Oct. 2021.

8. **S. M. Hernandez** and E. Bulut, "Adversarial Occupancy Monitoring using One-Sided Through-Wall WiFi Sensing," in Proceedings of the IEEE International Conference on Communications: IoT and Sensor Networks Symposium (ICC), Montreal, Canada, Jun. 2021.

9. **S. M. Hernandez** and E. Bulut, "Using perceived direction information for anchorless relative indoor localization," Elsevier Journal of Network and Computer Applications (JNCA), Vol. 165, 2020.

10. **S. M. Hernandez** and E. Bulut, "Lightweight and Standalone IoT based WiFi Sensing for Active Repositioning and Mobility," in Proceedings of the Interna-

tional Symposium on "A World of Wireless, Mobile and Multimedia Networks" (WoWMoM), Cork, Ireland, Jun. 2020.

11. **S. M. Hernandez** and E. Bulut, "Performing WiFi Sensing with Off-the-shelf Smartphones," in Proceedings of the IEEE International Conference on Pervasive Computing and Communications Demos: IEEE International Conference on Pervasive Computing and Communications Demonstrations (PerCom Demos), Austin, Texas, USA, Mar. 2020.

12. **S. M. Hernandez** and E. Bulut, "TrinaryMC: Monte Carlo Based Anchorless Relative Positioning for Indoor Positioning," in Proceedings of the IEEE Consumer Communications & Networking Conference (CCNC), Las Vegas, Nevada, USA, Jan. 2020.

13. E. Bulut, **S. M. Hernandez**, A. Dhungana, and B. K. Szymanski, "Is Crowdcharging Possible?" in Proceedings of the IEEE International Conference on Computer Communication and Networks (ICCCN), Jul. 2018.