WirelessEye – Seeing over WiFi Made Accessible

Philipp H. Kindt[†], Cristian Turetta[¶], Florenc Demrozi[§],
Alejandro Masrur[†], Graziano Pravadelli[*], and Samarjit Chakraborty[‡]
[*]*Dept. of Engineering for Innovation Medicine, University of Verona*, Italy, `name.surname@univr.it`
[¶]*Dept. of Computer Science, University of Verona*, Italy, `name.surname@univr.it`
[†]*Faculty of Computer Science, TU Chemnitz*, Germany, `name.surname@informatik.tu-chemnitz.de`
[‡]*Dept. of Computer Science, University of North Carolina at Chapel Hill*, USA, `samarjit@cs.unc.edu`
[§]*Dept. of Electrical Engineering and Computer Science, University of Stavanger*, Norway, `name.surname@uis.no`

2024 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops) | 979-8-3503-0436-7/24/$31.00 ©2024 IEEE | DOI: 10.1109/PerComWorkshops59983.2024.10503162

***Abstract*—While commonly used for communication, recently, WiFi is increasingly being used for sensing. In particular, wireless signals are altered (i.e., absorbed, reflected, and attenuated) by the human body and objects in the environment. This can be perceived by an observer to infer information on the environment and hence, to *"see"* over WiFi. So far, works in this area have led to a variety of custom software tools – each designed for a specific purpose. Moreover, given how scattered the literature is, it is difficult to even identify all processing steps or functionalities necessary for WiFi sensing. To the best of our knowledge, there has been no effort towards a generic solution that helps promote further research and boost new applications in the area. With this as a motivation, we propose *WirelessEye*, a freely available, generic software framework that allows bootstrapping WiFi sensing systems on low-cost hardware, such as a Raspberry Pi. WirelessEye consolidates all necessary processing steps in a single framework, from collecting and visualizing data to executing different machine learning models in real-time for the purpose of comparison. This way, researchers and practitioners can focus on aspects of their research/applications rather than dealing with the many implementation hurdles of WiFi sensing.**

***Index Terms*—WiFi Sensing, Channel State Information, Activity Recognition, Pattern Recognition**

I. Introduction

WiFi Sensing: WiFi is widely used for communication, e.g., for connecting smartphones and laptops with each other and to the Internet. This has catapulted WiFi to ubiquity over the last decade and, hence, WiFi signals are almost everywhere. While propagating from a sender to a receiver, WiFi signals are altered by the human body and objects in the environment. These alterations in the received signals can be analyzed, e.g., using machine learning (ML) methods, for recognizing activities [1], [2], the number of people in a room [3], their indoor [4] and outdoor positions [5], spoken words [6], people's identity [7] and whether they are smoking [8] or eating [9]. Further, it is also possible to measure the heart rate [10], [11], respiratory rate [12], [13], body temperature [14], sleep quality [15] and the emotional status [16], as well as to perform gait analysis [17], pose estimation [18], and gesture recognition [19]. In other words, besides communication, WiFi networks can actually be used to "see". Moreover, recently, the *IEEE 802.11bf Task Group* has been created with the goal of standardizing WiFi sensing in the future, which further underpins the growing importance of this area.

Compared to other technologies, WiFi sensing has a number of decisive advantages. First, no sensors need to be worn on the body. Second, WiFi networks are already deployed for communication purposes and, thus, WiFi sensing is more economical than most other wireless approaches, e.g., Ultra-Wideband (UWB) or radar-based ones. Third, WiFi sensing is intrinsically privacy-preserving, since it reveals significantly less personal data than, e.g., camera-based approaches [20].

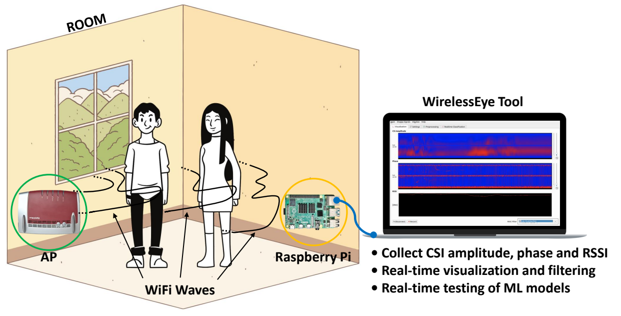

ROOM
WirelessEye Tool
AP
WiFi Waves
Raspberry Pi
• Collect CSI amplitude, phase and RSSI
• Real-time visualization and filtering
• Real-time testing of ML models

Fig. 1. A typical WirelessEye-based sensing scenario.

A typical scenario involving WiFi sensing is shown in Figure 1. It consists of one or more access points (APs) that are already installed in a room or environment, a passive observer, and one or more human subjects performing different activities. The observer extracts the *channel state information* (CSI) from WiFi signals in the environment and analyzes it to infer information on human activity or changes in the environment. This is typically done using ML methods, such as support vector machines (SVMs), decision trees (DTs), or more complex deep learning models. These techniques need to be first trained using labeled CSI data to learn how the signal is affected by different events of interest. Once the ML model has been trained, it can classify previously unseen CSI data [21]–[23].

Setting up a base system: There is a significant effort associated with setting up a base system for WiFi sensing, which – in our opinion – is slowing down the progress in this area. In particular, prior to any research on the actual classification, e.g., training ML methods, CSI data needs to be collected, formatted, pre-processed, visualized, and stored. In particular, the following steps are necessary for creating a WiFi-based sensing system. First, since most off-the-shelf radios do not grant their host computer access to CSI data, a custom firmware needs to be flashed onto the

WiFi radio, which is provided by open-source projects such as Nexmon [24], [25], PicoScenes [26], the Linux 802.11n CSI Tool [27], or the Atheros CSI Tool [28]. Second, CSI data cannot be fed directly into an ML framework such as Tensorflow or PyTorch. However, multiple parameterizable, application-specific preprocessing, filtering, and data format conversion steps must be applied before the data becomes usable. These steps are generally implemented separately from each other and require different tools, programming languages, and frameworks. Third, after implementing an ML method, its performance needs to be evaluated in a real-world setting. Rather than analyzing offline data, such an evaluation requires a workflow that streams the preprocessed CSI data to the ML-based classifier and records classification results in real-time. This typically requires combining and adjusting different software tools and frameworks, leading to a significant effort. To the best of our knowledge, no software framework covers all of these steps together.

Contributions: This work aims to substantially reduce the initial effort in developing a WiFi sensing system. To this end, we introduce a software tool called *WirelessEye* and make it available to the public domain under an open-source license[1]. It provides a ready-to-use base workflow, such that downstream WiFi sensing applications can focus on the machine learning aspects. Unlike most existing tools, WirelessEye is optimized for simplicity. Therefore, the entire workflow is interactive, intuitive, fully supported by a graphical user interface (GUI), and works "out of the box." Because WirelessEye provides immediate feedback on different design choices and options, it can also be used for educational purposes. Figure 2 provides an overview of the WirelessEye GUI, which will be explained in detail later.

WirelessEye supports interactively optimizing WiFi sensing setups, which is crucial for the proper functioning of the system. In particular, determining the positions of the senders (i.e., APs) and the observer that best suit the intended application significantly impacts the performance of a WiFi sensing system. Typically, this optimization is carried out iteratively by repeatedly modifying the setup, recording a sequence of CSI data and analyzing the impact of these modifications on the captured CSI data, e.g., using multiple plots. Multiple such iterations are often necessary until a suitable configuration is found, which can be a time-consuming and cumbersome procedure.

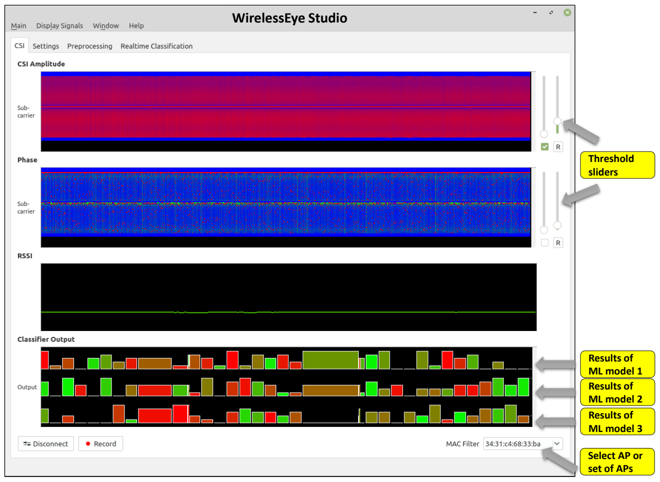
Using WirelessEye, the preprocessed CSI data is visualized *in real-time* and hence, the impact of every change can therefore be observed immediately. Further, WirelessEye can help optimize a setup interactively, leading to better results with a reduced effort. Also, WiFi sensing systems often fail to generalize to environments different from those in which they were trained [1]. By offering the possibility of visualizing classifier outputs in real-time, WirelessEye helps in detecting situations in which classification algorithms fail.

[1]Available at https://github.com/pkindt/WirelessEye

WirelessEye works together with a Raspberry Pi using a Nexmon-based firmware [24], [25], as well as literally any ML framework, e.g., Tensorflow, Keras, or PyTorch. WirelessEye provides all required steps for WiFi sensing, including data recording, format conversion, preprocessing, and visualization in a ready-to-use manner. The entire workflow can be parameterized and optimized online, while visualizing the effects of every adjustment in real-time.

In this paper, we make the following contributions:

1) We compile available knowledge from the literature into a ready-to-use software framework with real-time data visualization capabilities, viz., WirelessEye, which eases the design and optimization of WiFi sensing systems.
2) Further, by providing WirelessEye as an open-source tool, we make this topic accessible to a broader public beyond the wireless communications community.
3) Finally, we illustrate the proper functioning and advantages of WirelessEye using a case study related to activity recognition at a kitchen sink.



MAC Filter: 34:31:c4:68:33:ba

Fig. 2. Screenshot of WirelessEye's GUI.

We next provide a brief background on WiFi sensing using CSI and then present an overview of WirelessEye.

II. Background: Sensing based on CSI data

When a wireless signal X is sent, the signal Y received by another device is given by

$$\mathbf{Y} = \mathbf{H} \circ \mathbf{X} + \mathbf{N}. \tag{1}$$

Here, $\circ$ represents an element-wise multiplication. The term N describes the impact of noise. H is called the *channel state information (CSI)*. It describes how the wireless signal is altered when propagating through the environment. WiFi (i.e., the IEEE 802.11ac protocol) supports channels of $20 MHz$ to $160 MHz$ bandwidth. Using Orthogonal Frequency Division Multiplexing (OFDM), each channel is subdivided into $N = 64$ (for $20 MHz$ channels) to $N = 512$ (for $160 MHz$ channels) subcarriers, with data being transmitted simultaneously on all of them, except for a few guard subcarriers. Due to the small bandwidth, the channel for each subcarrier is assumed to be *flat*, i.e., all frequencies of a subcarrier undergo approximately the same perturbations. Hence, $H = [h_1, h_2, ..., h_N]$ from a received WiFi frame is a vector of complex values h_i, where the index i identifies the subcarrier. The magnitude of every complex number h_i represents the signal's attenuation, whereas the phase expresses the phase change induced, e.g., by human motion in the environment.

Using automatic gain control (AGC), every WiFi receiver continuously adjusts the CSI magnitude on average over all subcarriers to lie within a certain range. Hence, the measured CSI data relates to momentary changes in attenuation, while persistent changes are canceled out by AGC. In contrast, the *received signal strength indicator (RSSI)* accounts for the average received power over all subcarriers after antenna and cable loss, but before AGC takes effect. The RSSI is usually an integer that indicates the (average) received power in dBm. A WiFi-based sensing system typically evaluates the progress of the CSI and/or RSSI data over time. We next describe how this is done in WirelessEye.

III. The WirelessEye Framework

In this section, we first provide an overview of WirelessEye and then describe its components in more detail.

A. Overview

Figure 3 shows a WiFi sensing setup involving WirelessEye. The green block on the left represents multiple access points (APs) that emit WiFi signals. The yellow block in the middle represents a CSI observer, e.g., a Raspberry Pi board, which captures WiFi signals and extracts CSI and RSSI data. The gray block on the right represents a laptop/PC or the Raspberry Pi itself, which performs signal processing, classification, and visualization through WirelessEye. CSI data, which has been recorded into files, is used to train ML models for pattern recognition, which can also be queried in real-time using WirelessEye's live export feature, as shown in Figure 2 under *Classifier Output*.

B. WirelessEye's Components

WirelessEye mainly consists of two software packages. The *CSI Server* is deployed on the Raspberry Pi and an intuitive GUI called *WirelessEye Studio* is executed on a PC or laptop (cf. Figure 3), although it can also be executed on the Raspberry Pi, if desired. The CSI server collects data from a Nexmon-based [24] radio firmware and makes it available via a TCP socket. After receiving the data from the *CSI Server*, the *WirelessEye Studio* GUI mainly carries out the following 3 tasks: a) data management, b) CSI preprocessing, and c) CSI visualization.

Figure 4 summarizes the steps that need to be carried out to implement an arbitrary, CSI-based pattern recognition scenario using WirelessEye, based on a previously prepared Raspberry Pi. After downloading (1st block), compiling, and running WirelessEye (2nd block), one can collect data by using WirelessEye (3rd block). To facilitate setting up an initial ML model, WirelessEye additionally provides two preconfigured Python scripts. These scripts interface TensorFlow [29] as one of the most popular ML frameworks in a ready-to-use manner. In particular, a generic, CNN-based pattern recognition model is generated through the *train_model.py* script. This model can be trained for different purposes using previously recorded CSI data. The *classify.py* script can be executed from within *WirelessEye Studio* to query the designed ML models in real-time. The classification results are visualized in real-time in WirelessEye Studio – see *Classifier Output in Figure 2*. We next describe these functionalities in more detail.

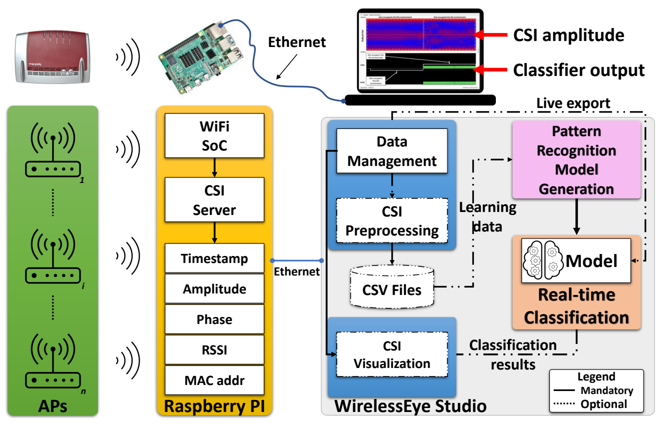


Fig. 3. Illustration of WirelessEye's workflow.

1) Data Management: As mentioned above, CSI data can be recorded into data files, exported to a classifier, and visualized in real-time. WirelessEye provides different preprocessing steps and data formats for each of these possibilities. For example, it might be beneficial to visualize only data belonging to certain transmitters of interest while simultaneously exporting data belonging to all transmitters into a *.csv* file. WirelessEye internally converts data into the required format and allows filtering for one or multiple transmitter/MAC addresses, which can be chosen interactively at runtime. WirelessEye supports recording files in three different formats, i.e., one comma-separated value (*.csv*) format optimized for simplicity, one *.csv* format optimized for size, and a binary format for minimalistic memory requirements. These files can be read by different standard tools, e.g., MATLAB, as well as the scrips for ML model generation discussed above. In addition to exporting data to a file, WirelessEye can launch an arbitrary executable, e.g., one or more ML models, from its GUI. The CSI data is then streamed directly to the selected executable by writing to its standard input, while classification results are read from its standard output and visualized along with the CSI data in real-time, as depicted in Figure 2 under *Classifier Output*.

2) CSI Preprocessing: Data received from the patched WiFi radio needs to be preprocessed before being stored or exported to the classification model. WirelessEye provides an

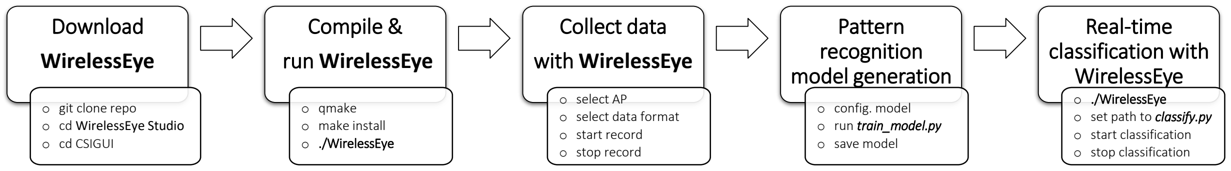
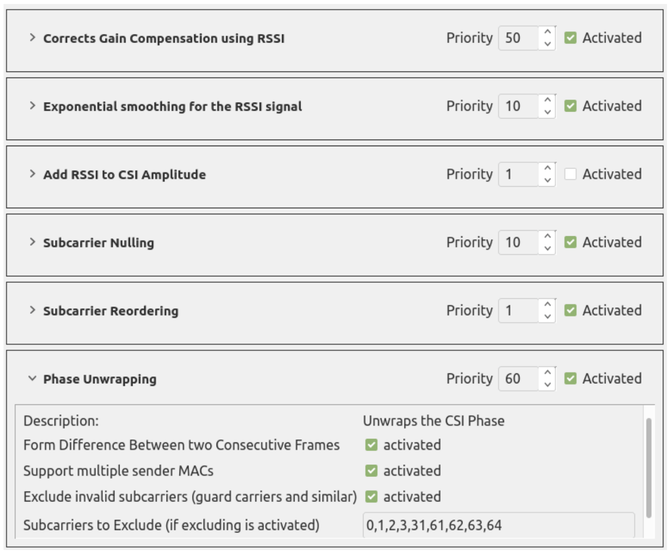


Fig. 4. Steps required to create a new CSI-based recognition system using WirelessEye.



Corrects Gain Compensation using RSSI	Priority 50	✓ Activated
Exponential smoothing for the RSSI signal	Priority 10	✓ Activated
Add RSSI to CSI Amplitude	Priority 1	Activated
Subcarrier Nulling	Priority 10	✓ Activated
Subcarrier Reordering	Priority 1	✓ Activated
Phase Unwrapping	Priority 60	✓ Activated

Description:	Unwraps the CSI Phase
Form Difference Between two Consecutive Frames	✓ activated
Support multiple sender MACs	✓ activated
Exclude invalid subcarriers (guard carriers and similar)	✓ activated
Subcarriers to Exclude (if excluding is activated)	0,1,2,3,31,61,62,63,64

Fig. 5. Parameterization of preprocessing plugins.

extensible architecture for this purpose. In particular, each preprocessing step is realized by a plugin. Plugins are linked dynamically during runtime, can be compiled independently from WirelessEye, and are easy to implement.

As shown in Figure 5, WirelessEye provides control over the CSI sensing pipeline through a set of GUI tabs. In particular, through the tab shown in Figure 5, WirelessEye allows the selection of a group of active plugins and determines their execution order by assigning individual priorities. Furthermore, every plugin can expose multiple adjustable parameters to the GUI, and the parameterization can be tuned interactively during runtime. Other tabs offer functionalities for specifying the connection to the CSI Server, selecting the CSI bandwidth for analysis, determining storage and file format options, and configuring visualization settings. WirelessEye supports a generic preprocessing workflow by default, whose individual steps are described next.

MAC filtering: When capturing WiFi frames, data from all APs in range is received, of which only a few are relevant. WirelessEye allows selecting a set of APs during runtime.

Amplitude/phase extraction: From the complex values h_i, $i = 1 ... n$, WirelessEye extracts the amplitudes a_i and phases Φ_i, which can then be processed separately.

Bandwidth narrowing: WiFi uses channels with different bandwidths, and the BCM43455c0 WiFi radio on the Raspberry Pi 4B can monitor channels from 20 MHz to 80 MHz. When a transmitter only uses, e.g., 20 MHz, whereas the radio captures using 80 MHz, most of the data does not contain any usable information. WirelessEye supports suitably narrowing the bandwidth of its input data to avoid problems in this case.

Subcarrier removal: WirelessEye can set the data related to subcarriers that do not carry any valid information, e.g., guard subcarriers, to zero.

Subcarrier reordering: CSI values from different subcarriers received by the WiFi radio are sometimes not ordered in a linear manner. WirelessEye reorders them if needed.

AGC compensation: As mentioned above, WiFi radios use AGC to bring the signal amplitude and, hence, CSI into a desired range. WirelessEye reverses the impact of AGC using the method described in [30].

Denoising RSSI: WirelessEye provides an exponential smoothing method to denoise RSSI and, thereby, also improve the AGC compensation.

Phase unwrapping and cleansing: The resulting phase information from the WiFi radio contains multiple discontinuities caused by the phase "wrapping around" for values greater than $2 \cdot \pi$ or lower than 0. WirelessEye provides a heuristic to remove such phase discontinuities.

3) CSI Visualization: Visualizing CSI data in real-time is crucial for optimizing any WiFi sensing setup. WirelessEye can simultaneously visualize the CSI amplitude, phase, RSSI, and the classification results of multiple ML models in real-time, as shown in Figure 2. It is challenging to meaningfully visualize CSI data in real-time since it has three dimensions (i.e., time, subcarrier, and the actual, complex value). Most available tools do not provide satisfactory plots, since they either depict CSI values (i.e., amplitudes/phases) of a few (selected) subcarriers over time, or all subcarriers at a single point in time. On the other hand, with packet rates of *100 Hz* or higher, it is also not practical to repeatedly plot the entire CSI data for every single frame, since most information on human activity is encoded in rather slow variations over time. Further, CSI amplitude typically exhibits a large dynamic range, and the visualization needs to focus on those ranges of values that represent the events of interest. However, which range of values contains information related to the events of interest is not known a priori. As a result, the visualization itself needs to be adjustable to magnify such ranges of values on demand.

In WirelessEye, CSI amplitude and phase are displayed using two independent, synchronized, 3-dimensional plots, of which one depicts the amplitude, whereas the other one depicts the phase (see Figure 2). The first dimension (i.e., the x-axis) is given by the reception time of the frame. The second dimension (i.e., the y-axis) represents the subcarrier index, and the third dimension is formed by the CSI amplitude or phase. This third dimension is encoded by the color: low amplitudes/phases are plotted in blue, high ones in red, while shadings between blue and red are used for intermediate values. The visualized CSI data is updated in real-time from right to left, thereby, depicting the progression of CSI data over time.

On the right-hand side of the CSI visualization (cf. Figure 2), there are two sliders, which can be used to select two thresholds that restrict the range of CSI amplitudes or phases to be visualized. All values that lie outside of the selected range are omitted, whereas the values in range are re-scaled to cover all possible colors. Hence, the sliders can be used to selectively visualize the ranges of values that are most sensitive to a certain event or activity.

WirelessEye allows visualizing classification results on incoming CSI data in real-time. To this end, horizontal bars are drawn synchronously to the CSI data. The height of each bar represents the model output to which the corresponding sequence of CSI data has been assigned, whereas the bar's color depicts the classification confidence (i.e., red means low and green means high confidence). This allows examining how a classifier reacts to particular environmental inputs (e.g., the number of people present in a room, their motion, etc.) in real-time and helps to evaluate, debug and optimize the designed WiFi sensing system.

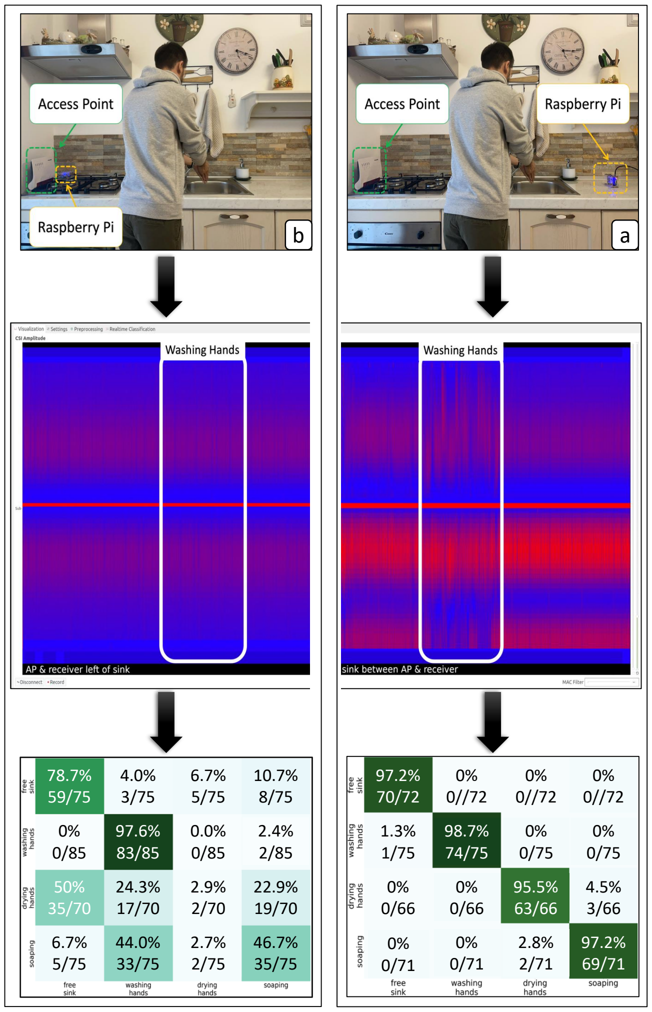
IV. Case Study

This section presents a practical experiment utilizing WirelessEye. The goal is to demonstrate WirelessEye's functioning and that WiFi-based sensing systems built using it can achieve high accuracy. Furthermore, our study shows the impact of optimizing the sensing setup by selecting different positions of the sender and the receiver. As already described, WirelessEye supports this optimization using its real-time visualization capability. The case study consists of recognizing activities carried out in a kitchen near the sink.

A. Experimental Setup

Figure 6 illustrates the CSI amplitude of two different setups. In the first setup, as depicted in Figure 6 (a), the WiFi router/AP is placed on the left side of the kitchen sink, while the Raspberry Pi 4B is placed on the right side. In the second setup, as shown in Figure 6 (b), both the WiFi router/AP and the Raspberry Pi 4B are positioned on the same side of the sink. The goal is to recognize different human activities using these setups, i.e., washing, drying and soaping hands, as well as whether the sink is not being used.

By comparing the two setups, it can be observed that CSI data is highly sensitive to hand-washing gestures when the sink is situated between the WiFi router/AP and the Raspberry Pi 4B. This sensitivity can already be observed in the real-time visualization of WirelessEye (cf. Figure 6). When both devices are on the same side of the sink, CSI data becomes less sensitive to the hand-washing gestures, resulting in a



	free sink	washing hands	drying hands	soaping
free sink	78.7% 59/75	4.0% 3/75	6.7% 5/75	10.7% 8/75
washing hands	0% 0/85	97.6% 83/85	0.0% 0/85	2.4% 2/85
drying hands	50% 35/70	24.3% 17/70	2.9% 2/70	22.9% 19/70
soaping	6.7% 5/75	44.0% 33/75	2.7% 2/75	46.7% 35/75

	free sink	washing hands	drying hands	soaping
free sink	97.2% 70/72	0% 0//72	0% 0//72	0% 0//72
washing hands	1.3% 1/75	98.7% 74/75	0% 0/75	0% 0/75
drying hands	0% 0/66	0% 0/66	95.5% 63/66	4.5% 3/66
soaping	0% 0/71	0% 0/71	2.8% 2/71	97.2% 69/71

Fig. 6. Real-time visualization and evaluation results: On the left of the image, sender and receiver are on the same side of the sink, whereas the sink is between them on the right of the image.

relatively constant CSI amplitude. This demonstrates that the sensing setup can be effectively optimized prior to training an ML model. In particular, by analyzing the real-time visualization and optimizing the preprocessing capabilities provided by WirelessEye, users are able to observe and understand the behavior of the CSI data during different activities. For example, they can systematically optimize the positions of the AP and the Raspberry Pi to maximize the sensitivity of the CSI data to the considered activity, without needing to train and analyze a ML model. This leads to a higher quality of the input data.

B. Evaluation

To numerically demonstrate the effectivity of optimizing a setup using WirelessEye, we collected data from both mentioned setups and used 75% of the recorded data to train our model and 25% to evaluate the accuracy of the resulting classification, based on 1-second time windows. For each window, we compared the classification results of the model with the actually performed activity. We assessed the performance in terms of the F1-score, which is a standard metric that combines precision and recall. When both devices were on the same side of the sink, the classifier was unable to distinguish well between the different activities, with an overall F1-score of only 0.5. When the sink was between both devices, the F1-score drastically improved to 0.97. In this case, the model could correctly classify the sink as being idle in 97 % of all windows, soaping hands in 97 %, washing hands in 99 %, and drying hands in 96 % of all windows. Figure 6 presents the detailed results for each of the tested activities and setups. Our results show that accurate WiFi sensing systems can be created with negligible effort using WirelessEye. In particular, with WirelessEye's real-time visualization, it is, in fact, possible to optimize the WiFi sensing setup's performance before actually training any ML model.

V. Concluding Remarks

Implementing a WiFi-based sensing system involves a significant effort, which goes beyond designing the ML models for the actual recognition task. Furthermore, detailed knowledge from different fields is required for developing such a system. In this paper, we presented WirelessEye, which significantly reduces this effort by providing all necessary steps for recording, processing and visualizing CSI data "out-of-the-box." Furthermore, through an intuitive user interface and real-time visualization, WirelessEye supports interactively optimizing a WiFi sensing setup, which is essential for achieving high recognition accuracies. By making WirelessEye publicly available, we intend to motivate more participation from other domains, for example, from the machine learning community. In this manner we expect to foster more innovation and help develop new applications of WiFi sensing.

References

[1] M. Cominelli, F. Gringoli, and F. Restuccia, "Exposing the CSI: A systematic investigation of CSI-based Wi-Fi sensing capabilities and limitations," in *IEEE International Conference on Pervasive Computing and Communications (PerCom)*, 2023, pp. 81–90.

[2] N. Gupta, S. K. Gupta, R. K. Pathak, V. Jain, P. Rashidi *et al.*, "Human activity recognition in artificial intelligence framework: a narrative review," *Artificial Intelligence Review*, pp. 1–54, 2022.

[3] F. Demrozi, C. Turetta, F. Chiarani, P. H. Kindt, and G. Pravadelli, "Estimating indoor occupancy through low-cost BLE devices," *IEEE Sensors Journal*, vol. 21, no. 15, pp. 17 053–17 063, 2021.

[4] I. Neupane, B. Alsinglawi, and K. Rabie, "Indoor positioning using Wi-Fi and machine learning for industry 5.0," in *IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops)*, 2023, pp. 359–362.

[5] G. Solmaz, P. Baranwal, and F. Cirillo, "Countmein: Adaptive crowd estimation with Wi-Fi in smart cities," in *IEEE International Conference on Pervasive Computing and Communications (PerCom)*, 2022, pp. 187–196.

[6] C. Du, X. Yuan, W. Lou, and Y. T. Hou, "Context-free fine-grained motion sensing using WiFi," in *Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*, 2018, pp. 1–9.

[7] P. Connor and A. Ross, "Biometric recognition by gait: A survey of modalities and features," *Computer vision and image understanding*, vol. 167, pp. 1–27, 2018.

[8] X. Zheng, J. Wang, L. Shangguan, Z. Zhou, and Y. Liu, "Smokey: Ubiquitous smoking detection with commercial WiFi infrastructures," in *IEEE International Conference on Computer Communications (INFOCOM)*, 2016, pp. 1–9.

[9] Z. Lin, Y. Xie, X. Guo, Y. Ren, Y. Chen *et al.*, "Wieat: Fine-grained device-free eating monitoring leveraging Wi-Fi signals," in *IEEE International Conference on Computer Communications and Networks (ICCCN)*, 2020, pp. 1–9.

[10] J. Liu, Y. Chen, Y. Wang, X. Chen, J. Cheng *et al.*, "Monitoring vital signs and postures during sleep using WiFi signals," *IEEE Internet of Things Journal*, vol. 5, no. 3, pp. 2071–2084, 2018.

[11] Y. Gu, X. Zhang, Z. Liu, and F. Ren, "WiFi-based real-time breathing and heart rate monitoring during sleep," in *IEEE Global Communications Conference (GLOBECOM)*, 2019, pp. 1–6.

[12] Y. Zeng, D. Wu, J. Xiong, E. Yi, R. Gao *et al.*, "Farsense: Pushing the range limit of WiFi-based respiration sensing with CSI ratio of two antennas," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 3, no. 3, pp. 1–26, 2019.

[13] X. Wang, C. Yang, and S. Mao, "Resilient respiration rate monitoring with realtime bimodal CSI data," *IEEE Sensors Journal*, vol. 20, no. 17, pp. 10 187–10 198, 2020.

[14] V. Ha and A. Nahapetian, "Received WiFi signal strength monitoring for contactless body temperature classification," in *EAI International Conference on Body Area Networks*. Springer, 2021, pp. 112–125.

[15] S. Yue, Y. Yang, H. Wang, H. Rahul, and D. Katabi, "Bodycompass: Monitoring sleep posture with wireless signals," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 4, no. 2, pp. 1–25, 2020.

[16] Y. Gu, T. Liu, J. Li, F. Ren, Z. Liu *et al.*, "Emosense: Data-driven emotion sensing via off-the-shelf WiFi devices," in *IEEE International Conference on Communications (ICC)*, 2018, pp. 1–6.

[17] L. Zhang, C. Wang, M. Ma, and D. Zhang, "WiDIGR: Direction-independent gait recognition system using commercial Wi-Fi devices," *IEEE Internet of Things Journal*, vol. 7, no. 2, pp. 1178–1191, 2019.

[18] W. Jiang, H. Xue, C. Miao, S. Wang, S. Lin *et al.*, "Towards 3d human pose construction using WiFi," in *Annual International Conference on Mobile Computing and Networking (MobiCom)*, 2020, pp. 1–14.

[19] C. Li, M. Liu, and Z. Cao, "WiHF: Enable user identified gesture recognition with WiFi," in *IEEE Conference on Computer Communications (INFOCOM)*. IEEE, 2020, pp. 586–595.

[20] B. Feng, Y. Lin, T. Xu, and J. Duan, "A survey on privacy preservation in video big data," in *International Conference on Electrical, Computer, Communications and Mechatronics Engineering (ICECCME)*, 2021, pp. 1–6.

[21] C. Li, Z. Cao, and Y. Liu, "Deep AI enabled ubiquitous wireless sensing: A survey," *ACM Computing Surveys (CSUR)*, vol. 54, no. 2, pp. 1–35, 2021.

[22] J. M. Fernandes, J. S. Silva, A. Rodrigues, and F. Boavida, "A survey of approaches to unobtrusive sensing of humans," *ACM Computing Surveys (CSUR)*, vol. 55, no. 2, pp. 1–28, 2022.

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

[24] F. Gringoli, M. Schulz, J. Link, and M. Hollick, "Free your CSI: A channel state information extraction platform for modern Wi-Fi chipsets," in *International Workshop on Wireless Network Testbeds, Experimental Evaluation& Characterization (WiNTECH)*, 2019, p. 21–28.

[25] M. Schulz, D. Wegemer, and M. Hollick. (2017) Nexmon: The C-based firmware patching framework. [Online]. Available: https://nexmon.org

[26] Z. Jiang, X. Wang, C. He, and R. Li, "PicoScenes: enabling UWB sensing array on COTS Wi-Fi platform," in *International Conference on Embedded Wireless Systems and Networks (EWSN)*, 2019, pp. 264–266.

[27] D. Halperin, W. Hu, A. Sheth, and D. Wetherall, "Tool release: Gathering 802.11 n traces with channel state information," *ACM SIGCOMM computer communication review*, vol. 41, no. 1, pp. 53–53, 2011.

[28] Y. Xie, Z. Li, and M. Li, "Precise power delay profiling with commodity WiFi," in *Annual International Conference on Mobile Computing and Networking*, 2015, pp. 53–64.

[29] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen *et al.*, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[30] Z. Gao, Y. Gao, S. Wang, D. Li, and Y. Xu, "Crisloc: Reconstructable CSI fingerprinting for indoor smartphone localization," *IEEE Internet of Things Journal*, vol. 8, no. 5, pp. 3422–3437, 2021.