Towards Continuous Mobile Sensing for Remote COPD Monitoring

by

Daniyal Liaqat

A thesis submitted in conformity with the requirements
for the degree of Doctor of Philosophy
Graduate Department of Computer Science
University of Toronto

Abstract

Towards Continuous Mobile Sensing for Remote COPD Monitoring

Daniyal Liaqat
Doctor of Philosophy
Graduate Department of Computer Science
University of Toronto
2020

Chronic Obstructive Pulmonary Disease (COPD) is a debilitating and life-threatening disease. In 2016 there were an estimated 251 million cases of COPD globally and the World Health Organization predicts that by 2030 COPD will be the third leading cause of death worldwide. Technologies that help people with COPD manage their condition could have significant impact on their lives. The work presented in this thesis outlines a system that uses wearable and mobile devices to passively sense and monitor patients with COPD.

Mobile and wearable devices contain a myriad of sensors and have been used in applications ranging from earthquake detection to flight control for drones. To make these devices relevant for COPD monitoring, this thesis focuses on two signals that can be extracted from wearable sensors, respiratory rate and coughing. To detect respiratory rate, we propose WearBreathing – our system for respiratory rate detection using the accelerometer and gyroscope sensors found in smartwatches. While respiratory rate from a smartwatch has been done in previous works, existing methods are only accurate in in-lab settings and while participants are stationary, making them unsuitable for remote monitoring. Therefore, WearBreathing is designed specifically to operate in the wild and we show that it is indeed more accurate in the wild than existing methods.

Similar to respiratory rate, we found that existing cough detection solutions do not perform well in the wild. Using an in-the-wild dataset that we collect from COPD patients, we first characterize the sounds captured by a smartwatch microphone in a wild setting. Using our dataset, we build a state of the art cough detector, which we call CoughWatch that works on in-the-wild data and is more accurate than existing cough detectors.

Finally, because mobile devices are resource constrained devices designed for intermittent use, battery life becomes a significant concern when attempting to continuously monitor sensor data. End users, such as patients with COPD, are unlikely to use a device that provides only a few hours of battery life per charge. Therefore, we propose Sidewinder, a developer friendly hardware architecture for energy efficient continuous sensing on mobile devices.

To my parents, who have always been a source of support, motivation and inspiration

Acknowledgements

I am incredibly grateful for my PhD experience – it has been a journey filled with learning and growth that despite the ups and downs, I have thoroughly enjoyed. This experience would not have been anywhere near as fulfilling without the two best advisors I think one could ask for, Professors Eyal de Lara and Frank Rudzicz. It's unfortunate that you can only have two official advisors because Dr. Robert Wu has played an equally important role in this PhD. Every discussion with these three, I walked away feeling I knew more than when I walked in.

I would like to thank the friends who have been there throughout this journey. The many impromptu desk-side discussions and even more many coffee breaks have been a cornerstone of my graduate school experience. In this regard I would like to especially thank Mohamed, Mazen, Nauman, James, Hossein, Caleb and Mickey. I would also like to thank the fantastic collaborators, lab mates and co-workers I have had the pleasure of working with - Dr. Andrea Gerson, Tanya, Jun Lin, Tina, Pegah, Mahbubur and Jilong to name a few. Then there's the students who were role models for me when I joined and made my transition to grad school much easier. Nosayba, George, Sahil, Daniel, Ioan and Andy – thank you.

None of this would have been possible without my family, all of whom have been endlessly supportive. My deepest gratitude goes to my parents who have given so much to myself and my siblings and instilled in all of us a passion for education and learning. I would also like to recognize my grandparents and soon to be parents-in-law for their constant kind words and prayers. And my siblings, whether here at UofT or 2075 miles away in Vancouver, the countless discussions with Amna, Salaar, Hayyan and Taha have been my way of organizing my thoughts, so this maybe should be an apology as well as a thanks. But now that they are on their own journeys, I will try to be there for them as much as they have been for me. I am also excited and looking forward to starting the next chapter of my life with my fiancée, Qanita.

Finally, all praise goes to God, Lord of the worlds. The Most Gracious, the Most Merciful, in whose name I have embarked on this journey of knowledge.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Applications that make use of continuous sensing in smartdevices have potential to greatly impact our lives. Their uses range from early earthquake detection [72] to pervasive games such as Pokemon GO. One particularly interesting and beneficial application of continuous sensing is health monitoring. Using ubiquitous mobile devices to monitor peoples health could provide significant benefits to health care systems and improve the lives of patients, doctors and caregivers. Mobile and wearable technology provide new opportunities to continuously collect and process objective sensor data over extended periods of time and transform what is currently a reactive health care system into a proactive one.

This thesis presents our work on initial steps towards a remote monitoring system for people with a chronic lung disease called Chronic Obstructive Pulmonary Disease (COPD). COPD is a debilitating and life-threatening disease characterized by restricted air flow in the respiratory system. Some of its symptoms include shortness of breath and chronic cough. The Government of Canada estimates that there were 2.3 million people with COPD in Canada in 2014 [32]. The World Health Organization estimates that there were 251 million cases globally in 2016, close to 3 million deaths per year due to COPD and that it will be the third leading cause of death worldwide by 2030[1]. One complication of COPD is what's known as an *acute exacerbation of COPD* (or exacerbation or AECOPD) where the disease significantly worsens. There is evidence that early treatment of exacerbations can reduce their severity and prevent hospitalization [108]. However, in order to begin treatment early, we need ways of detecting exacerbations early. Unfortunately, current attempts have been unsuccessful in part because they rely on *active* data collection, where participants have to manually record measurements, which introduces challenges with adherence [51]. Therefore, the work presented in this thesis focuses on using an easy-to-use, multi-purpose smartwatch to *passively* monitor people with COPD. In this context, passive sensing entails minimal effort from participants. For example, with a smartwatch, participants need only put on the smartwatch every morning and the smartwatch passively records data throughout the day.

A potential system for remote monitoring of COPD is shown in Figure 1.1. The key idea is that patients wear and use sensor-rich devices such as smartwatches. The data collected from the sensors on the devices contain signals that are relevant to COPD and if we can extract these signals and record them over time, we can establish a baseline for the patient. This opens the possibility of automatically detecting deviations from the baselines that are indicative of exacerbations. Detecting exacerbations early or ideally, predicting them ahead of time, can lead to earlier treatment and prevent hospitalization [108]. This full system requires many years of research and development and this thesis proposes only the initial steps towards this system. Specifically, it looks at collecting data from devices and extracting relevant signals from sensor data.

To make this passive sensor data relevant to COPD, we need to extract signals from the data that are relevant to COPD. The first signal we explore is respiratory rate and to that end, present WearBreathing. WearBreathing is a system for detecting respiratory rate, an important signal for those with COPD, using the accelerometer and gyroscope (collectively, IMU) sensor on smartwatches. While respiratory rate from IMU data is not a new idea, one limitation of all existing works is that they only work while participants are sitting very still in in-lab conditions. In order to monitor COPD patients as they go about their lives, we need a solution that is able to work in-the-wild. Therefore, we develop a system called WearBreathing that is designed to operate in an in-the-wild setting. Rather than trying to collect clean data, WearBreathing uses a machine learning model to predict the error if a respiratory rate

[1] `http://www.who.int/mediacentre/factsheets/fs315/en/`

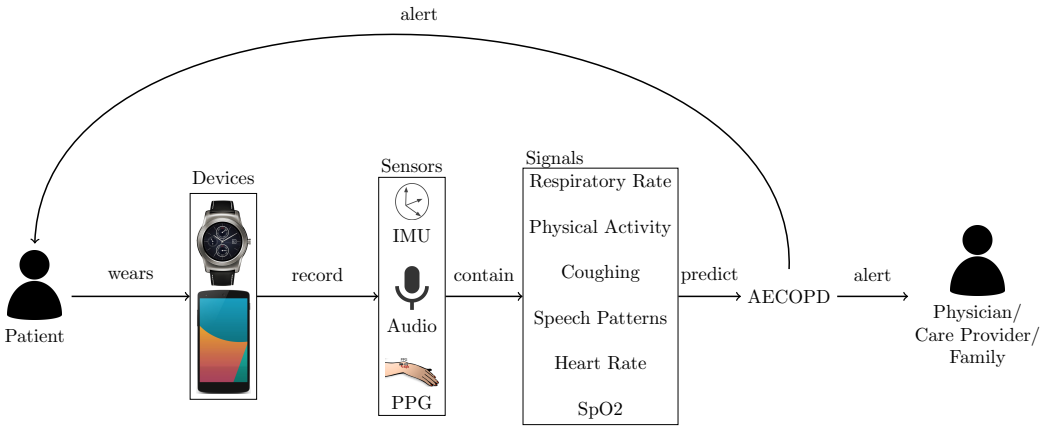


Figure 1.1: A potential system for remote monitoring of people with COPD.

measurement were taken from any given data. The result of this idea is a highly tune-able system that allows one to define an intuitive acceptable error threshold (e.g ±1 breath/min) and WearBreathing will record any respiratory rate measurements that it predicts are below that error threshold. We compare WearBreathing against existing methods and show that in the wild, WearBreathing has a 2.5 to 5.8 times lower mean absolute error than existing systems we compare against. Furthermore, we show that by using an energy saving technique called duty cycling, current smartwatches can run WearBreathing and still provide a full days worth of battery life.

The next signal we explore is coughing. Many existing methods of cough detection rely on audio data from a microphone. Therefore, first we examine audio data collected from a smartwatch microphone, characterize the types of sounds detected and the challenges of working with smartwatch audio. Next, we develop a system called CoughWatch for in-the-wild cough detection that achieves a 5.7 to 6.7 times higher F_1 score than current cough detectors. Our cough detection system also makes use of accelerometer and gyroscope data and we show that incorporating this additional data is able to improve precision by up to 15.5 percentage points compared to our audio-only cough detector.

Acknowledging the privacy concerns associated with recording audio, we also present a method for preserving audio privacy while retaining the ability to detect coughs. The key idea behind this approach is to use a speech obfuscation algorithm to render speech unintelligible and train a cough detection model on the obfuscated audio.

Finally, we deal with the battery constraints of mobile devices. Mobile devices such as smartphones and smartwatches are designed for intermittent use so when used to continuously collect and process sensor data, battery drain becomes a significant problem. Poor battery life not only creates inconvenience for patients resulting in poor adherence rates but also reduces the amount of data collected. The devices we used to collect our WearBreathing dataset could only record for roughly 6 hours before running out of battery. In our CoughWatch study, we relied on a duty cycling scheme where we recorded data for two minutes then paused recording for 8 minutes to preserve battery life. Duty cycling allowed us to record for a full day, however it also meant that we only recorded 20% of a full day. To deal with these battery concerns, we present Sidewinder, a heterogeneous architecture for continuous mobile sensing that is energy efficient and easy to program. Sidewinder proposes an interface between hardware

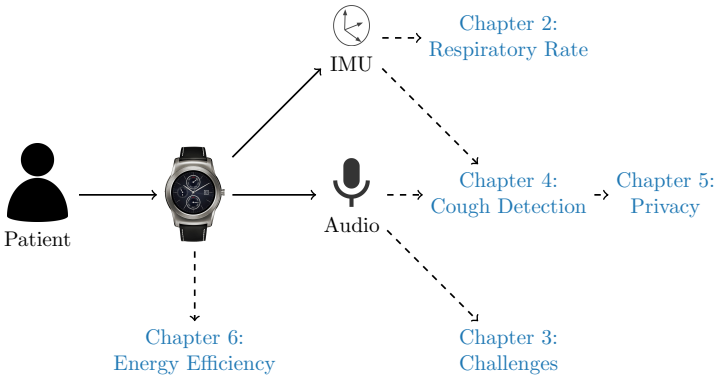


Figure 1.2: Thesis Overview

manufacturers, mobile operating systems and application developers that allows programmers to leverage a low-power co-processor to continuously monitor sensor data and wake-up the main processor only when an event of interest occurs.

Collectively, these works propose components of a practical remote patient monitoring system for people with COPD. The overall structure of this monitoring system and how each chapter of this thesis relates to it is shown in Figure 1.2.

1.1 Contributions

The contributions of this thesis can be summarized in the following points.

- A proposed system for remote COPD monitoring.
- A system for detecting respiratory rate from smartwatch IMU data that can operate in an in-the-wild setting.
- Exploration of the challenges of working with in-the-wild audio data.
- A state-of-the-art cough detection detection system based on smartwatch audio and IMU data that is designed for in-the-wild use.
- A demonstration of a privacy preserving method of cough detection from audio data.
- An energy efficient heterogeneous hardware architecture for continuous mobile sensing.

The rest of this thesis is organized as follows. Chapter 2 presents WearBreathing, our system for detecting respiratory rate from the accelerometer and gyroscope sensors available in off-the-shelf smartwatches. In Chapter 3, we discuss some of the challenges in working with audio data from a smartwatch and then in Chapter 4 we present CoughWatch, our system for in-the-wild cough detection from smartwatch audio and sensor data. Chapter 5 discusses some of the privacy concerns with audio recording and proposes a method of preserving privacy in an audio based cough detection system. In Chapter 6 we present Sidewinder, a hardware architecture and programming paradigm for continuous mobile sensing. Finally, Chapter 7 summarizes the thesis, explores future work in the COPD space and discusses how the components proposed in this thesis could be applicable to the monitoring of other health conditions.

Chapter 2

WearBreathing: Real World Respiratory Rate Monitoring using Smartwatches

2.1 Introduction

Continuous monitoring of physiological signals has the potential to revolutionize personalized health care. Respiratory rate is one signal that may be useful for a multitude of clinical applications. Higher respiratory rates are a strong predictor of cardiac arrest [23] and have been linked to negative outcomes in hospital wards and emergency rooms [15]. For example, Fieselmann *et al.* [23] reported that a respiratory rate over 27 breaths/minute was a significant predictor of cardiac arrest. In developing an early warning measure for cardiac arrest, Goldhill *et. al* [30] scored respiratory rate based on ranges of size 5 between 9 and 30 (e.g. $15-20$ or $21-25$), and reported that 21% of ward patients with a respiratory rate between 25 and 29 died in hospital. Cretikos *et al.* [15] recommend that patients with a respiratory rate greater than 24 breaths/minute should be monitored closely and those with respiratory rate greater than 27 should receive immediate medical review.

While these studies highlight the clinical importance of respiratory rate, the implications of continuous respiratory rate monitoring in uncontrolled settings have not been studied because there are no existing devices to facilitate such studies. Though devices such as chest bands that can be worn around the torso to measure respiratory rate have been available for quite a while, they are burdensome to use day after day and the effort of having to use an extra device will dissuade all but the most determined users. Monitoring respiratory rate outside labs and hospitals, on larger populations and over longer periods of time could lead to the development of new detection, monitoring and prediction systems for various conditions. These include not only respiratory conditions such as asthma and chronic obstructive pulmonary disease (COPD), but also non-respiratory conditions such as cardiac arrest, heart failure, panic attacks and anxiety disorders – all of which have shortness of breath as a potential symptom[1]. However, any research in the development of such systems must first devise a way to reliably detect respiratory rate in real world environments.

Our aim is to make respiratory rate monitoring as effortless as possible. This means we need inexpensive devices that are readily available, easy to use and contain sensors, as well as algorithms that can reliably measure the respiratory rate signal. Recent works [36, 38, 99] have identified smartwatches as a strong candidate for respiratory rate monitoring. Smartwatches are relatively inexpensive off-the-shelf devices that come with an assortment of sensors. They serve multiple purposes, making them more appealing for users to wear day after day. Respiratory rate monitoring will likely not even be the primary purpose of using such a smartwatch for most users, just an additional benefit. Furthermore, smartwatches are programmable, commercially-supported software platforms which means that turning a smartwatch into a respiratory rate monitor could be as simple as installing an application.

Existing works that use smartwatches for respiratory rate monitoring [36, 38, 99] make use of the accelerometer and/or gyroscope sensors. The underlying idea is that breathing produces subtle, periodic motions that can be measured by the Inertial Measurement Unit (IMU) in the smartwatch. The gist of these approaches is to use the periodic nature of breathing to detect a signal that falls within some frequency that corresponds to the expected breathing rate. However, the biggest challenge to these approaches is that they are highly susceptible to motion artifacts. Existing systems acknowledge this as a limitation and conduct their experiments in controlled or low-motion settings. Our goal is to make respiratory rate monitoring possible in everyday environments and during daily tasks.

Our key insight is that many applications benefit more from a small amount of accurate data rather

[1] https://www.mayoclinic.org/symptoms/shortness-of-breath/basics/causes/sym-20050890

than a large amount of inaccurate data. For example, an application that monitors the progression of a respiratory disease over time does not necessarily need to know the exact respiratory rate at any given second, but rather needs to keep track of trends over weeks or months. For such applications, sensors need only be accurate some of the time. The challenge then is being able to identify which readings are accurate.

Based on this insight, we develop WearBreathing, a two step system for respiratory rate monitoring. In the first step, a random forest (RF) model acts as a filter and rejects data that will result in an inaccurate respiratory rate reading. Sensor readings that pass the filter are then fed into the second step which uses a Convolutional Neural Network (CNN) model to extract respiratory rate from accelerometer and gyroscope data. Though, previous works have also used filters to reject some data [36, 38, 99], our random forest filter is fundamentally different. Previous filters are inflexible: they assume that excessive motion is the sole cause for inaccurate respiratory rate readings, and reject readings if the level of motion surpasses some threshold. Our approach does not make any assumptions about what causes unreliable readings and instead, learns what causes the extractor in the second step to be inaccurate.

This idea of a learned filter is the is the key difference between our work and previous works. In a more abstract sense, previous works only filter the source signal based on the amount of noise present. Our approach takes into account both the original signal and its interaction with the extraction algorithm. Our proposed approach, which includes a method for training the filter to learn the interaction between an extractor and the source signal, could be applied to other sensing tasks and potentially entirely different domains.

We collected data in a one hour long semi-controlled and a three hour long uncontrolled setting, from two groups of participants. By using data from two different groups of participants we highlight the generalizability of our approach. The first group consists of younger, healthy participants and the second of older participants suffering from a lung disease (chronic obstructive pulmonary disease, or COPD). We evaluate WearBreathing on our collected dataset and show that it is able to achieve a mean absolute error (MAE) of 2.05 breaths/min while delivering a respiratory rate reading every 50 seconds, which is a 3.6 times lower MAE than previous work [38, 99]. Moreover, unlike existing approaches, WearBreathing is highly tunable and can be easily configured to trade off reading frequency for accuracy. For example, WearBreathing can deliver a reading on average every 15 seconds with a MAE of 2.73, every minute with MAE 2.17 or every 5 minutes with MAE 1.09. This level of flexibility makes WearBreathing a good match for a wide range of applications. Some applications may require more readings and are willing to accept a lower accuracy while others may be willing to accept less frequent readings, but demand a higher accuracy. This tunability is possible because our random forest filter learns to directly control for accuracy. In previous work, the relationship between what the filter controls and accuracy is not intuitive, making this level of tunability difficult or impossible.

Using a combination of data traces and simulation, we explore the energy consumption of WearBreathing when deployed on a real smartwatch. We find that under ideal conditions, a duty cycling scheme can provide between 24-42 hours of battery life. Alternatively, applications that do not wish to use duty cycling can opt to use the smartwatch as a recording device and process the data offline, which will provide over 18 hours of battery life.

Our contributions are as follows:

- A demonstration that in in-the-wild settings, identifying when data is accurate is an important problem and that existing filters are not well suited to this task.

- A two-step filter/extractor system in which the filter learns the interaction between the extractor and input data.
- A highly tunable random forest filter that allows flexible trade off of frequency for accuracy data by changing a single, easy-to-understand threshold value.
- A novel method of respiratory rate extraction using a CNN that is more accurate than existing approaches.
- The combination of our random forest filter and CNN extractor which, for the first time, enables out-of-the lab respiratory rate monitoring using a smartwatch.
- An evaluation of out-of-the-lab respiratory rate monitoring on two different populations.
- An evaluation of WearBreathing showing that it can be run on a smartwatch with reasonable battery life.

The rest of this chapter is organized as follows. In Section 2.2 we describe our system for filtering data and extracting respiratory rate. In Section 2.3, we describe our dataset, existing methods and research questions. Next, in Section 2.4 we evaluate the performance of WearBreathing and existing methods. This is followed by a review of related works in Section 2.5 and a discussion of real-world deployment issues, avenues for future research and interesting overlaps with other research fields in Section 2.6.

2.2 System Design

As mentioned previously, not all data from the accelerometer and gyroscope will result in a reliable respiratory rate reading. The problem we solve is identifying when the respiratory rate reading is reliable. We do this by creating a system consisting of an extractor and a filter. The extractor takes sensor data as input and produces an estimated respiratory rate. The filter takes the same sensor data as input, but instead of predicting respiratory rate, it predicts the error the extractor will have on this input. To determine if a respiratory rate is produced, the predicted error from the filter is compared to a user defined threshold. If the predicted error is below the threshold, the sensor data is passed to the extractor which produces the estimated respiratory rate. This process is illustrated in Figure 2.1.

Although the filter comes before the extractor, because the filter predicts the error of the extractor and is therefore dependent on the extractor, it makes sense to describe our extractor first. This dependency also means that there are some subtleties in how we train our model, which we describe later in this section.

2.2.1 Extractor

We develop a novel method for extracting respiratory rate from accelerometer and gyroscope signals using a CNN. We employ a CNN model to extract respiratory rate because CNNs excel at detecting patterns in spatial or temporal sequences of multi-channel data and have been used extensively for time series data [37, 41, 112, 114]. While typically, recurrent networks such as LSTMs have been used for time series data, some work has shown that in some cases, simple CNNs can outperform LSTMs. Additionally,

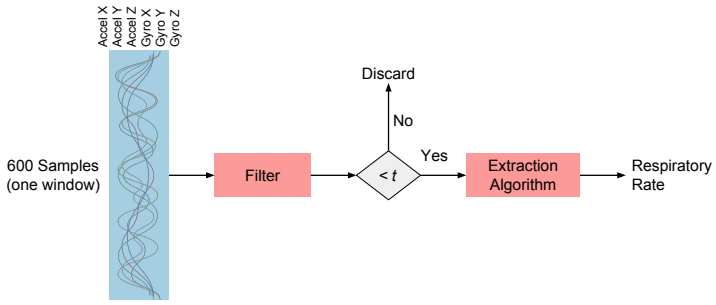


Figure 2.1: High level system diagram of WearBreathing.

CNNs tend to be less computationally expensive than recurrent networks[2], which is critical for a model designed to run on a resource constrained device such as a smartwatch.

In our case, we want the network to identify the breathing signal, which has a distinct pattern, in time series IMU data where the axes of the accelerometer and gyroscope are represented as channels.

The architecture for our CNN is shown in Figure 2.2. The CNN operates on 30 second windows of 6 axes of raw accelerometer and gyroscope data. We use 30 second windows because it is a commonly used window length in existing respiratory rate monitoring work [78, 99]. We also experimented with using the Fourier transform and derivative of each axis as input to the CNN, however, we found that using raw data performed better. We use a shallow CNN composed of a single convolutional layer with a rectified linear unit activation, 16 hidden units, a kernel size of 5 and stride size of 1. Following the convolutional layer is a max pooling layer with a pool-size of 10 and stride of 1. The output of the pooling layer is connected to a dense layers with 128 hidden units that feeds into another dense layer with 64 hidden units. Both dense layers use a rectified linear activation and a 0.2 dropout after each. The second dense layer connects to a single predictive node, again with a rectified linear activation. The network is optimized using *adaptive moment estimation* [49] (Adam), with mean absolute error as the loss function. The network is implemented using the Keras framework [13] and for any unspecified hyper-parameters the Keras default values were used. Treating this as a regression task, our network is trained to predict respiratory rate values. The respiratory rate labels used to train this CNN are obtained from a chest band. Our procedure for collecting labeled data is explained in more detail in Section 2.3.1.

2.2.2 Filter

The goal of the filter is to predict the error in the extractor. Previous works operated on the idea that respiratory rate extraction would be inaccurate in the presence of motion. Therefore, they employed simple filters that assumed the amount of motion is directly related to the error. An example of such a filter, is one that looks at the average vector magnitude of the x, y and z axes of the previous n accelerometer readings (Equation 2.1). For our experiments, we use a window size of 30 seconds and sampling rate of 20 Hz, which means that $n = 600$. So for a given window of 600 samples, if the value resulting from Equation 2.1 exceeds some predetermined threshold, t, then this window will not be passed on to the extractor. However, as we show later, simple filters such as these do not perform well

[2] https://github.com/baidu-research/DeepBench

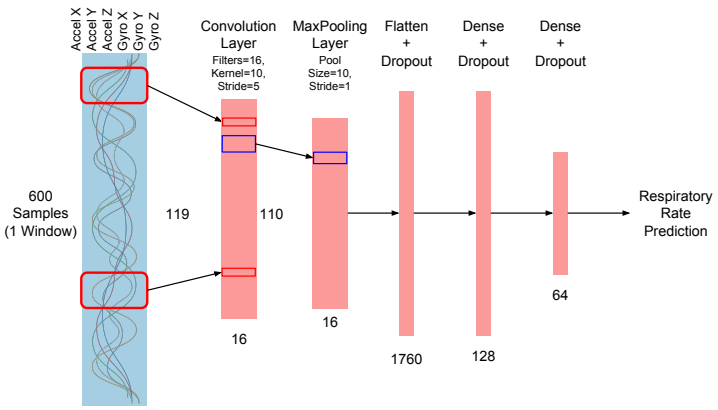


Figure 2.2: CNN extractor architecture showing how input data is transformed to produce the estimated respiratory rate.

and are not intuitive.

$$accept(window) = \frac{\sum_{i=1}^{n} \sqrt{x_i^2 + y_i^2 + z_i^2}}{n} < t \tag{2.1}$$

Our random forest regression-based filter is a novel approach to building these kinds of filters. Instead of assuming that motion is the only source of error and estimating the amount of motion using a formula, we train a classifier to learn what causes errors. This filter has the same interface as the simple filter in that it takes 600 readings as input from each axis of the 6-axis accelerometer/gyroscope data and outputs a single value that is then compared to a threshold. However, unlike the simple filter, which estimates the amount of motion in a window, our filter predicts the error we can expect if we were to apply our CNN extractor on a given window. As we show later, this results in a filter that is easier to tune since the threshold directly controls error in respiratory rates, which users care about and understand, rather than the motion within a window, which is harder to reason about.

Our random forest takes as input a summary of the accelerometer and gyroscope data. This summary is obtained by first computing 16 aggregate measures for each axis. These measures are the mean, median, minimum, maximum, kurtosis, skew and 10^{th}, 20^{th}, ..., 90^{th} percentiles. This reduces the dimension of each window from 6 vectors of length 600 to 6 vectors of length 16. The resulting 6 vectors are then concatenated into a single vector of length 96. Principal component analysis (PCA) is used to further reduce the vector's dimension to an empirically determined length of 20. The output of PCA is then used as input to a random forest regression model. The PCA projection matrix is computed using the training data only. Our general idea still works without PCA, however, we found that using PCA slightly improved our results.

For labels, we use the error of our CNN-based extractor. That is, for each input, we extract respiratory rate using our trained CNN and take the absolute difference between the predicted respiratory rate and the ground truth respiratory rate as the error. Our random forest model's task is to predict this error.

2.2.3 Training Scheme

Our random forest and CNN models both require training and testing. To prevent over-fitting and ensure that our results are generalizable, we use a leave- one-out cross validation scheme. That is, for

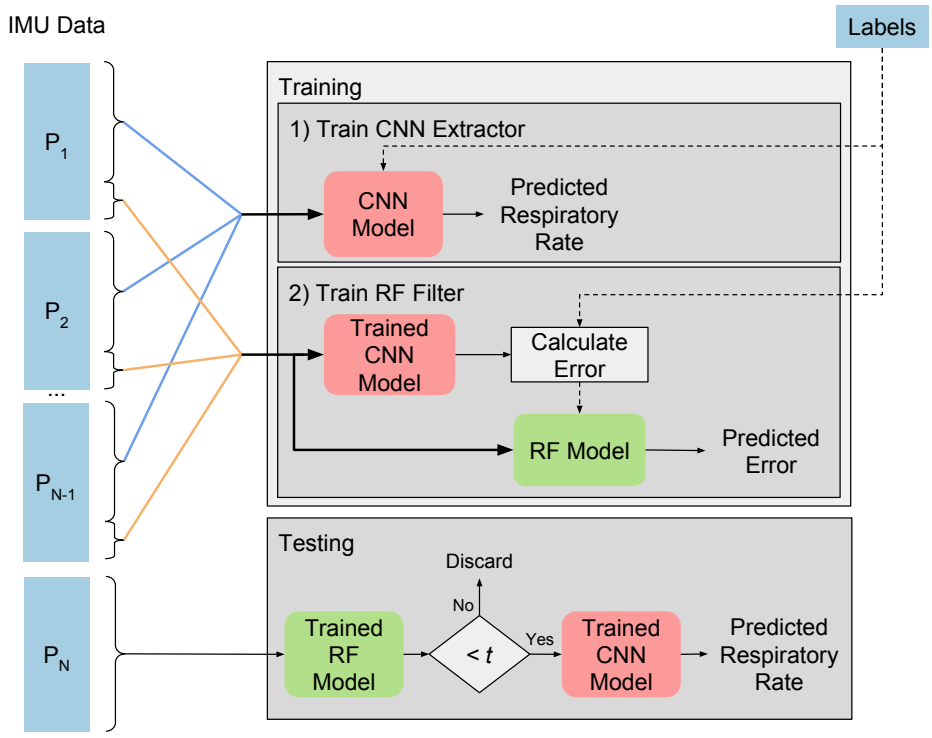


Figure 2.3: Training scheme illustrating how we use leave-one-out cross validation and separate CNN and RF training data to improve generalizability.

each participant x, we train a model using data from all participants except x. This means that when we evaluate a trained model on a participant, the model has never seen data from that participant during training. It also means that all the results we present are averages across participants.

We also have to be cautious about training the random forest regressor because its labels are dependent on output of the CNN and this could potentially bias the random forest. For example, if a window that's used to train the random forest was previously used to train the CNN, we would expect that the CNN is more accurate in extracting respiratory rate from this window. Therefore, the error that the random forest is attempting to predict is not a true representation of the error we would expect if the CNN was predicting on unseen data. To avoid this, we hold out a small amount of data (10%) from each participant in the CNN's training set. These data are not used to train the CNN. Once the CNN has been trained, the hold out data is passed through the CNN model and a respiratory rate is predicted. We compute the absolute error for these predictions and use these errors as labels to train the random forest filter. This scheme is illustrated in Figure 2.3.

2.3 Experimental Setup

We first describe our dataset and how it was collected. Next, we summarize two existing works that we compare WearBreathing against. Finally, we present the questions we ask to evaluate WearBreathing.

Figure 2.4: Zephyr BioHarness 3.0 used as a ground truth respiratory rate monitor.

2.3.1 Data Collection

To test and evaluate respiratory rate monitoring in the wild, we collect and make use of a dataset that contains data from a smartwatch and a ground truth device. Collection of this dataset was approved by the University Health Network Research Ethics Board (# 17-5704).

Our dataset contains data from 14 participants, 7 of which were healthy and 7 had chronic lung disease (3 female and 4 male in both groups, healthy group mean age: 28.4 years, chronic lung disease group mean age: 69.3 years). There is an age difference in these groups because COPD occurs most commonly in older adults. All participants were asked to wear an LG Urbane smartwatch on their non-dominant hand. The watch was running Android Wear and a data collection application we developed to collect accelerometer and gyroscope data. Data from both the accelerometer and gyroscope are recorded at 20 Hz because as shown by BioWatch [38], this is sufficient sampling frequency to capture the respiratory rate signal which has a frequency between 0.13 Hz and 0.66 Hz. This data is transmitted over Bluetooth to a smartphone. The smartphone simply acts as a relay and uploads the data to a remote server. All analysis and model training is done offline on the remote server. Although our system can support on device real-time analysis, the purpose of this data collection was to collect data that can be used to train our RF filter and CNN extractor. Later, in Section 2.4.3, we deploy our trained models on real devices to estimate WearBreathing's impact on battery life.

To obtain ground truth data, participants also wore a Zephyr BioHarness 3.0[3], shown in Figure 2.4, which uses a capacitive pressure sensor to measure expansion and contraction of the chest. The BioHarness has been validated in a several studies [34, 45, 46] to be accurate under the conditions we set in our data collection study. It has also been validated for participants with COPD [87]. For each respiratory rate reading from the BioHarness, we consider the preceding 30 seconds of accelerometer and gyroscope data as a window.

While wearing both the smartwatch and BioHarness, participants were asked to complete specific activities during the first part of the study which was semi-controlled and took place in a lab. We call this portion semi- controlled because while participants were asked to perform specific activities, there were no restrictions on how to perform the activities. For example, participants were not told how to place/move their arms. They had the freedom to move their arms however they wanted during the study.

The activities performed during the semi-controlled portion, listed in Table 2.1, are selected because we expect that they are the most frequently occurring activities in daily living. The six minute walk test (6MWT) was included because it represents the fastest participants are likely to walk in their daily lives and because it is a commonly used test for respiratory conditions [21]. Participants were allowed

[3]https://www.zephyranywhere.com

to take breaks between activities, so the total duration of the semi-controlled portion ranged from 40 to 75 minutes.

The second segment of the experiment was completely uncontrolled. Participants still wore the smartwatch and BioHarness, but were free to go about their day outside the lab. After three hours, the participants could take off the smartwatch and BioHarness.

During our data collection study, we collected over 53 hours of data from our 14 participants resulting in over 144,800 individual respiratory rate measurements from the BioHarness. The mean respiratory rate according to the BioHarness across our entire dataset is 18.31 breaths/min with a standard deviation of 4.72 and a range of 7-29 breaths/min. The range shows the 1^{st} and 99^{th} percentile of observed values. In Table 2.1, we break down the mean, standard deviation and range of the respiratory rate by group and activity.

Activity	Duration	Healthy		COPD	
		Mean (SD)	Range	Mean (SD)	Range
6MWT	6 min	20.9 (4.0)	14 - 29	24.9 (4.5)	13 - 34
Sitting	4 min	20.6 (5.2)	10 - 29	17.7 (3.0)	10 - 24
Walking	4 min	19.7 (4.6)	13 - 30	25.4 (5.4)	16 - 37
Lying	4 min	17.6 (2.5)	14 - 26	15.5 (5.7)	6 - 23
Standing	4 min	15.2 (5.0)	4 - 22	18.1 (4.0)	12 - 23
Eating	3 min	14.9 (2.7)	9 - 20	17.7 (3.7)	12 - 24
Brushing	2 min	16.0 (3.7)	9 - 25	18.5 (4.4)	11 - 24
Uncontrolled	3 hours	17.4 (4.2)	6 - 28	19.9 (4.9)	10 - 29

Table 2.1: Summary of our dataset showing duration of activities along with the mean and range (1^{st} and 99^{th} percentile) of respiratory rate (breaths/min) for each activity according to the ground truth BioHarness.

2.3.2 Existing Approaches

To compare WearBreathing, we implement the respiratory rate extraction methods explained in BioWatch [38] and SleepMonitor [99].

BioWatch [38] describes an extractor that first performs noise removal on gyroscope data by applying an averaging filter and then a band-pass Butterworth filter of order two with cut-off frequencies of 4 and 11Hz. Using this noise-removed data, their extractor obtains three respiratory rate predictions by computing an FFT on each of the three axis and selecting the frequency with the highest amplitude within 0.13 Hz and 0.66 Hz (i.e., their expected respiratory rate). To determine which of the three respiratory rate predictions to use, they again look at the FFT amplitude and select the one from the axis with the greatest amplitude. To validate our implementation of the BioWatch algorithm, we collected a small sample of data from two of the authors in the same method as described in the paper (i.e., sitting very still) and were able to achieve similar results (MAE < 1). Like BioWatch, our study includes sitting, standing and lying down, however, we give very little instruction to participants on how to perform these activities. Therefore, even when participants are sitting or lying down, they were not trying to be still and moved their arms quite frequently. This is evidenced by the fact that in the BioWatch paper, their filter, which takes the vector magnitude of the derivative of the accelerometer data, preserved 85.87% of windows when using a threshold 0.15 [38]. BioWatch used a threshold of 0.15

because it was the *maximum* value they observed from their filter during their in-lab study. However, when we apply that same filter on our dataset, we see an *average* value of 3.56. When BioWatch rejected windows where the filter value was above 0.15, they accepted 85.87% of windows. However, when we apply this filter and threshold on our data, we accept only 5.14% of windows. This highlights the drastic difference in the amount of motion between data collected out-of-the lab, as in our study, compared to data collected in-lab, as in previous studies and why methods developed in-lab may not necessarily work out of the lab.

We also replicated the algorithm described in SleepMonitor [99]. Their extractor uses a total variational filter (TV filter) [4, 43] to remove both high and low-frequency noise. After noise removal, respiratory rate estimation is performed on each accelerometer axis using an FFT in a manner similar to BioWatch. However, unlike BioWatch which selects respiratory rate from a single axis, SleepMonior merges the respiratory rate from all three axes using a Kalman filter. The filter to reject widows described in SleepMonitor [99] looks at the proportion of accelerometer readings in a window with a vector magnitude greater than $10m/s^2$. The threshold used in their paper is 5, so that any window where more than 5% of samples have a a vector magnitude greater than $10m/s^2$ is rejected.

2.3.3 Research Questions

The two metrics we consider in our analysis are accuracy (measured as MAE) and frequency (average time between readings). Using these metrics, we ask the following questions to evaluate WearBreathing:

- How does WearBreathing perform and compare to existing methods?
- How does activity affect performance?
- Does WearBreathing work on both the healthy group and the chronic lung disease group?
- Is there agreement between WearBreathing and the chest band ground truth?
- How tunable is WearBreathing?
- Is it feasible to run WearBreathing on a smartwatch?
- What is the battery impact of running WearBreathing on a smartwatch?

2.4 Evaluation

In this section, we present an evaluation of WearBreathing on a diverse dataset that includes data from younger, healthy individuals and those with COPD while performing a wide range of activities. Having this diverse dataset allows us to analyze how the participant's activity and group affects WearBreathing's performance. Throughout the entire study, participants wore a smartwatch and a chest band for ground truth data.

In addition to accuracy, we explore the feasibility of running WearBreathing in real-time on a smartwatch. Using a combination of data traces and simulation, we show that with duty cycling, WearBreathing can be run in real-time on a modern smartwatch while providing enough battery life to last a full day.

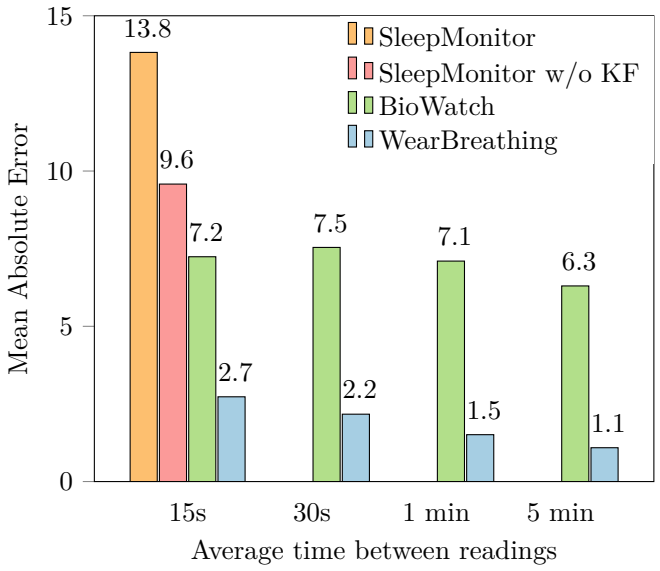


Figure 2.5: Mean absolute error for WearBreathing and existing methods a filter is used to provide readings at various frequencies. SleepMonitor only shown for 15s because there was no threshold that resulted in readings at other intervals.

The rest of this section is organized as follows. First, in Section 2.4.1, we select a few fixed threshold values and compare the performance of WearBreathing against the two existing methods. In Section 2.4.2, we analyze tunability and characterize how WearBreathing and existing methods perform across their entire threshold domain. Finally, in Section 2.4.3 we deploy WearBreathing on a smartwatch and analyze runtime and battery life impact.

2.4.1 System Performance

The performance characteristics of WearBreathing are heavily dependent on the threshold applied to the random forest filter. By using lower threshold, we are able to increase the accuracy of readings but we receive data less frequently. We highlight this in Figure 2.5 by setting threshold values to produce readings on average every 15s, 30s, 1 min and 5 mins and showing the mean absolute error at these different frequencies for WearBreathing, BioWatch and SleepMonitor.

We observe that SleepMonitor has a very high error. We suspect this is because of how the Kalman filter described in SleepMonitor relies on exploiting historical readings to boost predictive accuracy. By basing the predicted respiratory rate ($rr_{t|t}$ in the SleepMonitor paper [99]) as a limited change from the posterior/prior respiratory rate in the previous time step ($rr_{t|t-1}$), the Kalman filter is able to reduce the random noise caused by sudden movements. This, however, is predicated upon the assumption that readings are coming in at consistent time intervals. However, this assumption does not hold "in-the-wild" when a filter is used to discard windows because windows are no longer occurring at consistent time intervals. To validate this, we modified the SleepMonitor algorithm to combine the respiratory rate prediction from each axis by taking a simple average instead of using a Kalman filter (SleepMonitor w/o KF in Figure 2.5) and see that the modified algorithm does indeed perform better on noisy data.

For SleepMonitor, we also observe that there was no threshold we could set that would result in a reading on average every 30 seconds, 1 min or 5 mins. For BioWatch, although we could tune the

frequency at which we receive data, we did not see any improvement in the MAE as we decreased the frequency of readings. WearBreathing, on the other hand, in addition to having a dramatically lower MAE, also allows trading off frequency for accuracy.

WearBreathing has a substantially lower MAE for all frequencies. When no filter is used, WearBreathing has a MAE of 2.86 compared to BioWatch's 7.01 and SleepMonitor's 9.86, which is a 2.5 and 3.4 times improvement, respectively. When providing a reading every $15s$, we see a 2.6 and 3.5 times improvement (2.73 MAE for WearBreathing, 7.24 for BioWatch and 9.58 for SleepMonitor). At a reading every 5 minutes, WearBreathing has a 5.8 times lower MAE than BioWatch (1.09 vs. 6.30)

This highlights two points. First, the CNN extractor in WearBreathing by itself has a lower MAE than any existing system. Second, despite the CNN already having a lower MAE, by applying our random forest filter, we can further lower the MAE if we decrease reading frequency. Having this trade-off available makes WearBreathing more applicable to a wider range of applications because now applications can decide whether or not the trade-off is worthwhile based on their requirements.

Conclusion: WearBreathing performs substantially better than existing methods. Using thresholds that result in similar times between readings, WearBreathing's MAE is between 2.5 and 5.8 times lower.

Performance by Activity

As we have shown, the threshold used with our random forest filter greatly affects performance. For fair comparisons, we want to control the frequency at which the three systems provide readings and compare their accuracy. BioWatch's default threshold value of 0.15, provides a reading on average every 50 seconds with a MAE of 7.49. If we set a threshold of 1.05 with WearBreathing, we can achieve the same frequency and a MAE of 2.05 (3.7 times lower). Because SleepMonitor cannot provide a reading on average every 50 seconds, we do not include it in the remaining analysis in Section 2.4.1. Later on, in Section 2.4.2, we perform a sensitivity analysis to characterize how all three systems perform across their entire threshold domain.

Using these thresholds, we analyze the MAE and frequency (% of windows accepted) for both WearBreathing and BioWatch during the various activities in our data collection. The results are presented in Table 2.2. Regardless of activity, WearBreathing has a lower MAE than BioWatch. In some cases, BioWatch does accept more windows than WearBreathing. For example, while standing, BioWatch accepts on average 17.2% of windows compared to WearBreathing, which accepts 7.7% windows. However this increased frequency comes with a much higher MAE (8.2 compared to 1.6). There are also multiple examples of BioWatch not producing any readings during an activity (ex. COPD patients during 6MWT) or only producing readings for one participant (indicted by a missing standard error), but WearBreathing is able to produce readings for multiple participants for all activities.

Conclusion: WearBreathing is able to produce accurate readings across all activities while previous works tend to not generate readings during activities involving more motion.

Performance by Group

Table 2.2 also shows that BioWatch's performance on the COPD group is lower than on healthy participants. This highlights an important point that is generally well known but worth emphasizing: that systems developed on one population may not generalize to other populations. BioWatch was developed and tested on younger participants with no respiratory conditions, and we see that it works better on

Activity	BioWatch				WearBreathing			
	Healthy		COPD		Healthy		COPD	
	MAE	Freq	MAE	Freq	MAE	Freq	MAE	Freq
6MWT	7.3 (-)	4.5	- (-)	-	0.8 (0.3)	10.0	0.6 (0.3)	11.3
Sitting	5.3 (1.3)	6.8	8.9 (-)	13.8	1.9 (1.5)	16.5	1.2 (0.4)	10.0
Walking	7.5 (-)	27.5	- (-)	-	1.6 (0.8)	20.0	0.7 (0.2)	16.0
Lying	6.8 (0.8)	26.7	12.4 (2.2)	30.3	1.3 (0.6)	7.0	1.3 (0.7)	14.8
Standing	8.2 (3.3)	17.2	7.6 (2.5)	63.5	1.6 (0.7)	7.7	1.3 (0.8)	12.8
Eating	3.2 (1.4)	22.8	7.7 (-)	26.3	1.4 (0.5)	9.2	0.6 (0.2)	17.2
Brushing	5.7 (4.9)	8.5	- (-)	-	2.1 (1.0)	5.2	1.6 (0.6)	18.3
Uncontrolled	6.4 (0.9)	6.3	7.7 (1.2)	15.7	1.5 (0.5)	4.0	2.9 (0.8)	3.0

Table 2.2: Mean absolute error (SE in brackets) and frequency (% of windows accepted) for BioWatch and WearBreathing broken down by activity and group.

healthy participants than on those with COPD. Applying BioWatch to COPD participants without validating performance would have lead to higher than expected errors. If we wanted to monitor participants with COPD for a study, we would have to be careful and ensure that all the devices and algorithms we use have been validated for this population.

One interesting result we see in Table 2.2 is that using the BioWatch method, respiratory rate is particularly inaccurate (MAE 12.44) for participants with chronic lung disease when they are lying down. This may be because people with COPD have increased airway obstruction and reduced lung volume while lying down [19]. However, because the filter used by BioWatch summarizes the amount of motion in a window, it accepts quite a few windows while patients are lying down, which leads to large errors. Our system on the other hand, is trained on participants with chronic lung disease and is better able to learn and recognize their breathing patterns.

Conclusion: Because WearBreathing is trained using data from healthy participants and participants with COPD, it performs well on both these groups.

Agreement with Ground Truth

To measure agreement between WearBreathing and the ground truth BioHarness, we use a Bland-Altman plot [7]. A Bland-Altman plot helps analyze agreement between two measurement methods. For each pair of measurements, it shows the mean of the two against the difference of the two. This is a useful tool to visualize fixed bias (i.e., a non-zero mean difference), proportional bias (i.e., non-zero slope for the line of best fit) and limits of agreement (i.e., between which two y-values do 95% of data points lie).

The Bland-Altman plot for both the healthy group and the COPD group is shown in Figure 2.6. For clarity, these graphs only plot a small random sample of data points, however the mean error, limits of agreement and best fit line were computed using all data points. The points are plotted with transparency which means darker regions represent a higher density of points.

For the healthy group, we see a fixed bias of 0.19 breaths per minute, proportional bias of 0.02 and limits of agreement between -3.21 and 3.60. This is a fairly low although significant (one sample t-test $p \approx 0$) fixed bias and negligible proportional bias. If desired, the fixed bias can be removed by subtracting 0.19 from all our predicted values. The negligible proportional bias suggests that the error in our prediction is not correlated with the magnitude of the predicted value. Finally, the limits of

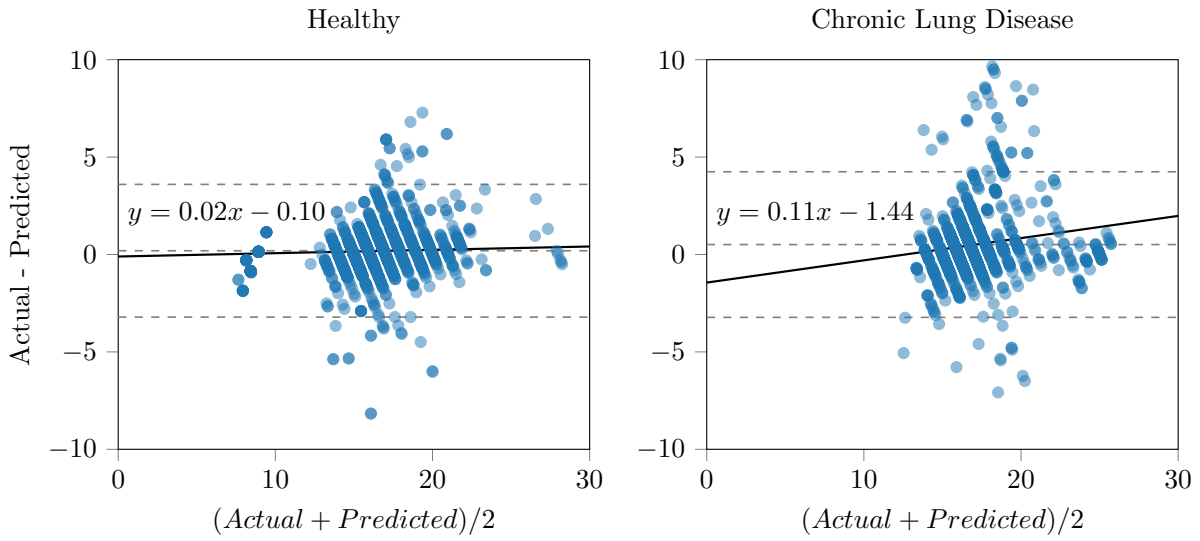


Figure 2.6: Bland Altman plot showing agreement between the BioHarness and WearBreathing for both healthy participants and participants with chronic lung disease. For clarity, only a random sample of data points is shown.

agreement suggest that 95% of our predicted respiratory rates will be within −3.21 and 3.60 of the BioHarness readings.

For the chronic lung disease group, we see a slightly higher fixed and proportional bias of 0.51 and 0.11, respectively, and limits of agreement between −3.22 and 4.27. While, there is slightly less agreement in the chronic lung disease group than the healthy group, both do still show strong agreement.

Conclusion: There is strong agreement between WearBreathing and our ground-truth BioHarness data.

2.4.2 Tunability

In earlier sections, we selected a few threshold values and presented results for those threshold values. In this section, we explore how the selected threshold effects performance and expand on the black box aspect of our random forest filter. While our random forest filter works really well in conjunction with our CNN, because it treats the extractor as a black box, it can also be used with existing respiratory rate extraction methods. By training the random forest to predict the error of BioWatch and SleepMonitor we are able to use our random forest filter to improve the performance of these existing methods. In showing that our random forest can be used with existing methods, we highlight that our idea of trading off frequency for accuracy by learning when error will be high is not limited to just our CNN but can be used with other extractors and potentially tasks other than respiratory rate monitoring.

There is a trade-off in selecting a filter threshold. A lower threshold may result in a lower MAE, but at the cost of the number of readings. In Figure 2.7, we show how the proportion of windows discarded by the filters change as a function of threshold. We observe that with the SleepMonitor filter, even with a threshold of 0, it does not discard 100% of the windows. This is because in each window in our data there are always accelerometer readings where there are no forces exerted on the smartwatch besides gravity and therefore the vector magnitude is close to $9.8m/s^2$. This highlights a useful feature that

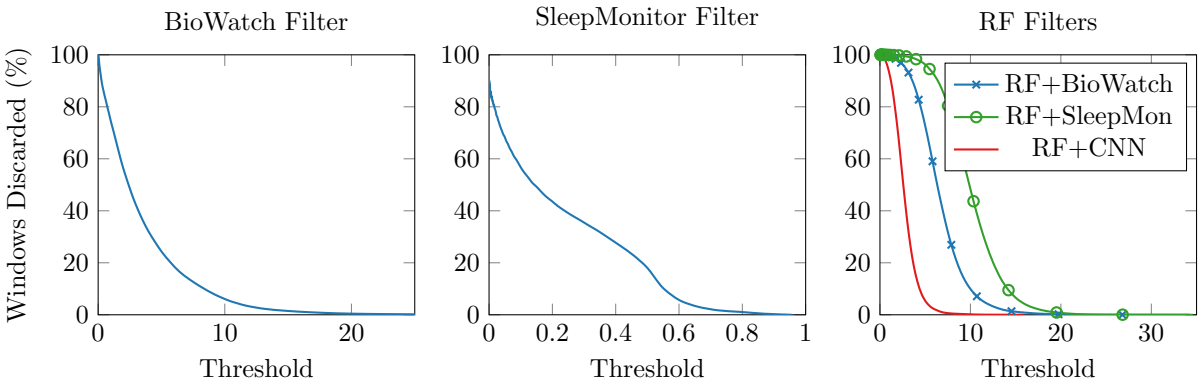


Figure 2.7: Proportion of windows discarded by various filters as a function of threshold. RF filter shows three curves depending on which extractor the RF was trained on.

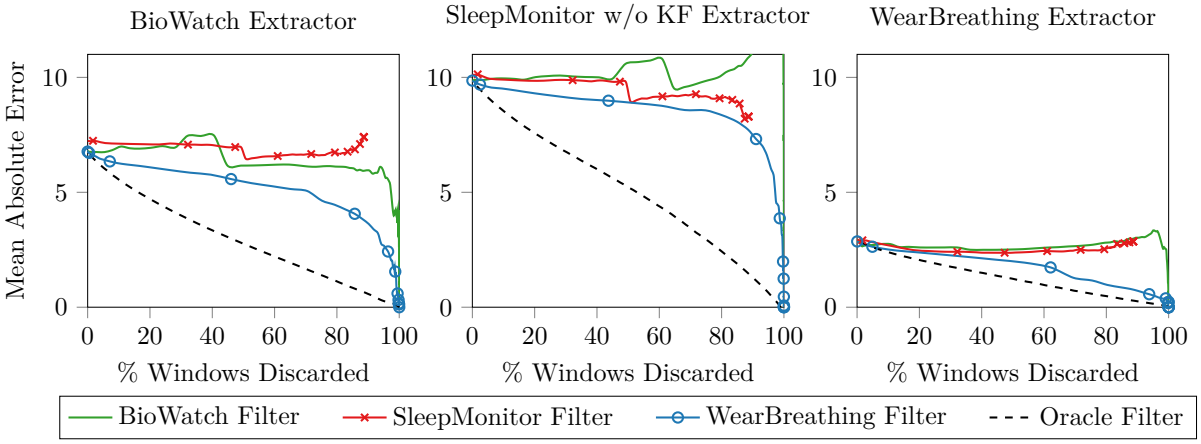


Figure 2.8: Mean absolute error vs proportion of windows discarded by filters for different extractors.

filters should have, which is that they should allow selecting as much or as little data as desired.

Next, we evaluate how the threshold affects the respiratory rate extraction error in an all-vs-all comparison where we apply each filter to each extractor. We vary the threshold for each filter and monitor how that affects the number of windows discarded and the mean absolute error when passing the accepted windows to the extractor. The results for this analysis are shown in Figure 2.8. We see that simple filters are not smooth functions, which is not a desirable characteristic for tunability since they make it hard to predict how small changes affect accuracy. Our hypothesis for the spikes seen in these filter's curves is that the value these filters are computing is not well-distributed across the threshold domain and there is no direct link between the threshold and filter value. This makes it so that a small change in the filter threshold does not necessarily result in a small change to the number of windows accepted by the filter. Combining our two observations so far, we argue that a good filter should be a smooth function that is capable of discarding anywhere from 0% to 100% of the windows.

Our random forest filter meets both these requirements. When used in conjunction with our CNN, it is much better behaved in that a small increase to the threshold results in a slight increase in number of windows accepted and MAE. This linear property makes the filter a good "turn dial" solution that can be

tweaked to the requirement of individual applications. For example, if one wanted to detect respiratory rate with a MAE of 2, they could set the threshold to 1, which would result in a reading roughly every 50 seconds. If one wanted higher accuracy, you could set the threshold to 0.5 which will give an MAE of 1.09 and a reading roughly every 5 minutes. Although the RF predicts error, the threshold value used with the RF filter is simply a value that can be adjusted and not a *guarantee* of accuracy. This is because the RF model itself has a prediction error. Our RF filter had a 1.20 MAE in predicting the absolute error in our CNN extractor. We also see that while the random forest filter also works for the BioWatch and SleepMonitor extractors, performance is not as good as with the CNN. The random forest filter has a MAE of 3.48 for the BioWatch extractor and 4.57 for the SleepMonitor extractor.

We also introduce the idea of an *oracle filter*. If the filters goal is to predict the error in an extractor, the oracle is able to do so with 100% accuracy. Additionally, the oracle knows the extractors error on all past, present and future windows of data. Therefore, if we use a threshold of 20 with the oracle filter, it only accepts a window if the extractors error on this window is among the lowest 20% of all errors. As we can see in Figure 2.8, none of the filters perform close to the oracle when applied to the BioWatch or SleepMonitor extractors. This is because both the extraction and filter have some error which compounds to make a poorly performing system. However, for our CNN which has a relatively low error, the random forest filter is much closer to the oracle filter. This again suggests that our RF filter is learning what makes the CNN perform well on a window.

Conclusion: WearBreathing is highly tunable because its threshold value is intuitive and its filter is a smooth function that is capable of rejecting any given proportion of windows.

2.4.3 Battery Life

Battery life is a critical consideration for any mobile system. After collecting data from participants and training our models, we explore the implication on battery life of running WearBreathing on an actual smartwatch. To do so, we use a combination of experiments and simulation. Our simulation makes use of the IMU data collected from our 14 participants as traces to simulate real-world battery life.

In addition to the IMU traces, we need battery consumption measurements under different modes of operation. These modes are idle, recording IMU, running RF and running CNN. When the watch is idle, its CPU is in a sleep state so no monitoring or processing is occurring. During the RF mode, a wake lock is held to prevent the watch from enter a sleep state and IMU data is collected. The RF mode collects IMU data and runs our random forest filter. Finally, the CNN mode collects sensor data, runs the RF filter but does not use the RF output to determine whether or not the CNN is run. Instead, the both the RF and CNN are run on all windows. For the RF and CNN states, we also need measurements on how long the RF and CNN take to execute on a window of data.

Battery Data Collection

To run our CNN and RF on the smartwatch, we first convert our trained model to TensorFlow Lite[4](version 1.10.0). The converted model can be loaded on a smartwatch and used by a TensorFlowLite interpreter that we call from our Java app. To run the RF model, we transpile it directly into Java code using sklearn-porter[5]. While the CNN operates on raw sensor data, the RF model requires features extracted from the data. This feature extraction is also done in Java code.

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

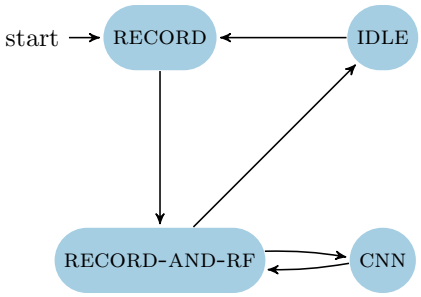


Figure 2.9: Battery life simulation state machine.

We configure our app to run these models according to the different modes of operation and intermittently record battery levels. We charge six LG Urbane watches, the same ones we used with our participants, to over 90% battery and run our application with the screen off until the battery completely drains. We run each mode of operation on all six watches, and using the initial and final battery level, we calculate the mean and standard deviation of change in battery level per hour. The results for this are shown in 2.3.

Mode	Battery Δ (%/hour)	
	Mean	SD
Idle	-1.67	2.71
Continuous IMU	-5.42	0.26
Continuous IMU + RF	-22.52	1.47
Continuous IMU + RF + CNN	-22.84	1.42

Table 2.3: Battery % change per hour during different modes of operation.

To measure execution time, we use one watch and run our application for 10 minutes. During these 10 minutes, we time how long it takes to run the RF on 100 windows and divide the result by 100. This helps minimize the overhead of our timing functions. The same procedure is repeated for the CNN. We find that the RF takes on average 26.54ms (± 1.69ms) per window and the CNN takes 32.47ms (± 0.68ms) per window. With a sampling rate of 20Hz we obtain a reading every 50ms, meaning our processing should run in under 50ms to run in real-time. While running both the RF and CNN on all windows would not meet the 50ms requirement, WearBreathing requires running only the RF on all windows. The CNN is run on windows where the RF output is below the threshold. With a threshold of 1.05, less than 20% of windows are accepted by the filter. Therefore, the CNN is run infrequently enough that WearBreathing is able to run in real-time.

Simulation Setup

We use the IMU traces described in Section 2.3.1, energy consumption measurements and runtime measurements to simulate battery life of WearBreathing. Our simulator, which is written in Python, implements the state machine shown in Figure 2.9. It starts at $time = 0$ seconds with battery level at 100% and in the RECORD state. While in the RECORD state, the simulator steps through the IMU traces for 30 seconds until the data window fills up. Once the window is full, it enters the RECORD-AND-RF

[5]https://github.com/nok/sklearn-porter

state. In this state, the RF is run and if the RF output value is below a specified threshold, the CNN is also executed (CNN state).

The time spent in each state is dependant on the simulation conditions (i.e. RF threshold and duty cycling scheme) and measured values such as the RF and CNN execution times. In our simulation, we randomly sample the execution time for each RF and CNN run from a normal distribution parameterized on our measured mean and standard deviation. Using the time spent in each state ($duration_s$), we update the battery level according to Equation 2.2 where $battery\_consumption_s$ is also drawn from a normal distribution based on the mean and standard deviation of the corresponding mode from 2.3. The simulator steps through this state machine until the battery level drops to 0% at which point, the time is recorded and the current simulation run ends.

$$battery\_level_{new} = battery\_level_{prev} + duration_s * battery\_consumption_s \tag{2.2}$$

We also simulate the effect of duty cycling. With duty cycling, data is recorded and processed for some period of time and then the watch is allowed to sleep for some period of time. While this reduces the amount of data collected, it can be useful way to save energy. We simulate two types of duty cycling; *fixed duty cycle* and *adaptive duty cycle*. In both cases, our simulator cycles between IDLE, RECORD and RECORD-AND-RF, but difference lies in when these transitions occur. In fixed duty cycling, transitions occur periodically. For example, with an 8min/2min duty cycle, the simulator spends 8 minutes in the IDLE state and then transitions to RECORD. The RECORD and RECORD-AND-RF states then collectively run for 2 minutes. In adaptive duty cycling, the IDLE state runs for a fixed period of time, however the RECORD-AND-RF state only runs until n windows where the RF output is below the threshold have occurred.

Another factor to consider with duty cycling is that according to Liaqat et al. [59], transitions between sleep and wake states consume additional power. For example, they find that the sleep to wake transition, which lasts 1 second, consumes 19% more power than the awake state. The wake to sleep transition, also lasting 1 second, consumes 5% more power. In our simulation, we very pessimistically assume that each transition takes two seconds and consumes twice as much power as the continuous IMU mode.

Table 2.4 shows the simulated battery life of the smartwatch under various recording conditions. To validate our simulator, we configure it to perform the modes of operation listed in Table 2.3, for which we have experimental data. Simulating a continuous IMU recording (simulator always in the record state) yields an expected 18.5 hours of battery life. This is slightly higher than our experimental observation (approximately 17 hours). The small difference in these battery lives is because in our experiments, we charged the devices to anywhere from 90% to 100%, whereas our simulator always begins with 100% battery life. The other two modes produce similar results.

WearBreathing Battery Life

Simulating WearBreathing with data traces from our 14 participants, we estimate that running WearBreathing continuously would result in just over 5 hours of battery life, which is expected because running the RF continuously results in roughly 5.5 hours of battery life and WearBreathing is equivalent to always running the RF and occasionally running the CNN. While five hours of battery life would not be acceptable in a real deployment, using a fixed duty cycle (2 minutes record, then 8 minutes idle),

our simulation estimates a battery life of over 21 hours. Using an adaptive duty cycle, where the device sleeps for 8 minutes, and then wakes up until it obtains three reliable respiratory rate readings, we obtain an estimated battery life of over 42 hours.

While duty cycling allows a full day's worth of battery life, it comes at the cost of missing data while the watch is in a sleep state. Depending on the application, this may or may not be an acceptable trade-off. For applications that need continuous recording of data, there is the option of collecting sensor data continuously, but delaying processing the data. The watch could record data while being worn by a user but wait until being charged to process the data or upload it to a server for processing. This would allow continuous recording, but the measured respiratory rate would not be available immediately. The energy consumption of this offline processing scheme is the same as the Continuous IMU condition shown in 2.4 (17 actual, 18.5 hours simulated), and is also enough to last a full day.

Condition	Actual		Simulated	
	Mean	SD	Mean	SD
Continuous IMU	16h 54min	1h 35min	18h 30min	11min
Continuous IMU + run RF	4h 17min	12min	5h 28min	2min
Continuous IMU + run RF and CNN	3h 50min	26min	3h 57min	2min
WearBreathing Continuous			5h 18min	3min
WearBreathing DC (2min/8min)			21h 30min	25min
WearBreathing Adaptive DC ($n = 3$)			1 day 20h 19min	56min

Table 2.4: Battery life of a smartwatch under various conditions. First three conditions were both experimentally run and simulated and used to test the accuracy of the simulation. Bottom three conditions show WearBreathing under different recording conditions.

One limitation to our battery life analysis is that the battery measurements used in our simulator were obtained while the smartwatch was still and the screen off. Under normal usage, when the "always-on screen" is disabled, a wrist gesture or screen press can be used to turn on the display temporarily. However in our experiments, the screen was almost never turned on. According to Liu et al. [66], the screen is the most power hungry component of a smartwatch. Therefore, we expect that our simulated battery life is overestimated. However, even if actual battery life is half our predicted values, using either a fixed or adaptive duty cycle should be enough to last a full day's worth of recording.

Conclusion: Full day battery life is possible with WearBreathing using either duty cycling or offline analysis.

2.5 Related Work

The key difference between WearBreathing and existing works is that both our filter and respiratory rate extraction is automatically learned from data. This is beneficial to our filter because we remove any assumptions about what makes data unreliable. So for example, we don't assume that motion makes data unreliable and therefore try to estimate motion. As we have shown, this results in a better behaved, more intuitive filter. Similarly, for the extractor, we do not manually look for a specific sinusoidal pattern in the signal and instead, let the CNN learn this pattern. As demonstrated by the CNN being able to generalize across participants and participant groups, it is able to learn more complex patterns which accurately predict respiratory rate.

Another recent paper, MindfulWatch [36], uses accelerometer and gyroscope data to estimate respiratory rate during meditation. Similar to SleepMonitor, MindfulWatch builds a historical model of respiratory rates. However, since MindfulWatch is designed for meditation where participants hold certain postures for periods of times, they monitor for posture changes and reinitialize a model for each new posture. These ideas of building a historical model of respiratory rate may be applicable to our system, especially when lower threshold values are used, however, we would need further work to understand how well they work.

While we used the accelerometer and gyroscope on a smartwatch, other works have looked at detecting respiratory rate from a pulse oximeter collected photoplethysmogram (PPG) [9, 14, 17, 44]. However, these techniques are also susceptible to motion artifacts [68]. PPG also has the downside of being affected by skin tone [20, 56] and conditions such as anemia [91]. Additionally, while most smartwatches use a pulse oximeter for heart rate monitoring, very few provide access to the raw PPG data and as of Android API version 28, the Android sensor manager does not have a constant for PPG sensors[6]. While most studies examining pulse oximetry use data collected from a fingertip, the forehead or an earlobe, with the current interest in smartwatches, there is a push to bring pulse oximetry to the wrist. For example, the first wrist based pulse oximeter was approved by the FDA in 2018 [33]. As wrist based pulse oximetry becomes more developed, respiratory rate from wrist-based PPG data may also be a viable option. This could open doors to sensor fusion techniques that use both IMU and PPG data for even more accurate respiratory rate monitoring.

2.6 Discussion

The idea of filtering data by throwing out sections that are unreliable is not new. However, we argue that in in-the-wild environments this becomes a much more significant and challenging problem because there are no guarantees about what's happening in the environment and there are many assumptions embedded within systems. For example, although we tried to make our data collection study as close to real-world as possible by placing very few constraints on participants, we did operate under the assumption that the participants were wearing smartwatch. In the real-world, users are likely not wearing their smartwatch for large parts of the day (ex. while sleeping or relaxing at home). In our experience, detecting when the watch is not being worn is not trivial. Applying any sort of detection algorithm to sensor data while the watch is not being worn can result in unusual results. For example, we have observed that even when a smartwatch is not being worn, the heart rate sensor still produces valid heart rate readings. Similarly, BioWatch takes the most periodic signal within a frequency range, so it is possible that it will produce respiratory rate readings even if the watch is not worn. If we wanted to deploy WearBreathing in a real-world experiment, we would have to analyze how it behaves in these kinds of scenarios. Ideally, the filter would reject data from when the watch is not being worn. In our current work, the random forest was only trained with data where the watch was worn so we do not know how it will behave in a real-world deployment. However, this is not an inherent limitation of our result and can be solved with more training data or by implementing another filter rejects data from when the watch is not worn.

In our analysis, we accept all windows where the value generated by the filter is below some threshold. Because the goal of the filters is to select windows that will produce a high accuracy reading, accepting all values below a threshold is in a way, controlling for accuracy. If we set a lower threshold, we are

[6] https://developer.android.com/reference/android/hardware/Sensor

requesting a higher accuracy. While empirically, we found that 90% of readings occur within 90 seconds of each other, there is no guarantee of how long until a reading is produced (although not receiving readings may itself be a useful signal). Stochastically receiving readings may be acceptable for some applications, however, other applications may prefer to have consistent, periodic readings. WearBreathing is also able to support these applications. If an application wants a reading every minute, it could buffer filter and extractor outputs for the last minute, and select the best window(s) from the buffer when a reading had to be produced. In this use case, applications also have the freedom to determine how they choose the best window(s). They could, for example, choose the window with the minimum filter value or take an average of the 10% of windows with the lowest filter value.

Furthermore, while we are able to achieve accurate results using a single threshold value for all participants, it may be possible to further improve accuracy by selecting custom thresholds for each user. However, this would require collecting ground truth data from each participant in order to find the optimal threshold. For example, for a study with a small number participants, the participant on-boarding process could require wearing a ground truth device and a smartwatch for a short period. Then, the filter threshold applied to this users data is chosen based on the timing/accuracy requirements of the study. However, such an approach would not be suitable for larger studies or crowd-sourced data.

We would also like to point out the importance of validating algorithms on target populations. A system for respiratory rate monitoring is more likely to be useful for people who have some lung disease. However, as we show, there can be a difference in accuracy on participants with chronic lung disease and healthy participants. The essence of the issue is that the performance on our test data may not match performance in our actual deployment. One way around this would be to collect a small amount of ground truth data from some or all participants in the real deployment. For example, in a 3 month deployment of 20 participants, it may be feasible to randomly select 5 participants to wear the BioHarness for a few hours. This would give an estimate of how the system is performing in the deployment. While this is extra work and has overhead, we believe a scheme like this is akin to insurance. It is worth paying this overhead and having an idea of the systems accuracy rather than ending up with hard acquired and expensive data with little understanding of the accuracy.

Finally, our proposed system makes use of smartwatches, which have limited battery and processing capabilities. We have shown that with duty cycling or offline processing, our approach can run on a smartwatch and provide a full day's worth of battery life. However, offloading approaches may allow real-time processing with very little overhead and without the need for duty cycling by running the random forest filter on an auxiliary low-power processor and waking up the main CPU to run the CNN when a good window is detected.

An example of an application leveraging offloading technology is Google's *Now Playing* [111]. This system uses an always-on microphone and digital signal processor (DSP) to listen to the environment. When the DSP is confident that music is playing, it wakes up the main processor, which is able to search a small, local database of popular songs or send the information to the cloud to match against a large database. With this three-tier architecture, they are able to provide always-on song recognition with less than 1% battery daily battery usage. Similarly, we could run our random forest filter on an always-on, low-power processor which wakes up the main processor when it accepts a window and either runs the CNN on the smartwatch GPU or off-loads it to the smartphone or a cloud service.

While programming an integrated DSP is feasible for large companies, it is difficult for most developers and researchers. There are, however, research projects such as LittleRock [83], K2 [63] and

Sidewinder [59] that try to make these kinds of systems easier to develop. Sidewinder in particular, proposed providing developers with data processing algorithms as building blocks that would run on the low power processor. The BioWatch and SleepMonitor filters we discussed would be straightforward to set up on a system like Sidewinder. If support for machine learning models such as random forests was supported, we would be able to run the random forest filter on the low-power processor.

2.7 Conclusion

In this chapter, we presented WearBreathing, which enables everyday respiratory rate monitoring. The WearBreathing system is composed of two parts. The first is a random forest based filter that is able to detect when input data will result in an accurate respiratory rate. The second is a convolutional neural network based model for extracting respiratory rate from accelerometer and gyroscope data. The combination of these two models resulted in a tunable respiratory rate monitor that enables users to trade frequency for accuracy. Testing on a diverse, out-of-the-lab dataset, we demonstrate that WearBreathing is able to detect respiratory rate with an MAE of 2.05 breaths/minute while producing a reading on average every 50 seconds, which is 3.6 times better than the previous state of the art. We demonstrated that WearBreathing is highly tunable which means that if someone was more interested in accuracy, they could opt to receive less, but more accurate data by simply decreasing a single threshold. Finally, we showed that a current smartwatch is able to run WearBreathing while providing a full day's worth of battery. The net result is a system that, for the first time, is able to accurately monitor respiratory rate outside of lab environments using a smartwatch, that hopefully inspires and enables new research into respiratory rate analysis.

Chapter 3

Challenges with Real-World Smartwatch Based Audio Monitoring

3.1 Introduction

Audio data from a microphone can be a rich source of information [54]. The speech and audio processing community has explored using audio data to detect emotion [110], depression [27, 97], Alzheimer's disease [26] and even children's age, weight and height [57]. The mobile community has made use audio data from smartphones to detect coughing and other respiratory sounds [55, 98], predict students' GPA [106, 107] and detect sleep [12]. However, these studies tend to use well placed, high quality microphones and/or more controlled environments. For example, [97] describes the dataset they use as studio quality and [55] uses a smartphone as a neck pendant during a short (few hours) study where participants followed their daily routine. While a shorter, uncontrolled study is more realistic than an in-lab study, the short duration makes it so that participants likely remain cognizant of the device and of being recorded.

Smartwatches, and wearables in general have potential to make continuous and in-the-wild sensing much more feasible. Smartwatches are readily available and come equipped with many different sensors, often including a microphone. Compared to smartphones, which may be in a user's pocket, purse or on a table for portions of the day, a smartwatch is much more likely to be on the user's wrist. This means data from a smartwatch's sensors is more likely to reflect the user's state. Additionally, smartwatches are much easier to use day after day as compared to other types of wearable devices (e.g., a chest belt or sensors embedded in clothing). However, the relatively recent emergence of smartwatches and the difficulty of conducting in-the-wild studies creates much uncertainty. It is unclear what kinds of sounds smartwatches will pick up in in-the-wild environments and whether these sounds will be of high enough quality to enable detection of events of interest, such as speech and coughing.

To answer these questions, we built a system that uses Android Wear smartwatches to record raw audio and other sensor data from patients with chronic lung disease. We recruit patients with chronic lung disease because this work is part of a larger study that uses passive sensor data from smartwatches to monitor these patients. While we focus on a specific population, which may affect some of our numbers, we do not think this affects the generality of our key results. For example, the amount of speech in healthy patients may be higher than in patients with lung disease who have difficulty breathing. However, our finding that we need more robust methods for detecting speech from smartwatch based audio still stands and is relevant to many different applications.

To our knowledge, we are the first to record raw, unfiltered audio from an in- the-wild smartwatch. In a Research Ethics Board approved study (University Health Network REB # 15-9068), we recruited 16 patients to wear the smartwatch for a three month period while our application recorded data. Our findings include that the audio recorded is of high enough quality to discern speech and respiratory sounds. However, because our data comes from an in-the-wild environment and contains a large variety and amount of noise, algorithms tuned for in-lab studies do not perform well. We find that existing algorithms for Voice Activity Detection (VAD) and cough detection have limited accuracy when applied to our data and that additional sophistication is required to address the challenges of real world audio, which could be an interesting avenue for future research. We also find that a surprisingly high proportion of speech and coughing does not come from the user. These results highlight two problems that will need to be addressed in order for in-the-wild audio analysis to become viable. First, we need more robust methods for automatic event detection such as VAD and cough detection to better handle noisy and inconsistent environments. Secondly, we need reliable methods for distinguishing the source of sounds of interest (user vs. someone else).

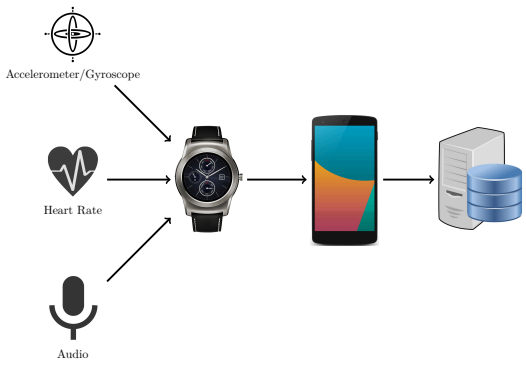


Figure 3.1: Flow of data in our data collection system

3.2 Design

In this section we describe our data collection system and study.

3.2.1 System

Our data collection framework consists of three main components; (1) an Android Wear smartwatch, (2) a phone, and (3) a server. The smartwatch collects sensor data and transmits it to the phone. The phone receives data from the smartwatch and uploads it to a remote server. Finally, the server stores all uploaded data and makes data available for processing and analysis. This data flow is shown in Figure 4.2.

We use two smartwatch models, the LG Urbane W150 and the Moto 360 2nd Generation, all running Android 6.0.1. For the smartphone, we use either the LG Nexus 5 (Android 6.0.1) or Moto G 3rd Generation (Android 6.0). The smartphone is equipped with a 5GB per month data plan and our data collection framework was tuned to fit within the 5GB per month limit. To prevent users from installing other applications, which could potentially interfere with our data collection (other applications recording from the microphone would prevent ours from doing so) and throw off our battery and processing requirements estimations, phones are locked down with a custom launcher and firewall rules.

The smartwatch runs an application that collects sensor data. The main design consideration for this application is battery life. To record sensor data, our application must obtain a partial wake lock from the Android Battery Manager, which prevents the processor from entering sleep mode. Continuously holding this wake lock would drain the battery very quickly. To get around this, we use duty cycling, i.e., recording for a fixed amount of time and then sleeping for a fixed amount of time. Through in-lab testing, we found that for our smartwatches a 20% duty cycling scheme with a 10 minute interval (record for two minutes, sleep for eight) provides enough energy savings to last on average 16 hours on battery, which should be enough to last a full day's use. After deploying we found that these battery saving measures were sufficient with the smartwatches ending 99.9% of days with at least 10% battery remaining.

The smartwatch application has a data collection service that records audio from the microphone as well as data from other sensors. Audio data is sampled at 16 kHz. Unlike regular Android, Android Wear does not support codecs for recording compressed audio. Therefore, we record uncompressed PCM audio and convert it to MP3 using a copy of the LAME MP3 encoder that we cross compile and bundle

with our application. Although lossy compression such as MP3 is undesirable, it is necessary to make data transfers feasible and our annotation and automated methods do not suggest that lossy compression is an issue.

Data transfer from the watch to the phone occurs when the watch is placed on charging. The phone application receives sensor data from the smartwatch over Bluetooth and automatically uploads data to a remote server once per day.

It is worth mentioning that for a practical, production-ready application, transmitting raw audio is not required. Ideally, preprocessing on the phone or smartwatch would either extract events of interest or audio features that are transmitted to the remote server rather than raw audio. However, for our research, we need to be able to evaluate the accuracy of preprocessing and to do that, we need the raw audio.

3.2.2 Study Participants

To recruit participants for the study, we approach patients at three different hospitals and ask them to enroll in a 3 month long study. During the study they are asked to wear a smartwatch that passively collects accelerometer, gyroscope, heart rate and audio data. Patients are informed of the study, its goals, and the invasive nature of the data that we are collecting. We also inform users of the security and privacy measures we are required by the ethics committees to take, such as keeping all data and data transmissions encrypted and stored on privately owned and hosted servers. The biggest hurdle in recruiting users is the privacy concerns associated with continuous recording of audio. Despite the privacy concern, we have been able to find patients who agree to participate in the study.

Patients who agree to participate are shown how to use the smartwatch and smartphone. We include features giving patients some control over their data in order to ease some privacy concerns and make it more likely that patients will agree to participate. Patients are able to stop the smartwatch from recording for a short time and on the smartphone selectively listen to and delete recorded audio. They are advised that an optimal way to use the system is to place the smartwatches charging cradle and the smartphone on their bedside table and plug them both in.

3.3 Analysis and Results

To date, we have collected over 4,100 hours of audio from 15 patients. This data spans over 1059 days with an average of 75 days per patient and 3.9 hours of audio per day. Patients typically put on the smartwatch between 6am and 9am and take it off between 8pm and 10pm. Although the trial was 90 days long, some patients wore the device longer than 90 days due to difficulties scheduling their off-boarding and some ended their trial early but gave us permission to use the data collected so far.

Based on manual annotation, we characterize the audio data in terms of amount of silence, speech and coughing. But manual annotation of audio is expensive, time consuming and infeasible for a study of even our scale, so for speech and coughing sounds, we also evaluate how well existing tools can detect these sounds.

3.3.1 Manual Annotation

To annotate audio, we recruit volunteers to listen through the audio and label speech and respiratory sounds. Because we are working with patients and sensitive audio, we have strict restriction on how we store and use this data. For example, we are required to host the data ourselves on servers within our province. These regulations help keep our patients data secure but also mean we cannot crowd-source annotation.

Manual annotation is expensive and time consuming, so we want to maximize the time our annotators spend listening to useful audio. We do this by removing silent portions of the audio. Our duty cycling means that our audio files as recorded are 2 minutes long. We apply a simple silence detection algorithm to these files that first applies an A-Weighting [24] to the audio signal, followed by a low-pass filter and a moving average. The result of the moving average is compared to a preset threshold to determine if the audio segment contains silence. Using this silence detection algorithm, we find that on average 38.3% of the audio data collected from users contains non-silence and the remaining 61.7% is silence. This proportion ranged from a maximum of 59.3% non-silence down to 20.3% across users who participated in the study. We take non-silence segments of audio and stitch them into longer audio files so that annotators are not constantly loading the next file. This mapping of two minute files to non-silent segments to long files for annotation is maintained so that labels created during annotation can be mapped back to the original two minute file.

During our annotation, we also ask annotators to label coughing, speech, throat clearing, sneezing, sniffling, labored breathing, forced expiration and wheezing. Each label consists of a confidence (low, medium, high) and source (patient, 2nd person, TV/radio). The confidence indicates how sure the annotator is that the label is correct. After a bit of practice, annotators are able to learn the patient's voice in order to identify the source of the event of interest. Contextual information is often useful in identifying non- speech events. For example, if the patient is speaking, stops speaking, coughs and then resumes speaking this is an indication that it was the patient coughing. Additionally, over time annotators were able to learn how the patients coughs sound. The two events of interest that we have found the most occurrences of are speech and cough, which is why we focus on these two for our analysis.

Speech To estimate how much of our audio data is speech, we randomly select one week of audio from eight patients from which speech will be analyzed. After removing silence, we are left with an average of 12.20 hours of audio (4.53 hours SD) per user, which annotators listened to and labeled.

We found that overall, 59% of the non-silent audio was speech. Of the speech, 17.66% was from the user, 17.64% was from another person and 54.35% was from TV/radio. While these proportions may vary between users and populations, it does show that a significant portion of speech comes from non-users. This also poses a challenge in using smartwatch based audio as speech from the patient will have to be differentiated from speech coming from other sources.

Respiratory Sounds Detecting respiratory sounds such as coughing is highly relevant to monitoring lung disease and possibly other health conditions. After annotating 53 hours of silence-removed audio across 7 users we discovered 750 coughs, 238 throat clears and 210 other sounds such as labored breathing, sneezing and sniffling. Figure 3.2 shows the proportion of labels at each confidence level. While the confidence level is a subjective measure, the proportion of labels at each confidence level can serve as a rough approximation of how clear sounds are in the recorded audio and how confident humans are that

they can recognize the sounds. Just over 67% of the annotations were made with high confidence, which shows that humans are fairly confident that they can recognize our sounds of interest in smartwatch based audio.

Looking at the source of coughs, we found that 11.4% of the coughs are not from the patient. This was a surprisingly high proportion given that our patients have a chronic lung disease. When trying to monitor coughing, the proportion of coughs coming from other people may be a source of error worth addressing. As mentioned briefly, we collect other sensor data in addition to audio. It is possible that accelerometer and gyroscope data may be helpful in differentiating the source of coughs.

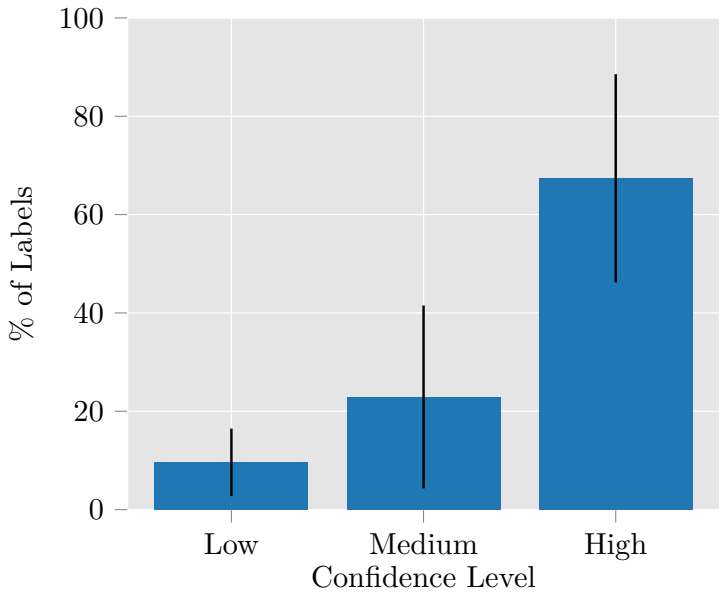


Figure 3.2: Proportion of annotations at the three confidence levels

3.3.2 Automatic Detection

We wanted to evaluate how well existing tools for automatic sound classification perform on our real world data. These tools are often developed and evaluated in controlled environments so validating them is essential if they are to be used in health monitoring applications. We look at tools for detecting speech, known as Voice Activity Detection (VAD), and borrow from existing literature to build a cough detection model.

Speech To evaluate speech detection, we look VAD tools from WebRTC's[1], Loizou [67], Giannakopoulos[2], and LIUM SpkDiarization [86]. WebRTC's VAD has a parameter to control the aggressiveness of the VAD that ranges from 0 to 3, where 0 is the least aggressive about filtering out non-speech and 3 is the most aggressive. The proportion of audio each of these VAD tools classify as speech is shown in Table 3.1. Interestingly, while speech makes up 59% of the audio, most of these tools were too lenient and classified around 90% of audio as speech (the exception being VAD(2) at 80% and VAD(3) being far too strict at 2%). One explanation for this is that these tools were developed and tested on more

[1] `https://webrtc.org`

Method	Speech Proportion (%)
WebRTC(0)	96
WebRTC(1)	95
WebRTC(2)	80
WebRTC(3)	2
Loizou [67]	91
Giannakopoulos[2]	92
LIUM	89
Annotation	59

Table 3.1: Proportion of speech in our non-silence audio data as estimated by different tools and from manual annotation.

consistent audio sources. LIUM for example, was developed for TV and radio broadcasts, [67] used curated dataset of in-lab recordings and [102] assumes that the level of background noise is low.

It is clear that these tools cannot be used as-is on real world smartwatch based audio. Further sophistication is required to not only filter the vast types of noise present in real world audio but also differentiate speech from different sources.

Respiratory Sounds To build an automatic cough detector, we take inspiration from [18], [100] and [2]. We use similar features and machine learning methods as these studies. For feature extraction, we use OpenSMILE [22] to extract spectral features, zero crossing rate, signal energy and Mel-Frequency Cepstral Coefficients from our audio signal using 0.5 second windows with a 0.25 second step. These features, along with annotations from volunteers, are used to train a random forest with an 80/20 random split for training/testing. The average classification accuracy over 100 iterations using monte carlo cross-validation is shown as a confusion matrix in Figure 3.3.

Our feature selection and classifier is inspired by [18], [100] and [2]. However, our classifier does not perform as well as these previous studies. For example, [2], has a sensitivity of 92.8% and specificity of 97.5% in detecting coughs. Our implementation has a slightly lower sensitivity of 91.43% and significantly lower specificity of 83.83%. Similar to speech, we think this is because these studies use higher quality microphones in more controlled environments. Additionally, in these controlled environments, it is unlikely that there are coughs from other sources so these studies do not attempt to differentiate coughs from the user vs other people.

As discovered through manual annotation, 11.4% of coughs did not come from the user. For the cough classifier, we do not take this into account. Coughs are labeled and classified as coughs regardless of the source. However, for real applications of cough detection, this may be a source of error worth addressing. Additional classifiers could be used to determine whether a given cough came from the user or another person.

From our annotation, it is clear that cough and speech signals are present in the audio. The challenge is the significant amount of other noise present in the signal. This shows again, that more sophisticated methods are required to filter noise and to determine whether the user or someone else is coughing.

[2] `http://www.mathworks.com/matlabcentral/fileexchange/28826-silence-removal-in-speech-signals`

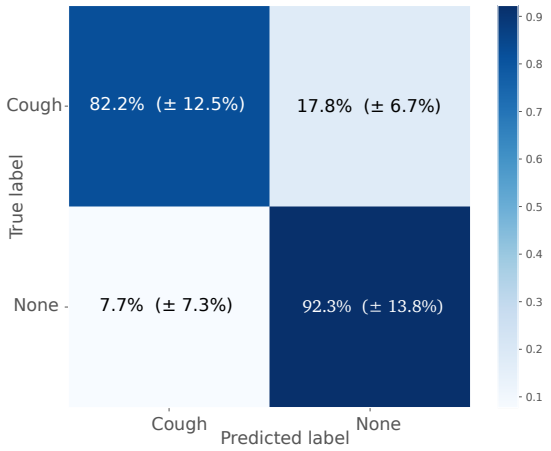


Figure 3.3: Confusion matrix for detecting coughing and clearing throat sounds using a Random Forest.

3.4 Summary and Discussion

We find that the quality of audio from the smartwatch is good enough for humans to be able to detect speech and respiratory sounds. Furthermore, the quality is even high enough discern the source of speech and respiratory sounds. We also find that in real-world audio there is a lot of noise which makes automatic classification more challenging. However, we feel that borrowing more from the audio processing community and utilizing advances in machine learning will yield solutions more robust to the kinds of noise seen in real-world audio.

We also saw that simple detection may not be sufficient. We found that 54% of speech in our data was from a TV or radio. In an in-lab, it is unlikely that there would be any sound from a TV. The exact proportions we report are not as important as the fact that there is a significant amount of unexpected sound. The exact proportions can vary based on the population or even location. For example, teenagers who spend a large portion of their day at school may have a lower proportion of TV sounds and higher proportion of "other speaker" sounds. Regardless of the proportion of sounds, for real-world sensing applications, other sources of sound have to be considered and adjusted for. The task of identifying who is speaking when in an audio signal, known as speaker diarization, is a known challenge and will be highly important to wearable audio sensing. However, identifying the source of non-speech sounds, such as coughs, is a novel problem and may be relevant to monitoring various diseases. In a wearable context, the tasks of speaker diarization and non- speech diarization may be able to leverage other available sensors such as the accelerometer and gyroscope to make smarter decisions about the origin of sounds. For example, sudden movement co-occurring with a cough detected in audio could be a strong indicator that the cough was produced by the user.

3.5 Related Work

Many studies have used smartphones for monitoring. Crosscheck [105] for example, uses a smartphone to monitor patients with schizophrenia. They record audio amplitude (not raw audio), accelerometer, location information, application usage and Android's Activity recognition API to track symptoms related to schizophrenia. Using a similar platform, StudentLife [106] monitors student mental health and

educational outcomes and SmartGPA [107] predicts student's GPA from smartphone sensor data. These works show that smartphone sensor data can be used for a plethora of monitoring tasks. There are benefits and drawbacks to using smartwatches instead of smartphones. A smartwatch is more likely to be consistently on the users whereas a smartphone may be in a pocket, purse, backpack or table. The location of the device can hinder the interpretation of sensor data. For example, a microphone on the wrist is less likely to be muffled than a microphone in a pocket or backpack. On the downside, smartwatches have less computational power, reduced battery sizes and more limited connectivity. Additionally, while a smartwatch may produce more usable data, some data from a smartphone may be of higher quality. For example, during a phone call users are speaking directly into the smartphone resulting in more speech and less noise.

A study by Kalantarian and Sarrafzadeh [48] uses smartwatches to differentiate eating, chewing and speaking. Their audio is recording in a lab setting and because they are interested in eating events, audio is recorded when subjects are eating and the smartwatch is inches from the subject's mouth. Additionally, their audio is recorded in a lab environment, with noise from a mall edited in after the initial recording. Our study takes place in a completely uncontrolled environment which gives us a better representation of real-world audio.

3.6 Conclusion

After deploying a smartwatch based sensing application with real patients, we find that the smartwatch microphone is good enough to pick up speech and respiratory sounds. However, extracting these sounds automatically is difficult because real-world audio contains a wide variety of noise and because a surprising proportion of these sounds do not originate from the patient. We found that existing VAD and cough detection tools have poor accuracy when applied to smartwatch based audio and that more work is needed in filtering out the noise seen in real-world data and to determine whether sounds originate from the user.

Chapter 4

CoughWatch: Real-World Cough Detection Using Smartwatches

4.1 Introduction

Coughing is a common reflex that, although often harmless and normal, can sometimes be indicative of illness or worsening health. In individuals with lung disease, for example, an increase in coughing frequency may be associated with the onset of an episode, or general worsening, of their disease [42]. Therefore, continuously monitoring coughs could be highly valuable for monitoring the health of people prone to, developing, or suffering from lung disease. For healthy individuals, cough monitoring could provide a baseline for health and indicate changes from this baseline.

Unfortunately, current mobile cough detection systems used by the medical community rely on either manual cough counting [71] or specialized, standalone hardware [6, 104] making them burdensome to use. Recently, the European Respiratory Society has stated that there is an urgent need for continuous cough detection systems [74].

The popularity of commodity mobile devices has given rise to a potential alternative: these lightweight, ubiquitous, inexpensive, and unobtrusive devices can be used as sensing platforms for cough detection. However, while there has been some work on cough detection using mobile phones and smartwatches for cough detection [55, 60, 98], that work is usually developed and evaluated using in-lab data, due to the lack of public, high-quality, labeled real-world datasets. In-the-wild data collected from smartwatches is quite different from lab data: audio is noisy, its properties change with the environment, and microphone position is affected by arm movements. As we show in Section 4.2, cough detection models designed with in-lab data do not perform well on real-world data, even when trained using such data.

We thus identify two main challenges in building a cough detector using real-world audio: collecting and annotating real-world cough data, and developing a suitable model for smartwatches, which are low-power, battery-limited devices.

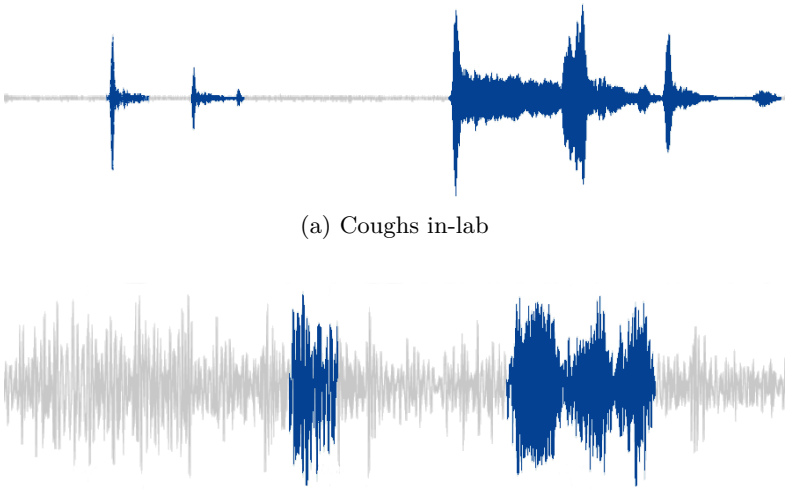
Our Contributions

We propose CoughWatch: a cough detector designed to run on smartwatches and operate on real world data, enabling continuous and unobtrusive cough monitoring.

To build CoughWatch, we conducted a study wherein we collected continuous audio and motion data from 16 participants for 3-months each using smartwatches, resulting in over 4200 hours of audio and other sensor data. Our data collection was approved by the University Health Network Research Ethics Board (REB # 15-9068 and 18-5462). We explore the technical and privacy challenges in such data collection, and how we designed our system and studies to address these challenges while obtaining high quality data. To increase annotator efficiency when dealing with infrequent events such as coughing, we developed a multi-step annotation process and used it to annotate portions of the collected data.

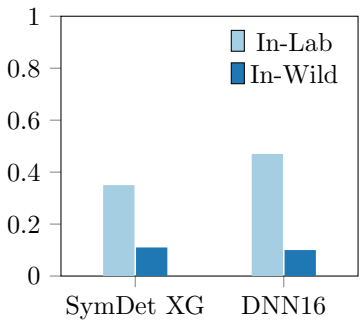
Using this annotated data, we designed and trained CoughWatch: a system for cough detection using smartwatch data. We address noisy in-the-wild data in three ways: we design a larger, more sophisticated model compared to prior work [1]; we augment audio with accelerometer and gyroscope sensors from the Inertial Measurement Unit (IMU) of the smartwatch; and we use a data augmentation approach to improve how well the model learns from our data.

Our evaluation shows that our dataset, combined with our augmentation and sensor fusion methods, results in a cough detection system that achieves a maximum F_1 score of 0.66 on in-the-wild data, compared to 0.10 and 0.11 by previous work. Moreover, we implement CoughWatch on an actual smartwatch and show that our model can be run in real-time on device and, with some battery saving



(a) Coughs in-lab

(b) Coughs in-wild



(c) Detector F1 score.

Figure 4.1: Difficulty of in-the-wild cough detection. While in-lab audio is clear a, in-the-wild audio is often noisy b, impacting performance of existing cough detectors [1, 98] even when trained on in-wild data.

schemes, it is feasible to run CoughWatch and still provide a full day of battery life.

The key contributions of this chapter are:

- A blueprint for conducting continuous monitoring studies that collect sensitive audio from wearable devices.
- A process for efficiently annotating infrequent events.
- A state-of-the-art cough detection system that achieves a 5.7 to 6.7 times higher F_1 score on in-the-wild data and can run for a full day on commodity smartwatches.

4.2 Motivation

Existing cough detection systems are designed and evaluated on in-lab datasets and do not perform well on data collected from smartwatches in the wild. This is because data collected in the wild contains more noise both in terms of quantity and variety. The difference between coughs recorded in an in-lab setting and those recorded in an in-the-wild setting is illustrated in Figure 4.1a and Figure 4.1b.

Collecting in-the-wild data is challenging and obtaining labels for in-the-wild data particularly so [61]. This is why existing cough detectors do not use data collected truly in the wild. Unfortunately, we find that cough detectors developed on in-lab data do not transfer well to data collected in the wild even when retrained on the in-the-wild data. For example, Figure 4.1c shows the difference in F_1 score for two existing works [1, 98] when trained and evaluated on in-lab data compared to in-the-wild data. Despite having 9× more data in the in-the-wild dataset, these two models have a 3.2 and 4.7 times lower F_1 score. A detailed description of the datasets used for the evaluation and our implementation of the algorithms is given in Section 4.3 and 4.5.

The rest of the chapter describes how we addressed the challenges of collecting in-the-wild data, labelling that data and using that data to train cough detection models. First, in Section 4.3, we outline

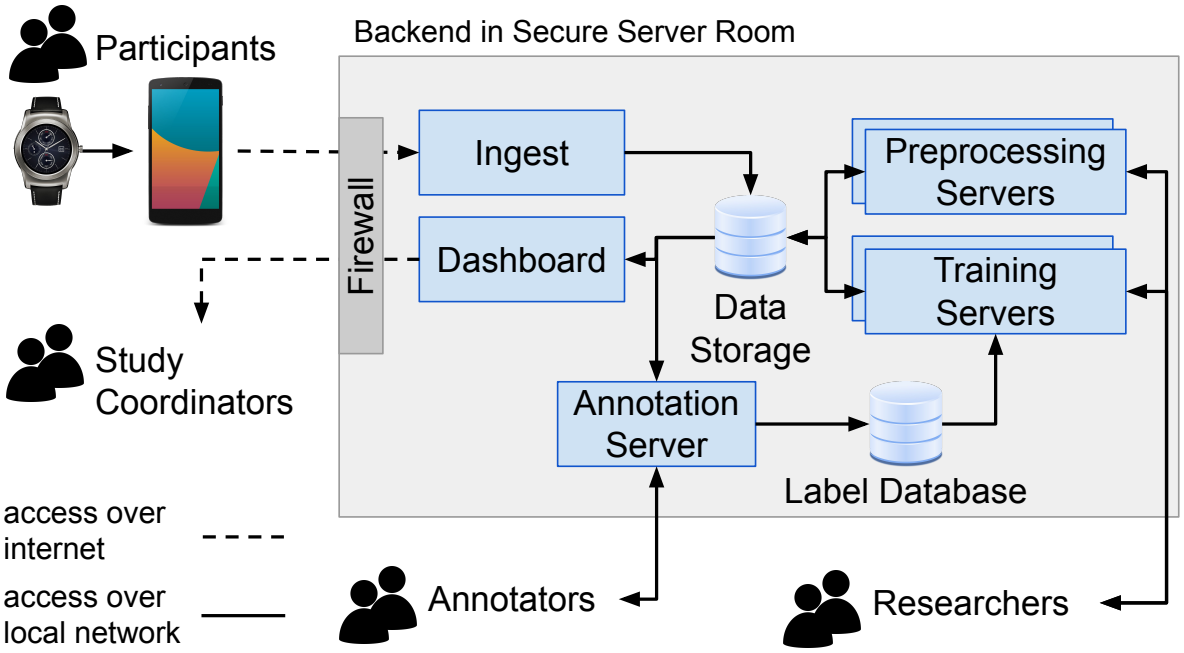


Figure 4.2: Data collection and analysis system.

the design considerations of our data collection system and explain how the various components of the system have been designed in light of the considerations. In Section 4.4, we discuss our model design and data augmentation procedure. In Section 4.5, we evaluate the effect of different components of our system and show the feasibility of running our system on actual smartwatches.

4.3 Data Collection and Preparation

In this section, we describe our data collection, annotation and processing methodology, which we believe would be relevant for any in-the-wild wearable study. Collecting usable in-the-wild smartwatch data from participants requires careful design of the data collection and annotation methodology. Our design was guided by several high-level considerations:

1. **Privacy:** Audio recordings are potentially sensitive. We wanted to design our study to minimize potential risks. Designing for privacy would not only make participants more comfortable but is also a key requirement for ethics approval.

2. **Participant Quality of Experience (QoE):** Poor QoE could cause participant to drop out, or simply not use the system.

3. **Annotator QoE:** Annotating in-wild-data is a tedious manual process. Spending time and effort on improving the annotation process and building tools to make annotators user experience better meant more labels with fewer errors.

4.3.1 Infrastructure

The overall architecture of the data collection and analysis system is shown in Figure 4.2. We describe the motivation and purpose of different components of the system below and the implementation of the more complex components is discussed in the subsequent subsections.

Smartwatch and mobile phone Each participant received an Android Wear Smartwatch (LG Urbane or Moto 360) and a LG Nexus 5 smartphone, both with our data collection applications preinstalled.

These applications are designed for minimal interaction: the only requirement from participants is to wear the watch in the morning and charge it at night. The smartwatch application records data from the accelerometer, gyroscope, heart rate and microphone sensors. To preserve battery life we used a 20% duty cycle: 2 minutes recording followed by 8 minutes not recording. This resulted in over 16 hours of battery life which was generally sufficient for a full day's recording. The smartphone application receives data from the smartwatch and eventually uploads it to our back-end. Therefore participants were instructed to just leave the smartphone at home and plugged in to a charger. This design not only reduced the chance of data loss due to lost or malfunctioning phones, but also meant that participants had one less object to carry around during the day. It also minimized potential risk to personal data (contacts, calls, texts, etc.) since participant did not need to install our software on their phone, nor use the provided smartphones.

We also implemented additional privacy features. Smartwatch recording could be paused at any time, while the smartphone app allowed participants to listen to and delete audio recordings that have not yet been uploaded. Data that had already been uploaded could be deleted by making a request to our study coordinators.

Data Ingestion Data from watches is relayed by the smartphone over an encrypted SSH connection to the ingest server, which then stores the data in the central data store.

Data Storage The central data storage stores raw data, as well as audio and features that have been pre-processed for annotation and training. Like all servers in the system, this server resides behind a firewall in a secure server room; internal access to the data happens over a private network.

Study Coordinators The study was facilitated by dedicated study coordinators who recruit and enroll new participants, and teach them how to use the system. The coordinators also monitored the status of enrolled patients to identify if participants were having any issues with the system and then reached out to investigate and resolve issues.

Only the coordinators have access to the personal and health information of study participants. Coordinators assign a random ID to participants and any digital data we collect is linked with this ID and not the participants' identifiable information.

Dashboard The dashboard helps study coordinators keep track of the status of participants and collected data. It shows the participants currently enrolled, whether we are receiving the expected amount of data and flags potential issues.

Pre-processing and Analysis Servers For pre-processing and analysis, we use multiple powerful servers located in the secure server room. We developed a configurable, multi-threaded feature extraction pipeline engine that allowed researchers to quickly define features to be extracted on the full dataset. We have open sourced this pipeline engine[1].

Annotation Server Our annotation server is a bespoke web application for annotating data. Serving audio segments through a web app makes it harder for annotators to copy data, improving security and privacy. It also allows all actions performed in the app to be logged and audited if needed. This server sits behind the firewall and is only accessible on-site, and is also password protected.

Annotators We hired annotators to come in to our lab and annotate data. On their first day they were given training on how to use the annotation interface as well as privacy expectations when listening to participant audio. During their first week, at the end of each shift their annotations were reviewed together with one of the researchers. Annotators were provided high quality, comfortable headphones to make listening to audio easier.

4.3.2 Data Collection Process

We collected two datasets: a large in-the-wild, and a smaller in-lab dataset.

In-the-wild Dataset We recruited 16 participants who had a chronic lung disease (4 female, 12 male, mean age 69.3), for a 3-month in-the-wild study. Participants were on-boarded by the study coordinator and given a smartwatch and a smartphone equipped with our data collection applications. As part of informed consent, participants were made aware of the data being collected by the smartwatch application and shown how to use the privacy features.

Participants went about their normal lives, wearing the smartwatch during the day and charging it at night. While most participants completed the 3 month period, some dropped out early due to either personal reasons or difficulty in using the system but still consented to their data being used.

In-lab Dataset While the goal of our work is to build an in-the-wild cough detector, we found that many existing works were developed and evaluated on in-lab datasets. We collected the in-lab dataset to enable direct comparisons.

For the in-lab portion, we recruited an additional 13 participants. The exact selection criteria is listed on our `ClinicalTrials.gov` entry (NCT03857061). To more closely match existing works, for this portion we record audio data from the smartphone rather than a smartwatch. Participants either held the smartphone or placed it on the table in front of them. The participant is guided through several exercises, tests and tasks, such as speaking, walking, lung function tests, and voluntary coughing. Because these tests were run in a lab, the audio recordings contained very little noise. The study lasted just under 1 hour for each participant, so we did not have any battery limitations and recorded continuously rather than on a duty-cycle.

[1] `https://github.com/SPOClab-ca/COVFEFE`

4.3.3 Audio Preprocessing and Segmentation

Prior to annotation, we first apply a pre-processing step designed to reduce the amount of audio annotators must listen to and to make their task easier. We first apply an automatic silence removal step to remove portions of audio that do not contain any sound. To detect silence, we apply an A-Weighting filter [24], square the result and then apply a low-pass filter. We average the result of the low pass filter over 0.5 second rolling window and compare it to an empirically determined threshold (50000) to make a silence/non-silence decision. We convert consecutive non-silent decisions into segments consisting of a start- and end- time. During our preliminary annotation attempts we found that annotators wanted more context information in the these segments, therefore we expanded non-silent segments by one second in each direction to help capture extra context information. Additionally, if the gap between two non-silent segments is less than 1 second, we coalesce the two segments into one.

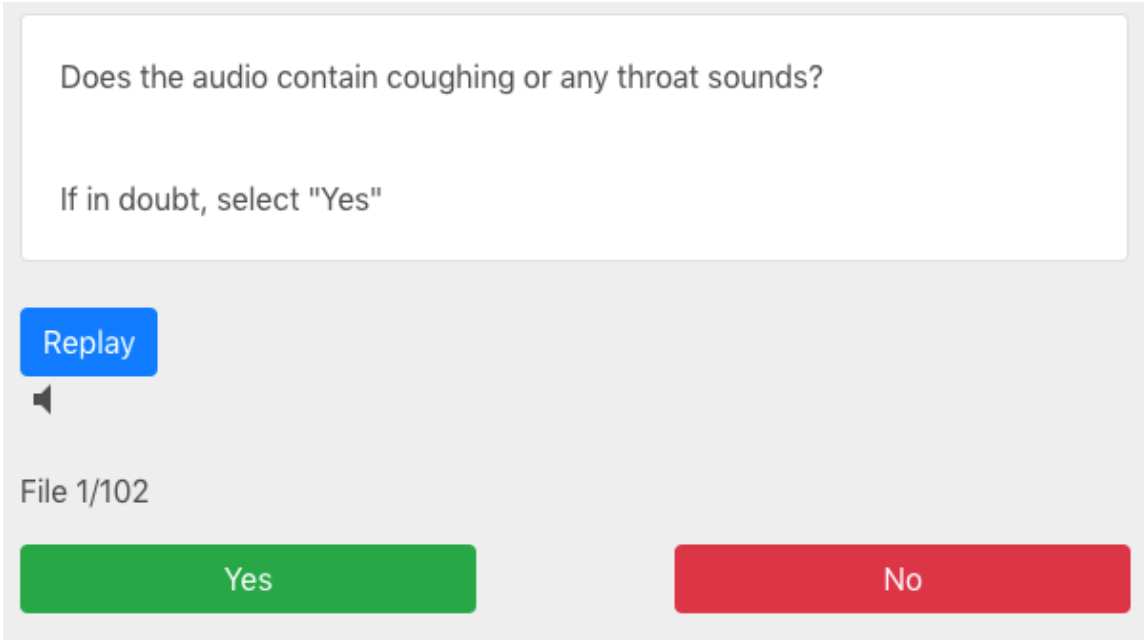
4.3.4 Annotation

Annotating data is a time consuming and tedious task. The data annotation procedure can either exacerbate the tediousness and time required or alleviate it. For example, in our first attempt to annotate data, we removed silence and stitched together non-silent segments into hour-long files. We asked annotators to use a standard audio playback software to find events of interest and record the details in a shared spreadsheet. However, we very quickly realized this was slow and error prone. Annotators were spending much of their time scrubbing through audio, manually typing in timestamps, and switching between the audio player and the spreadsheet. Time spent on these tasks was time not spent listening to audio, translating to poor efficiency.

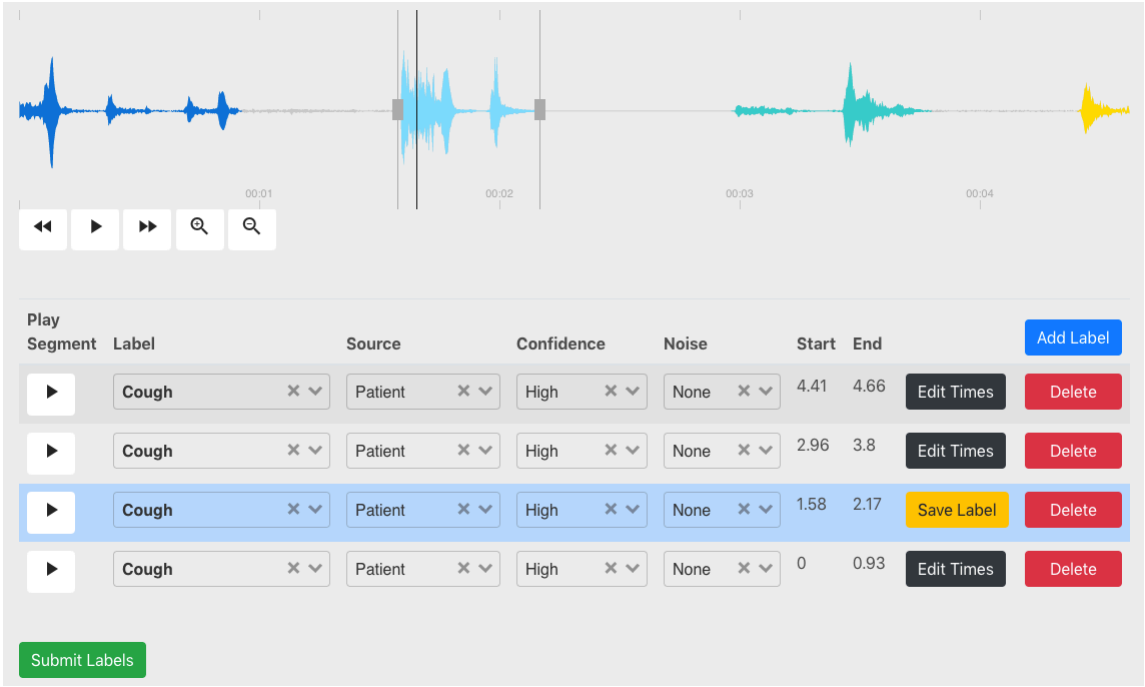
In designing a more efficient solution, we arrived at a bespoke web application that streamlines annotations to maximize the time spent listening to audio as well as to minimize cognitive load as high cognitive load has been shown to increase errors[101]. The key insight that drove this improved process was to split the annotation into a coarse-grained annotation step where annotators listen to a short clip and decide whether it *might* contain a cough, followed by fine-grained annotation to annotate the precise position and type of events in the clip.

Dividing the pre-processed non-silent segments to 10 second *clips* of unbroken audio, with 1 second overlap at the beginning and end of each clip, with a single label per clip presents annotators with a far less demanding task. Because our event of interest (coughing) is rare, much of the overhead and cognitively heavy tasks (scrubbing through audio, noting timestamps, identifying cough metadata) are completely unnecessary for the vast majority of the data. Therefore, for the coarse-grained annotation step, we use a simple user interface that removes these overheads. This interface, shown in Figure 4.3a, contains only three buttons. A “Yes” button indicating the clip contains an event of interest, a “No” button indicating the contrary and a “Replay” button that allows annotators to listen to the clip again. When the page loads, the audio clip starts playing automatically and the buttons are only made active once the clip has finished playing. We do not show a waveform or allow scrubbing through the audio, nor do we ask to locate the cough inside the clip. The clips marked “Yes” are placed in a pool to be fine-grained annotated. Coarse-grained annotation acts as a quick filter to eliminate audio that is “uninteresting”, so annotators are asked to label segments as relevant if they are unsure.

During fine-grained annotation, shown in Figure 4.3b, annotators are asked to mark the start- and end-times of a coughing episode. Since fine-grained annotation was run on a very small number of clips



(a) Coarse-grained annotation



(b) Fine-grained annotation

Figure 4.3: Interface for coarse- and fine-grained annotation.

Table 4.1: Hours of audio recorded and amount (hours and as % of total) at each stage of the annotation pipeline (non silence (NS), coarse-grained (CG), fine-grained (FG)) along with the number of coughing episodes discovered through labelling.

Dataset	Total	NS		CG		FG		Coughs
	h	h	%	h	%	h	%	
Lab	12.2	5.8	48	5.8	48	0.3	2.5	608
Wild	4225	1726	41	97.4	2.3	3.1	0.07	1279

which are likely to contain coughs, the complex interface is not as straining. In fine-grained annotation, the annotator is presented with a waveform of the audio, and given access to start, stop, skip, and magnify buttons. The annotator can play the audio and watch the waveform to better identify sounds. When a cough is identified, the annotator marks an interval on the waveform by dragging the start and end of the segment. We also asked annotators to try to capture the source of the sound (e.g., patient, 2^{nd} person, or TV/radio), their confidence level (i.e., low, medium, high), and the type of noise overlapping with the cough.

The two-stage annotation pipeline allows annotators to frequently switch between the two tasks, alleviating the tediousness of annotation. To make switching between the two tasks easier, we organize coarse-grained annotation into batches that would take roughly 10 minutes to complete (approximately 100 audio clips). After completing a batch of coarse-grained annotation, annotators can start another batch or switch to fine-grained annotation. To ensure that data from all participants are annotated concurrently, batches are sourced from participants in a round robin fashion. To measure inter- and intra-annotator agreement, 10% of audio clips are re-annotated again at a later date.

With a multi-step pipeline where we have multiple levels of splitting, removing, and annotating audio, special attention must be paid to ensure labels can be mapped back to the raw audio. The annotation server keeps track of where audio clips occur in the non-silent segments and where the non-silent segments occur in the raw audio and maintains this information in the label database.

4.3.5 Resulting Datasets

We collected 4225 hours of sensor and audio data from the in-the-wild study. Applying our annotation pipeline we found that of the 4225 hours, 1726 (41%) hours was non-silent. Of the 1726 non-silent audio, to date we have annotated 97 hours (2.3% of total) using coarse-grained annotation. Only 3.1 hours of audio were passed on from coarse-grained annotation to fine-grained annotation resulting in 1279 coughs identified. These numbers are summarized in Table 4.1.

The in-lab study resulted in a total of 12 hours of in-lab audio of which 5.8 hours (48%) were non-silent. We coarse-grained annotated all 5.8 hours, which lead to about 20 minutes needing fine-grained annotation. The fine-grained annotation resulted in 608 coughs. The proportion of coughs in the in-lab dataset is much higher than in the wild dataset because the in-lab study contained a voluntary coughing session where participants were asked to cough.

Table 4.2: Dataset summary after fine-grained labels.

Dataset	Positive samples	Negative samples	Balance Ratio
Lab	288	6779	1:23
Wild	912	65062	1:71

4.4 CoughWatch

We define the following classification task: given a 10 second audio segment (and, potentially, corresponding accelerometer and gyroscope measurement), we wish classify whether the segment contains a cough. We build two cough detection models for this task. The first, which we call CoughWatch Audio Only (AO), relies solely on audio data, while the second also, which we call CoughWatch Sensor Fusion (SF), also includes data from IMU sensors (gyroscope and accelerometer).

Input Data For input to our models, we use smartwatch audio, accelerometer and gyroscope data. Audio is monochannel PCM sampled at 16 kHz using 16 bit signed integer PCM and any audio clips shorter than 10 seconds are zero-padded to 10 seconds. Audio data is pre-processed using a 24-length gammatone filterbank applied to 20ms frames and converted into a spectrogram as described by Saba [88]. Accelerometer and gyroscope data is sampled at 20 Hz, and is fed into the models without any preprocessing.

Labels Note we cannot use the coarse-grained labels to train or evaluate our models. We explicitly instructed annotators to mark any segments they were unsure of as coughs, to be corrected during fine-grained annotation. These labels therefore contain many false positives that could hurt its performance. Instead, we cross reference the coarse-grained labels with the fine-grained labels to build corrected labels for training and evaluation. This correction process resulted in the labeled datasets summarized in Table 4.2. The number of positive samples is lower than the overall number of coughs because many 10 second clips contain multiple coughs.

Data Augmentation We augment the audio data in two ways. First, we drop every second sample in an audio segment and linearly interpolate the dropped samples. We apply this interpolation-based augmentation method twice, once dropping even samples and once dropping odd samples. The second augmentation method adds white Gaussian noise, scaled such that the amplitude of the noise is 1% the amplitude of the original audio. Combining these two augmentation methods quadruples the size of our training data. While we can vary the parameters of both methods (e.g., dropping every n^{th} sample, different noise amplitudes) and even repeat them, we found that additional augmentation did not yield better performance. When training the audio and IMU model, audio data is augmented but IMU data is left unchanged.

Audio Only Model For CoughWatch AO, we use a convolutional neural network [31] (CNN) as shown in Figure 4.4a. This model predicts whether the audio contains a cough, given the 10 second audio spectogram as input. The spectrogram is passed through three *convolutional sets* with the final set connecting to a flatten layer. Each convolutional set consists of a convolutional layer, followed by a batch normalization layer, followed by a max pooling layer. The flattened output of the convolution is

passed to a dense network with three layers of 128, 64, and 32 neurons each. To reduce overfitting, we add dropout layers with rate of 0.2 between every dense layer, as well as the start and end of the dense network. The final dropout layer connects to the output layer where we have two nodes corresponding to our prediction. All layers use rectified linear activations [76] except the output layer, which uses softmax.

Sensor Fusion Model CoughWatch SF consists of three sub-networks that feed into a single dense network as shown in Figure 4.4b. The audio sub-network is identical to the trained CoughWatch AO model, and its weights are frozen during training of the SF model. We feed the accelerometer and gyroscope data into two identical networks, consisting of a convolutional set, a flatten layer, and a dense layer with dropout. The output of all three networks is then concatenated and passed through two more dense layers with dropout, before the final output layer. The structure of the output layer, the activation function, and the optimization are the same as CoughWatch AO.

4.5 Evaluation

We train models using adaptive moment estimation [49] (Adam) with $lr = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $decay = 0$. We reduce the learning rate by a factor 10 if validation loss does not decrease for 3 epochs. We also use early stopping: we monitor the validation loss, and stop training if it has not decreased for 15 epochs. Our models generally train for fewer than 30 epochs.

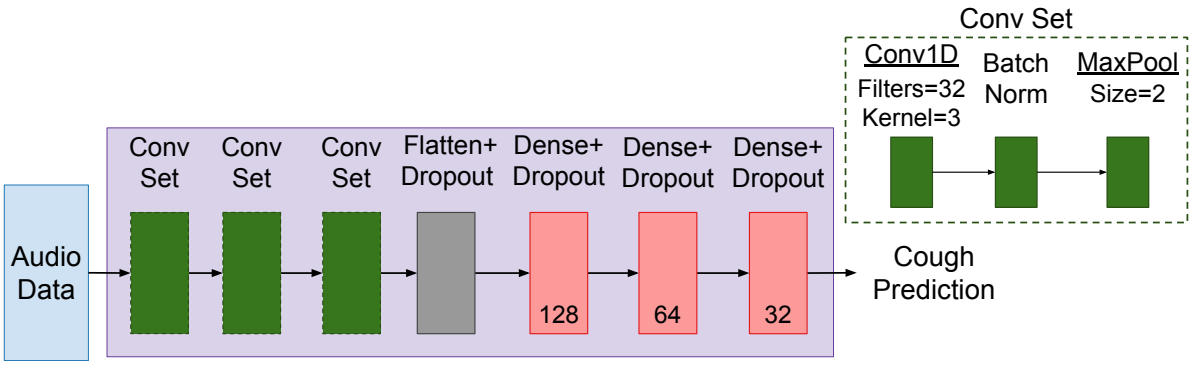
For all models, our primary method of comparison is through precision-recall curves. We use stratified Monte Carlo cross validation with 5 rounds. In each round, we choose 80% of the data at random for training and use the remaining 20% for testing. Of the 80% chosen for training, we randomly select a further 20% that is used as a validation set. Each of these sets is stratified, meaning the proportion of coughs is preserved within each set. We use the procedure proposed by Forman and Scholz [25] To aggregate multiple rounds of training into a single precision-recall curve: we combine predictions on the test set from each round into a single list and compute the precision-recall curve from that list; individual precision, recall, and F_1 scores are based on individual points from this precision-recall curve. This is an unbiased method of estimating F_1 scores from multiple rounds of training [25].

4.5.1 Cough Detection

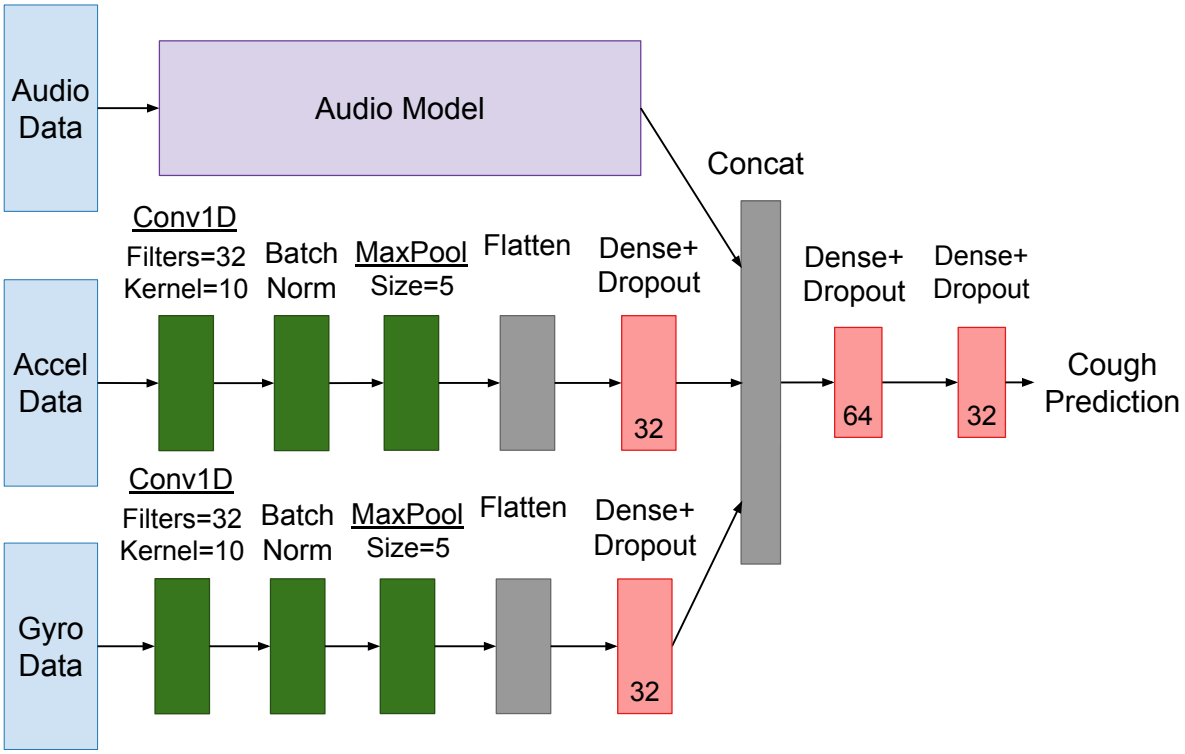
We evaluate the performance of CoughWatch AO and CoughWatch SF on our labeled datasets, and compare it to two state-of-the-art cough detectors. The first is SymDetector [98], which computes features on an audio segment and uses a support vector machine (SVM) to detect coughs. We found that replacing the SVM with a gradient-boosted tree [11] resulted in better performance, and we therefore also compare against a version of SymDetector with XGBoost (SymDet XG). The second model, denoted DNN16 [1], uses short-time Fourier transforms and a CNN-based architecture. We discuss our implementation of these existing works in Section 4.5.4.

Figure 4.5 shows the precision-recall curve for both the in-lab and in-the-wild datasets. For each curve, we find the point that maximizes F_1 score: Figure 4.6 shows the precision, recall, and corresponding F_1 score for that point for all detector and dataset combination.

CoughWatch's detection performance is superior to prior work on both datasets. On the in-lab dataset, CoughWatch AO achieves a maximum F_1 score of 0.773 with a precision of 0.838 and recall of



(a) CoughWatch Audio Only (AO) model



(b) CoughWatch Sensor Fusion (SF) model

Figure 4.4: Cough detection models.

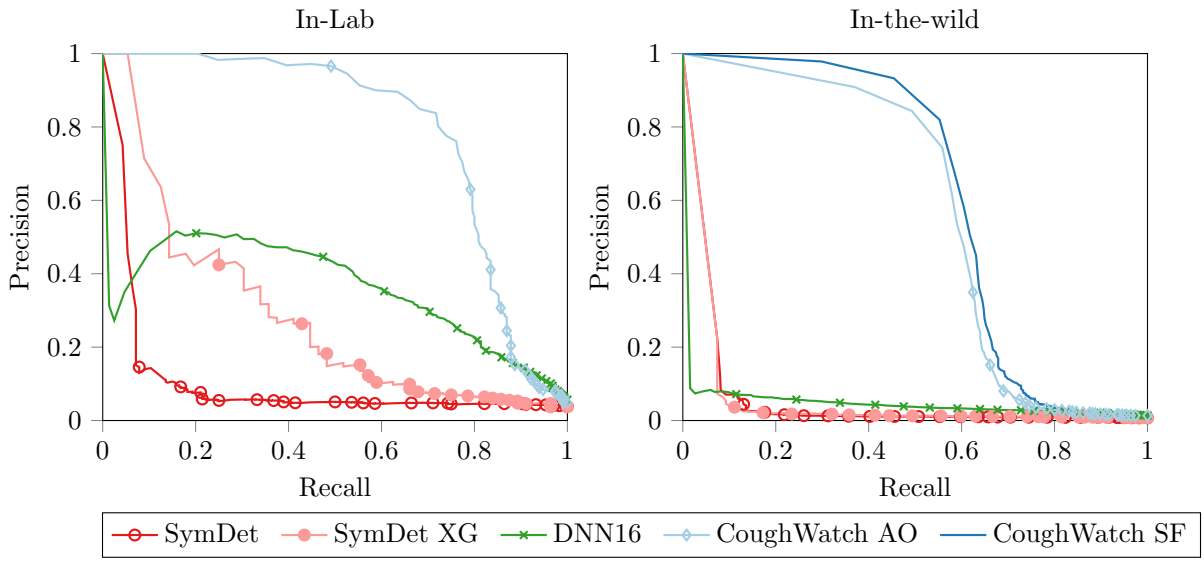


Figure 4.5: P-R curve for in-lab and in-the-wild data.

0.717, higher than DNN16 ($F_1 = 0.466$), SymDetector ($F_1 = 0.124$) and SymDetector with XGBoost ($F_1 = 0.352$). On the in-the-wild dataset, CoughWatch SF achieves a maximum F_1 score of 0.660 (precision of 0.820 and recall of 0.552), 5.7 times higher than SymDetector with a maximum F_1 score of 0.111 and 6.7 times higher than DNN16 with a maximum F_1 score of 0.095. CoughWatch AO is also superior to prior work, achieving a maximum F_1 score of 0.638 (0.743 precision and 0.559 recall).

Conclusion: CoughWatch substantially outperforms existing cough detection systems with on in-the-wild data and in-lab data.

4.5.2 Effect of IMU Data

As shown in Figure 4.5 and Figure 4.6, the CoughWatch SF model, which combines audio data with accelerometer and gyroscope data, outperforms the audio-only model. To better quantify this difference, we compare the precision of CoughWatch SF and CoughWatch AO models for the same recall. Between 40% and 70% recall, we observe a 4 to 15.5 percentage point increase in precision when using the IMU data.

4.5.3 Effect of Data Augmentation

We employed two data augmentation techniques that quadruple the amount of training data. Figure 4.7 shows how precision and recall grow when we increase the size of the training set, with and without data augmentation. Data augmentations boosts precision and recall substantially: using augmented data yields higher precision, from 3.5 to 6.2 times higher than when using un-augmented data. Similarly, augmented data results in between 1.3 to 4.3 times higher recall. Additionally, we observe that precision and recall scores have not plateaued with the amount of annotated data, implying that increasing the amount of annotated data is likely to yield further improvements.

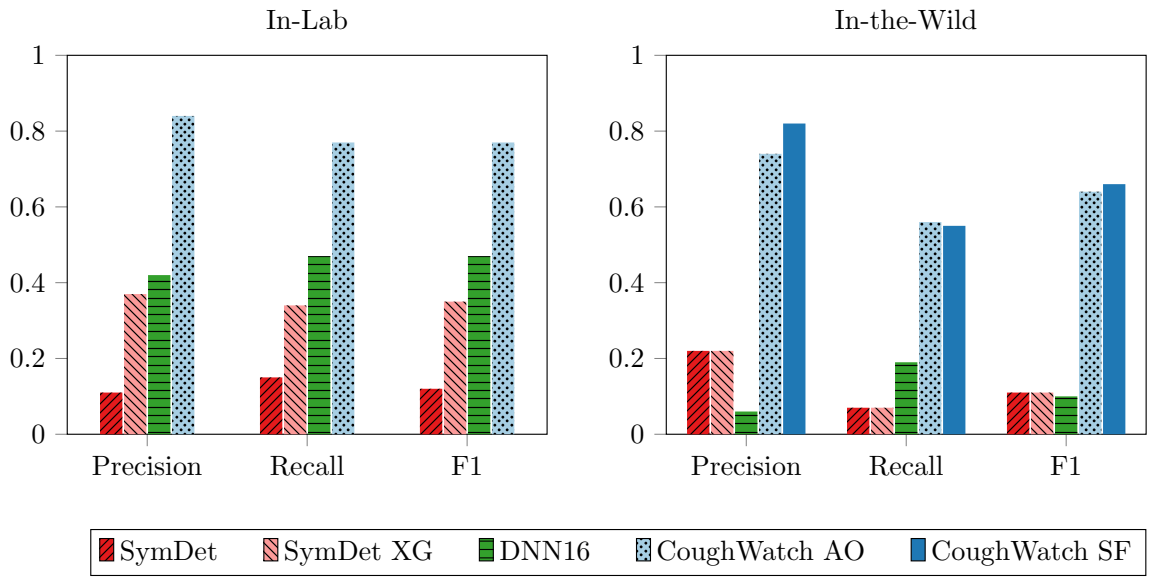


Figure 4.6: Maximum F_1 score of all cough detection models along with the precision and recall at that F_1 score.

4.5.4 Implementation of Previous Work

The datasets and models for the cough detectors described in SymDetector [98] and DNN16 [1] are not publicly available. We implemented these detectors as closely as possible, based on the details provided in their respective papers and using publicly available pre-processing code from DNN16[2], and trained them on our datasets. To verify our implementation, we compare the ROC curve and ROC AUC score of the CNN model proposed in DNN16 [1]. The ROC curve of all models is shown in Figure 4.8. We observe that the curve for DNN16 is similar to the one presented in [1] (Figure 6), and that the AUC of 0.92 we obtain on in-lab data closely matches DNN2016 reported AUC of 0.95.

We also observe that ROC curves do not well reflect the performance of the model on imbalanced real-world cough detection. For example, while DNN16's performance on the in-the-wild dataset is worse than on the in-lab dataset, this difference is not as drastic as seen on a precision-recall curve. ROC curves are insensitive to class imbalance. Given that cough detection in the wild is a highly imbalanced task since real-world coughs are comparatively rare, a ROC curve is a poor tool for reporting cough detection performance. In contrast, in-lab cough data is likely to be more balanced because participants are often asked to voluntarily cough and audio recordings are more limited. In our two datasets, we see a 1:23 class imbalance on the in-lab data, compared to the 1:71 seen in the in-the-wild data.

4.5.5 Running on a Smartwatch

A smartwatch is a battery- and CPU-constrained device. To evaluate its feasibility for continuous monitoring, we implement CoughWatch on a smartwatch, and measure runtime and the effect on battery life. In our testing, we use three Android Wear smartwatches representing three generations of wearable processors: LG Urbane (Snapdragon 400), Huawei Watch 2 (Snapdragon 2100) and Misfit Vapor X (Snapdragon 3100).

[2] `https://github.com/justiceamoh/eGRU/blob/master/Cough_Detection.ipynb`

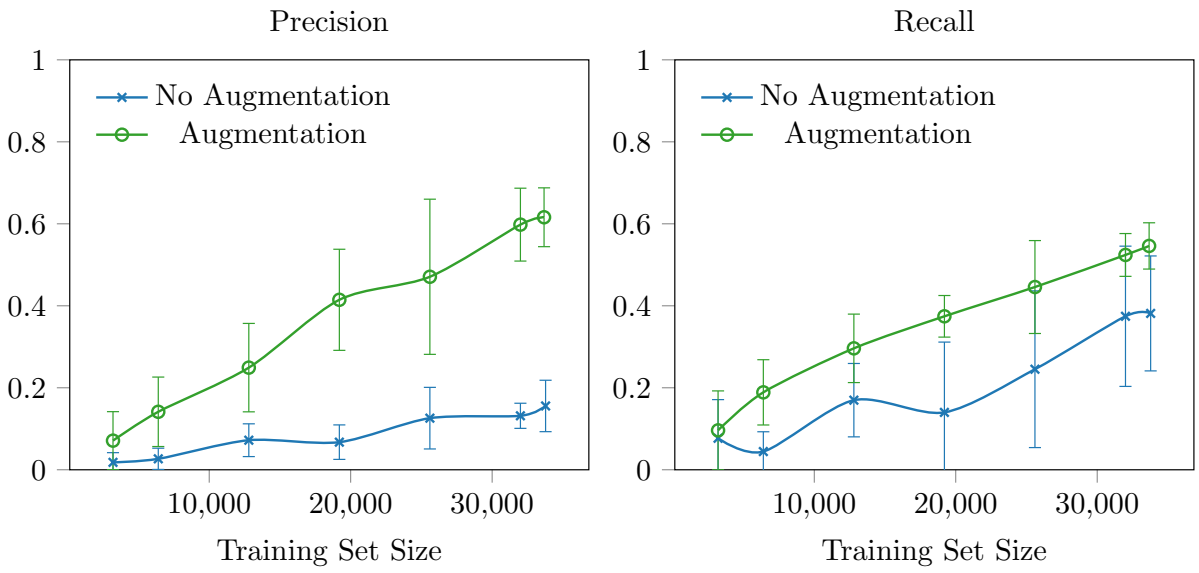


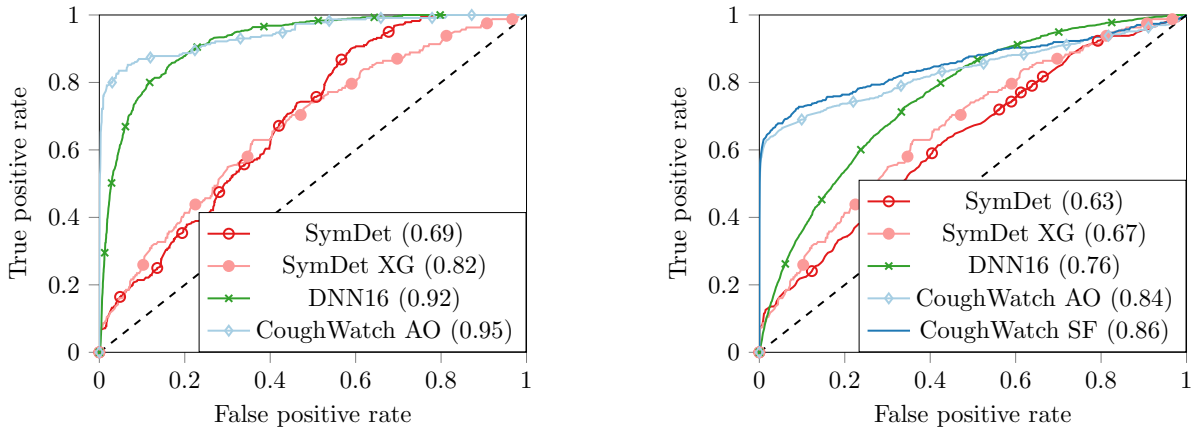
Figure 4.7: Learning curves with and without augmentation.

We modify the application we used for in-the-wild data collection to run continuously instead of with a duty cycling scheme. Sensor data from the accelerometer, gyroscope, and microphone are saved to a 10 second buffer, which is then sent to the our cough detector. The cough detector first pre-processes the audio data, converting it into a gammatone spectrogram using a cross-compiled gammatone library[3] and JTransforms[4]. JTransforms is a library for fast FFT calculations, and has been used in real-time audio processing applications[95]. We experimented with KISS FFT, a native C/C++ based FFT but found no significant difference in the runtime ($p = 0.16$). Therefore, we use JTransforms in our implementation. The spectrogram, along with the IMU data, is fed into the CoughWatch CNN, converted to run on TensorFlowLite[5].

First, we evaluate whether the pre-processing and CNN can run in real time on a smartwatch. We run our application and log the start and end timestamp of the pre-processing calculation and the CNN inference on 10 second data. We collected at least 2000 runs for each watch. The mean and standard deviation of the runtime for all three of our watches is shown in Figure 4.9a. We see little difference between the LG and Huawei watch (2.3s and 2.2s respectively), but a roughly 30% faster overall runtime on the Misfit (1.5s). We observe that all watches are able to run our cough detection system in real time, taking less than 2.3 seconds to process 10 seconds worth of data. Interestingly, the CNN is much faster than computing the spectrogram – accounting for only 3–10% of the overall runtime.

Next, we evaluate the effect on battery life of running CoughWatch SF on a smartwatch. To do so, we configure our application to run under four different conditions. For each condition, we conducted six runs where we charged the watch to over 95% and then ran the application continuously until the battery fully discharged. In the first condition, *idle*, our application runs periodically to record battery levels but does not record any sensor data or run our model. In the *IMU* condition, our application records data from the accelerometer and gyroscope but not from the microphone. In the third condition,

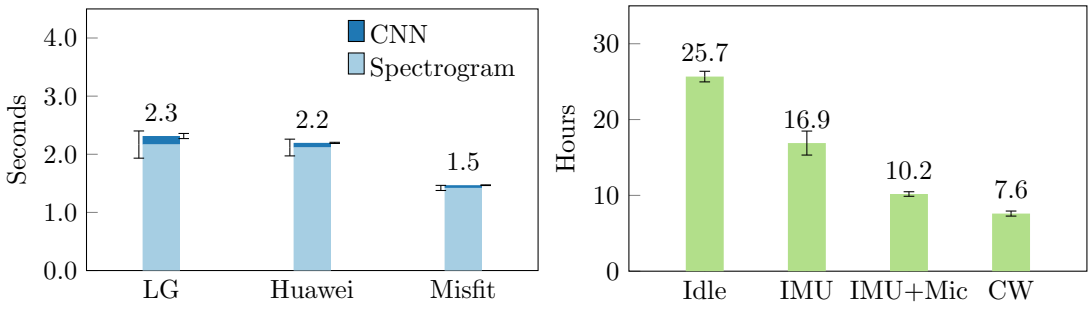
[3] https://github.com/mmmaat/libgammatone
[4] https://sites.google.com/site/piotrwendykier/software/jtransforms
[5] https://www.tensorflow.org/lite/



(a) In-lab

(b) In-the-wild

Figure 4.8: ROC curve for in-lab and in-the-wild data. AUC shown in parentheses.



(a) Runtime.

(b) Battery life.

Figure 4.9: Runtime of running our system on three smartwatches a, and resulting battery life on LG Urbane under different operating conditions b.

IMU+Mic, we read and save IMU and audio data but do not run the cough detection model. Finally, the *CW* condition both collects and preprocesses the data and then runs the CoughWatch SF CNN. Figure 4.9b shows the mean and standard deviation of the battery life for these conditions when running continuously on the LG Urbane. Since this watch has the oldest of the three processors and the smallest battery, we expect the other two watches to have longer battery lives. In the idle condition, we measured the LG Urbane to have a 25.7 hour battery life. Recording data from the IMU drops this down to 16.9 hours and recording from the IMU and microphone reduces battery life further to 10.2 hours. Running CoughWatch reduced battery life to 7.8 hours.

Given that in our data collection studies we rely on duty cycling to even simply record data, we wanted to estimate the battery life of running cough detection in a duty cycling scheme. To do so, we follow the simulation approach proposed in our prior work [58]. The simulator starts at $minute = 0$ with $batterylevel = 100$ and estimates what the battery level should be every minute by sampling from a normal distribution described by the mean and standard deviation of the idle condition, or the mean and standard deviation of the CW condition, based on the duty cycle state. Based on 100 runs of this simulator, we estimate that running CoughWatch on a smartwatch with a 2min/8min duty cycling scheme would provide 17.4 hours of battery life (SD: 0.48 hours), which is enough to last a full day. Using silence detection to reduce the number of times proprocessing and the CNN have to run would likely result in additional battery life.

Conclusion: CoughWatch can run on a smartwatch in real-time. Using a duty cycling scheme, which is already required for simply recording data, CoughWatch can be run while still providing a full day of battery life

4.6 Lessons Learned

We describe lessons we learned that we feel will be of use to others conducting similar studies:

- *Annotator User Experience:* Building a well thought-out, dedicated UI which minimizes wasted time resulted a less frustrating experience for our annotators. This, combined with a comfortable physical setup, improved annotator throughput, and reduced error rates compared to our original setup, which was based on general audio editing tools and shared spreadsheets.

- *Evaluation Metrics:* Past work has focused on Receiver Operator Characteristics (ROC) curves and the area under the curve (AUC). However, ROC does not take into account the precision of a model, which is important when dealing with imbalanced datasets. Given the rarity of coughs in real-world audio data, precision-recall curves and F_1 scores are better indicators of a model's performance.

- *Model Architectures:* We experimented with different alternative network architectures, including recurrent models. However, we found that on our data convolution networks not only outperform recurrent models in accuracy, but are also an order of magnitude faster to train.

What Worked

- The two-step annotation process with its early rejection worked very well. It allowed us to optimize the user interface for each step, reduced cognitive load, and gave annotators an opportunity to switch tasks to alleviate boredom, enabling them to work more efficiently over long periods of time.

- Having a dedicated study coordinator provided several benefits. The coordinator can contact participants to investigate and resolve any technical issues with the smartwatches or mobile phones. Since participants were already familiar with the coordinator from the on-boarding procedure, they were not surprised or wary when contacted. Moreover, only the coordinator had access to personal and health information of study participants, which helped maintain their privacy and made it easier to add researchers on the analysis side.

What Could be Improved

- While we explored several variants of active learning to prioritize annotator effort, we did not see any clear improvements in our models. Active learning may be effective, we have not yet found a solution we would be comfortable deploying.
- Our annotation server maintained a log of all requests made to the server. However, better client-side instrumentation would have allowed us to capture more detailed user behaviour to evaluate and improve our annotation pipeline. For example, time to annotate a clip, and where time was being spent inside specific tools.
- Our feature extraction pipeline was flexible and easy to use, but started hitting the limits of what a single machine can do in a reasonable amount of time. Improving our pipeline to support multiple machines would yield more efficient research. Data-parallel frameworks such as Flink [8] and Spark [113] could facilitate this.

4.7 Related Work

Beyond the two cough detection systems we previously discussed [1, 98], many other cough detection systems have been developed in other works. Some systems have been developed using publicly available videos of coughs [29, 53]. However, these models may not generalize to real-world applications; for example, the AudioSet [29] contains dog coughs, and may not be representative of real-world coughs. Other systems have designed and conducted studies in order to collect datasets. For instance, MobiCough [80] used a collar based microphone to collect their data. Similarly, Larson et al. [55] and Kadambi et al. [47] used a neck-worn device to record audio. While these ideally placed devices are better suited to pick up cough sounds, they are less practical for long term use. In our data collection study design, we used a smartwatch for cough detection as smartwatches are unobtrusive, readily available and have been shown to be a feasible method for monitoring Wu et al. [109].

Conducting real-world data collection studies can result in a large, noisy dataset. Annotating this data is challenging, as noted by Kadambi et al. [47] and Larson et al. [55]. Our proposed two-step annotation procedure directly addresses this challenge by exploiting the rarity of coughs.

In-the-wild data collection studies introduce many privacy concerns. In particular, audio, although a rich source of information, often contains sensitive information that impacts the privacy of the participants. Motti and Caine [75] showed that the users of wearable devices were concerned about data recording and sensitive information. While collecting raw audio was necessary for obtaining labeled data for our models, we have shown that the trained model can run directly on the smartwatch, eliminating the privacy concerns of uploading audio recordings to a remote server. Alternatively, running the model on the smartphone is a middle ground between running on a remote server and the smartwatch that has

the privacy benefits of keeping participant data on the participant's device. Finally, Larson et al. [55] and Liaqat et al. [60] have proposed solutions for privacy-preserving cough detection, when running the model on a remote server is essential.

4.8 Conclusion

Our goal was to build a practical cough detection system. To achieve this goal, we built and deployed a system that collects smartwatch data from real participants. We designed an improved annotation process that allows us better annotate this data, which in turn allowed us to train a better cough detection model. We showed that the model resulting from this system has a 5.7 to 6.7 times higher F_1 score than existing systems. Closing the loop, we show that running this model on a smartwatch is feasible in terms of battery life and compute requirements.

Chapter 5

A Method for Preserving Privacy During Audio Recordings by Filtering Speech

5.1 Introduction

Smartwatches contain numerous sensors and can be a rich source of information from which many details about their users can be inferred. Compared to smartphones, which may be in a user's pocket, handbag or on a table for portions of the day, a smartwatch is much more likely to be on the user's wrist. This means data from sensors will be more reflective of the user's physical and physiological state. Additionally, smartwatches are much easier to use day after day as compared to other types of wearable devices (e.g., a chest belt or sensors embedded in clothing). These attributes make smartwatches an exciting research tool for conduct sensing studies.

This is reflected in recent literature as smartwatch based sensing has received much attention. Most of these works tend to focus accelerometer and gyroscope. These motion sensors have been used to detect falls [92], infer respiratory and heart rate [38] and even detect shopping related hand movements [85]. While mobile researchers have been making good use of motion sensors, the microphone generally receives much less attention. This is despite significant work done by the audio processing community that shows that audio can be used for emotion recognition [110], detecting depression [27, 97] and even estimating children's age, weight and height [57]. Audio sensing with mobile and wearable technology is less prevalent due at least in part to the privacy concerns associated with recording audio and speech. This is especially true for in-the-wild studies where users are expected to go about their lives while wearing a recording device.

A likely example of such a study is a study that monitors patients with lung disease. Audio can provide information about the patient's breathing, coughing and wheezing. However, there are a few challenges associated with recording audio. Since speech is generally considered sensitive, getting ethics approval can be more challenging if raw speech is recorded. Additionally, depending on local laws and regulations, audio recording may not even be legal if third party (i.e. not the participants) speech is recorded. Even after addressing these challenges, another significant hurdle lies in finding participants who are comfortable wearing a recording device. Our proposed method aims to ease the privacy concerns of patients by filtering speech and ultimately, increase participation rates.

Some studies that record audio, such as [55], mitigate the privacy concerns by finding and recording only features that are sufficient for their specific objective (detecting coughs) but not complete enough to re-create speech. Others, such as [98], build a classifier for their objective (detecting respiratory sounds), and throw away any audio that the classifier does not identify as a respiratory sound. While these approaches can work well when the objective is well defined, they can result in lost data if the objective changes. As an example, say we are interested in monitoring people's coughs over time. We collect examples of coughs and use them to build a classifier or identify features relevant to cough detection. We then deploy a study with real participants where we use the classifier or identified features according to the schemes proposed in [55] or [98]. This gives us data about the participants coughs over the duration of the study. However, after data collection, we realize that detecting wheezing would have been extremely useful. But since we discarded data that wasn't relevant to coughs, we can no longer identify wheezing sounds in the collected data.

Our proposed solution does not suffer from this problem. They key idea is to apply a filtering algorithm to the audio that strips away sensitive information. In our implementation, we consider speech to be sensitive and use a speech obfuscation algorithm proposed by [10] to render speech unintelligible. After applying this filtering, the filtered audio signal is saved and can further analyzed later on. So after collecting data, if a researcher realizes they are interested in detecting a new sound in the audio, they

have the option of training a new classifier that can operate on the previously collected data.

The main contributions of this chapter are (1) a method for working with audio data that outlines at which stage audio filtering and machine learning should be applied and (2) a demonstration of the method using a filter that renders speech unintelligible and cough detection as the machine learning task.

In the rest of the chapter, we discuss and compare related works in Section 5.2. In Section 5.3 we describe our method in more detail. Next, in Section 5.4 we describe our experimental setup and present our results. This is followed by a discussion of the method and results in Section 5.5.

5.2 Related Works

Klasnja et al. [50] conducted interviews with 24 participants and found that while none of them objected to having accelerometer/gyroscope data recorded, only two (8.3%) were comfortable with audio recordings. The researchers also presented the idea of recording only specific audio frequencies that could be used for activity detection. This increased the comfort level of recording audio and 4 additional participants (6/24, 25%) said they would be willing. This shows that audio filtering methods that preserve some privacy could be used to increase participation rates. Additionally, it may be the case that with a demonstration of the filtering, participants could get a better understanding of "filtering" and further increase participation rates.

As mentioned in the introduction, [55] suggested extracting features from audio and throwing away the original audio. The challenge in using this approach in a study is that these features need to be defined before running the study. These features need to be specific enough to the objective that accuracy can be maintained but not general enough that speech can be reconstructed from the features. Identifying this set of features is in itself a challenge, and it has the additional downside that collected data cannot be used for other purposes. The idea in [98] was similar, except raw audio was retained for segments that contained sounds of interest and all other audio was discarded. Again, a classifier needs to be trained and tested to detect sounds of interest before conducting the study and the collected data cannot be used for other purposes. [89] also had a related suggestion, but described a general way of white-listing events of interest for arbitrary sensors.

A different approach to preserving audio privacy was taken by [52]. Their objective was to detect long lasting states such as whether the user was in a car. To achieve this, they down-sampled audio frames and randomized their order. While this can work well for long lasting activities where multiple consecutive frames will be similar, it is likely to miss shorter events such as coughing or sneezing.

5.3 System Design

The setup of the proposed system includes a mobile device (smartwatch, smartphone or other wearable device) that records audio from the microphone. Recorded data is filtered using the algorithm presented in [10] and then uploaded to a remote server for storage and analysis. This flow is shown in 5.1.

The filtering algorithm operates on 30 ms audio frames (with 10 ms step) and for each frame computes Linear Predictive Coding (LPC) coefficients, gain and whether or not the frame was voiced. Voiced in this context means the vocal chords were vibrating to produce the sound so the audio signal is periodic. This generally means that voiced sounds are vowel sounds. Since vowel sounds are more common in speech

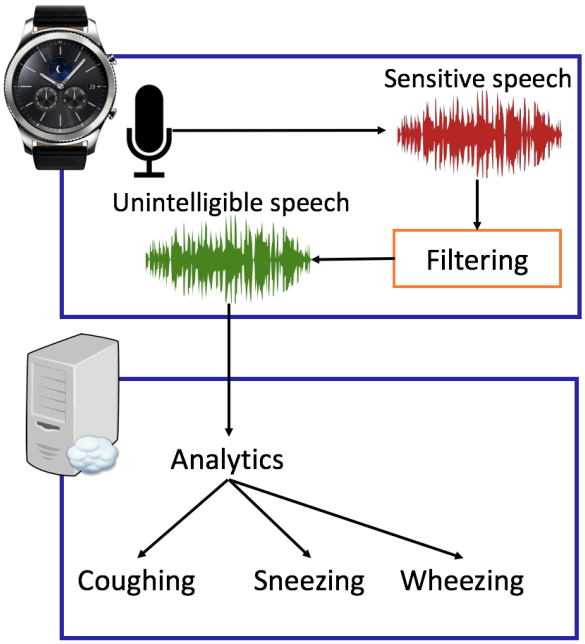


Figure 5.1: Architecture of the proposed system

than environmental sounds, this algorithm distorts speech to a greater degree than other sounds [10]. Using the LPC coefficients and gain, the original audio signal can be reconstructed. However, for frames that are voiced, the LPC coefficients are replaced with a randomly chosen set of coefficients for pre-recorded vowel sounds. The result is that voiced segments are randomized, making speech unintelligible. Chen et al. reported 7% word recognition rate in listening tests using this randomization approach.

There are two approaches to conducting a monitoring study. The more versatile but less privacy sensitive way is collect raw, unfiltered audio from the target population during an in-the-wild deployment. Afterward this deployment, audio can be manually annotated to create training data for supervised learning or a bench mark for unsupervised learning or signal processing based detection method. The other approach, which is more limited but more privacy sensitive, is to first run a small scale deployment in order to build a classifier or identify features for the desired detection objective. Our proposed privacy preserving mechanism aligns more closely to the second approach. However, instead of retaining only relevant features or segments of audio containing events of interest, the full, albeit filtered, audio is stored. This approach can be used in a study as follows. Filtered audio from the target population is collected during an in-the-wild deployment. Either before, during or after this deployment, a smaller data set is collected in the lab. This data can be be filtered or unfiltered depending on how the data is to be annotated. If the timestamp and label of events is recorded as they occur, then the data can be filtered before its saved. However, if events of interest will be listened to and annotated after the fact, then the data must be unfiltered and will be filtered after manual annotation is complete. Using the annotations and filtered audio, a classifier or algorithm to develop events of interest is created. This classifier can now operate on data that was collected during the deployment. Since this in-lab component is independent of the wild deployment, researchers can conduct multiple in-lab studies to add additional events of interest that they wish to detect.

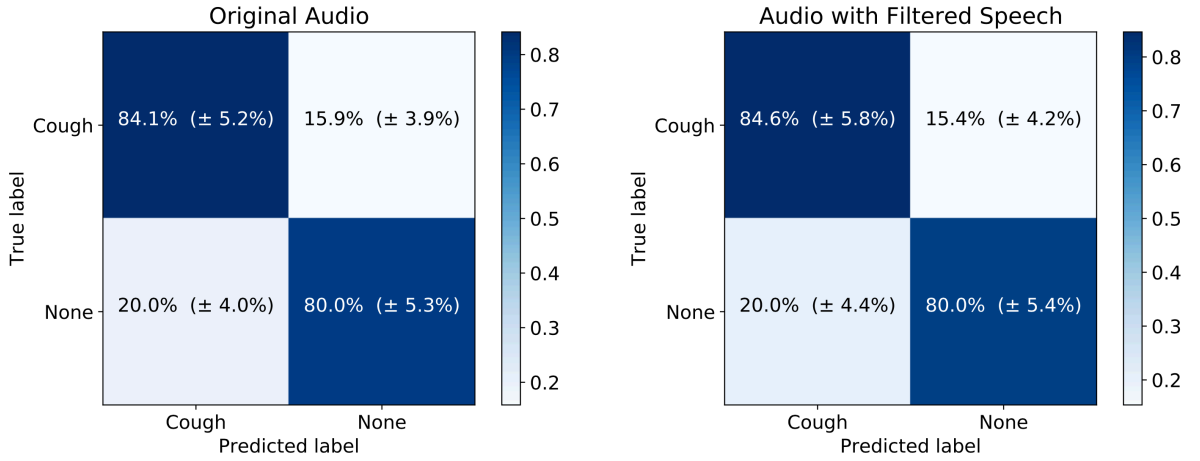


Figure 5.2: Confusion matrix showing classification performance of cough detection in raw and filtered audio

5.4 Evaluation

To evaluate our method of preserving audio privacy, we ran a data collection study with 20 participants. Ethics approval for this work is not applicable because only sound samples were recorded from participants who gave informed consent. We obtained lab approval and legal approval from Samsung Research America, Inc. to collect the sound samples and to conduct the study. Prior to participation, each participant provided informed consent to be recorded and have their sound samples used as part of the study. Participants were equipped with a Samsung Gear S3 smartwatch that collects audio data at 8 KHz. Data was collected from participants in pairs to reduce the time needed to run the study. Each pair was instructed to sit, stand and walk around while having a conversation with each other and two researchers. They were also instructed, prior to collecting data, to voluntarily cough every once in a while. This resulted in just under 9 hours of audio and 205 examples of coughing episodes (a coughing episode can contain more than one cough). During the data collection, the two researchers used an Android application to mark down the rough timestamps for when coughs occurred. These timestamps were later manually converted to intervals indicating the start and end of the cough episode using Audacity.

Audio was pre-processed with OpenSMILE [22] to extract features. Using a 1 second window with a 0.5 second step, spectral features, zero crossing rate, signal energy and Mel-Frequency Cepstral Coefficients were extracted. These features were then used to train a random forest classifier. Feature extraction and classifier training/testing was done for both the raw audio and audio that was filtered using the privacy preserving method. As shown in the confusion matrices in Figure 5.2, the filtering method had very little impact on classification. The matrices show the mean and standard deviation over 200 iterations of monte carlo cross validation. Overall, classification on raw audio had a mean accuracy of 75.86% and 75.75% on filtered audio, with a t-test p-value of 0.985 not showing any significant difference between classification on raw audio and unfiltered audio. Since cough detection is not the goal of this work, our approach to cough detection is fairly simple and the accuracy of the cough detector could likely be improved by using smarter feature selection and more sophisticated models.

5.5 Discussion

The key idea presented in this chapter is a method for preserving privacy when recording audio for in-the-wild sensing studies. The proposed method applies a transformation to the audio signal that filters out sensitive information, such as speech, before the signal leaves the device where it was recorded. While we presented one filtering algorithm, other filters are possible. There are two trade-offs to consider in selecting a filter.

Firstly, the effectiveness of the filter needs to be evaluated. An ideal filter would strip all sensitive information, but leave other sounds intact. We decided that speech is a sensitive signal and used a filter that aims to render speech unintelligible. However, the filter could consider other sounds (ex. the sound of slot machines at a casino or the sound of a cigarette lighter) to be sensitive and filter those sounds out of the signal as well. One novelty of this idea is that most existing work creates a white-list of sounds that should be preserved while this work creates a black-list of sensitive information that should be filtered. One of the reasons for this is that creating a perfect "black-list" is an impossible task because there can always be some context where a seemingly common sound could be considered sensitive. However, we argue that speech is what most people would consider sensitive and filtering speech could go a long way in preserving the privacy of study participants and ease some of the concerns of potential participants. Another example of a speech filter is a system that uses Voice Activity Detection (VAD) to determine audio segments that contain speech and just zeros out those segments. A more complex approach could be to detect only speech of second or third party individuals and filter out that speech. This could be useful if speech of the participant is of interest.

The second consideration is battery life. Since this filter should be running on a mobile or wearable device, an ideal filter should be computationally simple and low-cost in terms of energy consumption.

With these trade-offs in mind, designing and evaluating different filters could be an interesting avenue for future research. However, one characteristic to consider is how these filters affect non-sensitive sounds. Non-sensitive sounds should, ideally, be left as-is, which would be the case in a VAD based filter. Or, as is the case in our implementation, non- sensitive sounds should be transformed in a somewhat predictable way that does not alter what makes the non-sensitive sound unique. The characteristic sound of coughing is generated by the explosive phase of the cough, which is unvoiced. Because of this, our cough detection works well despite the filtering algorithm.

5.6 Conclusion

In this chapter, we presented an method for preserving privacy in audio recordings. The key idea behind the method is filtering sensitive information in the audio signal. In our setup and evaluation, we use a filter that makes speech unintelligible, and show that cough detection is unhindered by the filtering algorithm. The benefit of our proposed solution is that audio is retained and after recording, additional analysis can be performed to extract more useful information from the data.

Chapter 6

Sidewinder: An Energy Efficient and Developer Friendly Heterogeneous Architecture for Continuous Mobile Sensing

6.1 Introduction

Modern smartphones are fitted with a wide assortment of sensors. A typical smartphone, such as the LG Nexus 5, contains an accelerometer, barometer, compass, gyroscope, proximity sensor, ambient light sensor, hall effect sensor, geo-spatial positioning sensors (GPS, GLONASS, Beidou), a microphone and two cameras. Despite having such a rich array of sensors, most of them go unused most of the time. The way smartphones are typically used is that the user will open an application such as a game, news or social media application, interact for a few minutes and then put their device to sleep. Taking advantage of this usage pattern, mobile processors have been heavily optimized to rely on a low-power sleep state when not in use in order to improve battery life.

While this approach has fared well with how devices are currently used, it falls short in the face of emerging continuous sensing applications, examples of which range from context-aware applications [3, 39], such as medical applications that improve our well-being [35, 82, 103] to applications that use participatory sensing to get a better understanding of the physical world, such as noise pollution monitoring [69, 70], traffic prediction [40] or earthquake early warning [72]. While the processing demands of these applications are modest most of the time, they require continuous collection of sensor data, which prevents the processor from entering its' low-power sleep state, resulting in poor battery life and ultimately, a slow emergence of continuous sensing applications.

It is generally accepted that the solution to this problem lies in a heterogeneous architecture where one or more low-power, peripheral processors (known as a sensor hub) collect and process sensor data while the main processor is in its sleep state. When the low-power sensor hub detects an event of interest, it wakes up the main processor, allowing for further processing of sensor data. The programming model for this heterogeneous architectures remains, however, an open research question. On one end of the spectrum, researchers have proposed a **fully programmable model** [64, 83, 96] that allows application developers to write arbitrary code that runs on the low-power processor. While this would provide potentially great flexibility and power savings, there are many drawbacks. Application developers would not only have to be familiar with programming the low-power processor, they would have to account for hardware differences between devices. It is also unclear how this model would support different applications that are running concurrently. On the other end is **predefined activity**, an approach where hardware manufacturers provide a low-power processor which is hardwired to detect a few specific events. Both Apple[1] and Android[2] provide frameworks for detecting predefined activities such as significant motion and steps. Another example is the Motorola Moto X smartphone that has a dedicated Natural Language Processor[3] which is used to wake up the device when the user says a certain phrase such as "OK Google Now". Such frameworks are very easy to use from a developer's point of view and provide significant energy savings, but they are very limited because they only allow detection of events that have been pre-programmed into the device by the manufacturer.

This chapter presents *Sidewinder*, a new approach for continuous mobile sensing that splits the work of energy-efficient event detection between the platform and the application developer. With Sidewinder, the platform implements common sensor data processing algorithms (e.g., windowing, noise reduction, feature extraction, admission control) that execute on a low-power sensor hub, and application developers construct custom wake-up conditions by linking together and parameterizing these sensor data processing

[1] `https://developer.apple.com/library/ios/documentation/coremotion/reference/coremotion_reference/index.html`

[2] `http://developer.android.com/guide/topics/sensors/sensors_motion.html`

[3] `http://www.motorola.com/us/X8-Mobile-Computing-System/x8-mobile-computing-system.html`

algorithms. The custom wake-up conditions execute on the low-power sensor hub and, when events of interest are detected, the main processor is woken up and the rest of the application code is invoked.

While heterogeneous architectures have been used previously, the innovation in Sidewinder is the collaborative approach where the platform provides sensor processing algorithms and developers parameterize and chain these algorithms together to create wake-up conditions. Processing algorithms are written natively for the low-power sensor hub so application developers do not need to worry about writing code for the underlying heterogeneous architecture.

To evaluate the benefits of Sidewinder, we developed applications that use accelerometer readings or audio data to detect several events of interest. Additionally, we built a prototype implementation that extends a Nexus 4 phone with a low-power sensor hub. To enable us to conduct controlled and repeatable experiments, we mounted our prototype on a robot. Simulations conducted on accelerometer and audio traces collected in various environments show that Sidewinder can reduce the average energy required to run continuous sensing applications by up to 96% compared to keeping the phone awake at all times, while matching the detection recall and precision of the always on approach. Moreover, for most of our usage scenarios, Sidewinder achieves over 90% of the power savings achieved by a "perfect" wake-up mechanism, indicating that an implementation that supports custom code offloading will achieve only marginal additional improvements.

The rest of this chapter is organized as follows. Section 6.2 describes the Design of Sidewinder and 6.3 describes how we implemented the design. Sections 6.4 and 6.5 present our evaluation method and results. Finally, Sections 6.6 and 6.7 describe our work in the context of related work and conclude the paper.

6.2 Design

Continuous mobile sensing approaches have to address two main constraints: maximizing detection accuracy and minimizing energy consumption. Users expect high precision and recall and user experience is adversely affected when the application misses or over-reports events of interest. Achieving both high recall and precision requires highly specialized algorithms tuned to the event of interest. Specifically, our experience suggests that getting the last few percentage points in precision and recall is difficult and requires complex algorithms and fine parameter tuning. Additionally, these complex algorithms typically run on a fully featured processor, which is detrimental to battery life since a large portion of energy savings comes from keeping the main processor in a sleep state.

We achieve our two main goals (energy efficiency and high precision/recall) by using a multi-stage pipeline of algorithms to create a more complex classifier. Earlier stages are used to achieve the majority of available energy savings and later stages can then optimize for high detection precision and recall. For example, a voice recognition algorithm may have three stages, the first stage determines if microphone data contains sound, the second stage determines if the sound is human speech and the third stage converts the speech to text. The first two stages are relatively simple and can be run on a less powerful, more energy efficient processor. They have high recall but may have low precision (i.e. they will let through a high proportion of events that contain speech, but not all events passed through will contain speech). Since the first two stages are run on a low-power processor and reduce the amount of time the main processor has to be awake, this will result in an overall saving of energy.

Based on this observation, Sidewinder encourages continuous mobile sensing applications to be struc-

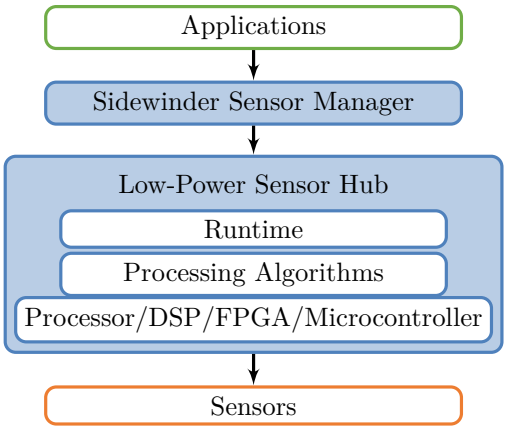


Figure 6.1: Proposed system architecture. The sensor manager is part of the OS, and the Sensor Hub and Sensors are hardware provided by the manufacturer

tured as pipelines of processing algorithms of increasing complexity: simple yet high recall, moderate precision algorithms that run continuously on a low-power sensor hub, providing energy efficient wake-up mechanisms for higher complexity algorithms that run on the main CPU and provide both high recall and high precision.

To facilitate the creation of wake-up conditions capable of running on a low-power sensor hub, Sidewinder provides a set of commonly used sensor data processing algorithms that are ready to run on the sensor hub. These algorithms can be parameterized and chained together to create wake-up conditions.

We conjecture: 1) that it is possible to implement custom wake-up conditions for a wide range of applications by configuring a small set of common processing algorithms; and 2) that this approach will achieve comparable energy savings to an alternative implementation that supports full programmability.

6.2.1 Sidewinder

Sidewinder is a new approach for continuous mobile sensing that divides the responsibility of energy-efficient event detection between the manufacturer and the application developer. The manufacturer provides a low-power sensor hub and implements common sensor data processing algorithms that execute on the sensor hub. Applications construct custom wake-up conditions for events of interest in their application code. These wake-up conditions are then pushed to and executed continuously on the low-power sensor hub and, when events of interest are detected, the main processor is woken up and the application code is notified.

Figure 6.1 shows the architecture of a system that uses Sidewinder. Applications interact with a sensor manager to define a custom wake-up condition. The manager contains an API for parameterizing and chaining algorithms that are available on the low-power sensor hub. Developers can use this API to create their wake-up conditions. The low-power sensor hub contains commonly used algorithms for windowing, filtering, transformations, feature extraction and admission control. Once configured by the developer, the wake-up condition is converted into an intermediate language by the sensor manager and pushed to a runtime or interpreter on the low-power sensor hub. When running on the sensor hub, the custom wake-up condition wakes the main CPU if an event of interest occurs.

We next describe the components of Sidewinder based on whether their implementation is the responsibility of the device manufacturer, the operating system, or the application developer.

The Manufacturer

The manufacturer is responsible for providing the hardware and software of the low-power sensor hub. The hardware could be a network of one or more processors, Digital Signal Processors (DSP), FPGAs or microcontrollers. For example, there could be one larger processor to handle all sensors and algorithms or a DSP for the microphone and an FPGA for each of the other sensors. The manufacturer also needs to provide a runtime to manage this hardware and an implementation of common sensor data processing algorithms. The runtime needs to be able to receive wake-up condition configurations from applications, configure the hardware and algorithms, execute the resulting wake-up condition and notify applications when an event of interest occurs.

The runtime could use an interpreter approach where it executes each algorithm, a compiler approach where it generates an executable or it could reconfigure FPGAs according to the requirements of the wake-up condition and the hardware available. In the case of FPGAs the algorithms will most likely be pre-compiled and the runtime would need to reconfigure according to the specific configuration.

The runtime provides a decoupling layer between the mobile platform (Android, iOS, etc.) and the hardware since the runtime is responsible for managing the hardware. And since the manufacturer will be providing both the runtime and hardware, any architecture for the low-power sensor hub can be used.

The Application Developer

The application developer creates custom wake-up conditions for their event(s) of interest. One important aspect of Sidewinder is that common sensor data processing algorithms are provided to the developer by the platform. This means for example, if the developer needs to use an FFT, they do not need to implement it themselves or find a library. Instead, they would use the system API to create a wake-up condition that uses an FFT. The API allows developers to parametrize the FFT and if needed, chain it with other algorithms together to create more advanced wake-up conditions. They can then use the API to push their wake-up condition to the low-power sensor hub.

Because wake-up conditions are defined by configuring generic algorithms designed to support a large set of applications, as opposed to writing custom code specific to any application, their performance may be suboptimal by design. To ensure that user experience is not adversely affected when the application misses events of interest, application developers should create conservative wake-up conditions that provide for high recall at the expense of lower precision. This approach will ensure that no events of interest are overlooked, but will result in some unnecessary wake-ups, i.e., false positives. Therefore, to ensure that the application does not adversely affect user experience by over reporting, additional filtering needs to be executed on the main processor on a wake-up event to eliminate any false positives. In Section 6.5, we show that while the moderate precision of wake-up conditions does result in additional energy use, the approach is nevertheless able to achieve 90% of the power savings achieved by an “ideal” wake-up mechanism.

The Operating System

The operating system needs to provide the API (part of the Sensor Manager in Figure 6.1) that allows developers to create wake-up conditions and push/receive data to/from the low-power sensor hub. A wake-up condition is pushed to the sensor hub in the form of an intermediate code to decouple the platform from the hub. The operating system will also need to provide a driver to allow communication with the hardware. Depending on the mobile platform (Android, iOS, Windows 10 Mobile), it is likely that the manufacturer will provide the driver.

6.2.2 Advantages

Sidewinder has many benefits:

Lower programming complexity. Programming complexity is decreased because application developers can use the predefined processing algorithms, rather than implementing their own. The intermediate language makes Sidewinder language independent so that developers can write their classifier in the same language as their application.

Better optimization. The hardware and software of the low-power processor is implemented by the manufacturer, allowing for much greater optimization by experts.

Better security. Providing access to these algorithms via an API has significant security advantages over the fully programmable offloading approach because application developers cannot execute arbitrary code on the low-power processor.

Improved portability. Programmers do not have to be aware of the specifics of the underlying hardware, nor create a version for every type of platform. Manufacturers are free to use any type of hardware they want (processor, DSP, FPGA or networks of processors/DSPs/FPGAs) as long as it can interpret and execute the intermediate language.

6.3 Implementation

In this section we outline our implementation of the components mentioned in the design section. We also describe the applications we developed to evaluate Sidewinder. We implemented Sidewinder on the Android platform. It is important to note that there are many different valid implementations of our design, ours is just an example of a valid implementation.

6.3.1 Sensor Manager

The Sidewinder sensor manager is based off the Android sensor manager [4]. It contains information about the available sensors and processing algorithms and gives developers access to them via the API.

6.3.2 API

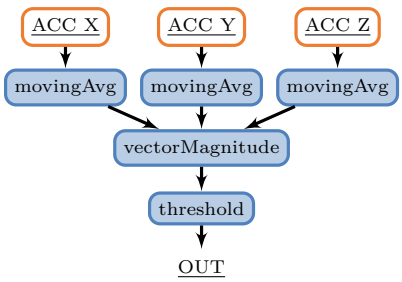
The API allows developers to create wake-up conditions. It contains four major components:

- *ProcessingPipeline*. This represents the entire wake-up condition from the input sensors to the final output. The pipeline consists of one or more processing branches.

[4] `https://developer.android.com/reference/android/hardware/SensorManager.html`

```java
public class SignificantMotion implements SensorEventListener {

    public void createClassifier() {
        ProcessingPipeline significantMotion = new ProcessingPipeline();

        ProcessingBranch[] branches = new ProcessingBranch[3];
        branches[0] = new ProcessingBranch(SidewinderSensorManager.ACCELEROMETER_X);
        branches[1] = new ProcessingBranch(SidewinderSensorManager.ACCELEROMETER_Y);
        branches[2] = new ProcessingBranch(SidewinderSensorManager.ACCELEROMETER_Z);

        branches[0].add(new MovingAverage(10));
        branches[1].add(new MovingAverage(10));
        branches[2].add(new MovingAverage(10));

        Algorithm vm = new VectorMagnitude();
        Algorithm ac = new MinThreshold(15);

        significantMotion.add(branches);
        significantMotion.add(vm);
        significantMotion.add(ac);

        SidewinderSensorManager sManager = (SidewinderSensorManager) getSystemService(
                SENSOR_SERVICE);
        sManager.push(significantMotion, this);
    }

    @Override
    public void OnSensorEvent(SensorData data) {
        // Do something with data
    }
}
```

(a) Java representation



(b) Conceptual representation

```
ACC_X -> movingAvg(id=1, params={10});
ACC_Y -> movingAvg(id=2, params={10});
ACC_Z -> movingAvg(id=3, params={10});
1,2,3 -> vectorMagnitude(id=4);
4 -> minThreshold(id=5, params={15});
5 -> OUT;
```

(c) Intermediate representation

Figure 6.2: Various representations of a Significant motion pipeline

- *ProcessingBranch*. Branches represent the flow of data from either a sensor to an algorithm or between two algorithms. At the start of the classifier pipeline, there may be any number of branches, each receiving data from any of the available sensor channels. At the end of the pipeline, there must be only one branch remaining. This means that if the pipeline contains multiple branches, aggregation algorithms need to be used to reduce the number of branches until a single branch is left.

- *Algorithm*. An algorithm is some operation that accepts one or more branches and produces one branch. For example, a moving average or admission control (threshold) algorithm accepts one branch and produces one branch. A vector magnitude algorithm accepts one or more branches and produces one. At the API level, these algorithms are simply stubs that represent the algorithm implantations at the low-power processor level.

- *SensorEventListener*. This is the Android SensorEventListener. It is a callback method that is registered with the sensor manager that will be called when the custom wake-up condition is satisfied.

An example of a significant motion wake-up condition is given in Figure 6.2a. A conceptual diagram of the condition is given in Figure 6.2b. First a ProcessingPipeline object is created. Next, three branches, one for each axis of the accelerometer are created and each branch is given one axis as its source. Then, three MovingAverage algorithms are created, each with window size of 10, and one MovingAverage is added to each of the branches. A VectorMagnitude object and MinThreshold object are also created. The order in which these algorithms and branches are added to the ProcessingPipeline specify how they are chained together. Since the MinThreshold is the last algorithm in the pipeline, if it produces any result, the callback method will be invoked. Now that the pipeline is configured, it, along with a SensorEventListener, is pushed to the SidewinderSensorManager.

6.3.3 Intermediate language

The intermediate language allows decoupling between the sensor manager and the low-power processor implementation. Upon receiving a wake-up condition configuration, the sensor manager generates its associated intermediate code. The intermediate code for the significant motion wake-up condition is shown in Figure 6.2c. Having the intermediate code allows developers to write their conditions in the same language as their application. In the intermediate code, each algorithm has a unique ID (generated by the sensor manager). In the example, the moving average algorithms are given IDs from 1, 2 and 3. Then the vector magnitude is setup to receive data from algorithms 1, 2 and 3 and the result of the vector magnitude is given to the admission control algorithm. Finally, the admission control algorithm is fed to OUT. A value being sent to OUT indicates that an event of interest has occurred and the main processor should be woken up.

6.3.4 Hardware

Our prototype is built around a Google Nexus 4 phone running Android 4.2.2. Since the Nexus 4 does not have an easily programmable sensor hub built in, we implemented our low-power sensor hub using a Texas Instruments (TI) MSP430 or LM4F120 microcontroller attached to an accelerometer sensor and a

microphone. We chose to focus our efforts on these sensors because in our experience they are the most commonly used.

The Nexus 4 and microcontroller communicate over the UART[5] port made available by the Nexus 4 debugging interface via the audio interface jack. The serial connection provides sufficient bandwidth to support low bit-rate sensors, such as the accelerometer, a microphone or GPS. However, extending the prototype to work with higher bit-rate sensors like the camera would require a higher bandwidth data bus, such as I^2C [6].

6.3.5 Runtime

The main responsibility of the Sidewinder runtime is to execute the intermediate language. In this regard, the implementation of the runtime is very flexible. Our implementation of the runtime resembles a simple interpreter (written in C). It contains implementations of algorithms and a list of all available algorithms. Upon receiving a new configuration, the runtime allocates memory for each algorithm in the configuration. The interpreter then waits for sensor data to be available and feeds the data into the appropriate algorithm. If the algorithm produces a result, it sets a flag. The interpreter checks the flag and if necessary sends the result to the next algorithm. The flag is required because some algorithms may not always produce a result. A moving average with a window size of N will not produce a result until it has received N data points and a threshold will only produce a result when the threshold is met. The final algorithm feeds into OUT, indicating that the main processor should be woken up.

6.3.6 Processing Algorithms

Our algorithms are written in C and packaged with the runtime. Each algorithm operates on its own instance of a data structure. The data structure is created by the runtime and stores the algorithm ID, type, size, data, whether a result is available and the result. It can also contain any other data needed by the algorithm. Each time the algorithm needs to be run, the interpreter invokes the algorithm and passes it its data structure. The algorithm operates on the data available in the structure and, if required, stores the result in the structure and sets the hasResult flag. We implemented the following algorithms:

- **Windowing** Partitioning sensor data into rectangular or Hamming windows.
- **Transform**
  - Fast Fourier Transform (FFT) from time-domain to frequency-domain
  - Inverse Fast Fourier Transform (IFFT) from frequency-domain to time-domain.
- **Data Filtering**
  - Noise-reduction algorithms such as a moving average and exponential moving average.
  - FFT-based low-pass filtering.
  - FFT-based high-pass filtering.
- **Feature Extraction**

[5] https://en.wikipedia.org/wiki/Universal_asynchronous_receiver-transmitter
[6] https://en.wikipedia.org/wiki/I²C

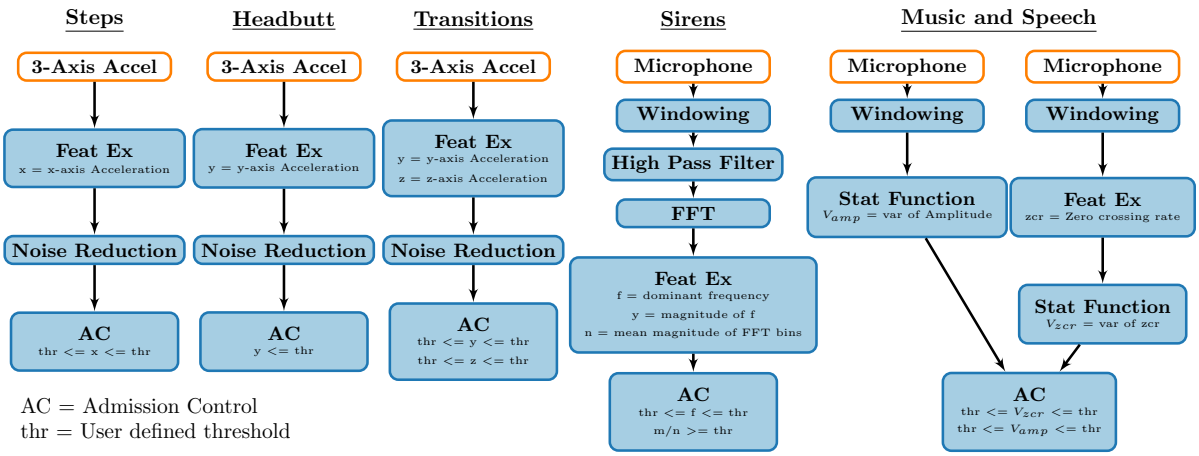


Figure 6.3: Wake-up conditions pipelines for each of the applications.

  - Magnitude of acceleration vector computation.
  - Zero Crossing Rate computation.
  - A set of statistical functions.
  - Determination of magnitude of dominant frequency.

- **Admission Control** Configurable high or low thresholds.

6.3.7 Applications

We developed six applications to run on the mobile device and, for each of the applications, we constructed a wake-up condition using the algorithms presented in Section 6.3.6.

Accelerometer Applications

We decided to use a robotic dog to conduct controlled and repeatable experiments. We developed three applications that detect activities that an AIBO ERA 210 robot can perform: walking, posture transitions, and headbutts. We chose these actions because they have similar acceleration signatures to human activities. It should be noted that, although classifiers to detect these activities will be similar (i.e. similar sensors used, similar algorithms and order of algorithms), each classifier requires its algorithms be parametrized specifically to the activity. This reinforces that a small set of algorithms can be used for numerous classifiers based on how they are chained and parameterized.

A walking robot has a similar acceleration signature as a human, though at a lower intensity. The headbutts are meant to represent very infrequent human actions such as falling. We found that robot stance transitions between the normal and sitting postures are very similar in their acceleration signature to humans sitting down and standing up. In Section 6.5 we show that the energy saving measured in our experiments with the robot approximate closely the results of experiments conducted on limited traces collected from human subjects.

Steps Counts how many steps the robot takes when it walks. The algorithm is based on the human step detection algorithm proposed by Ryan Libby [62]. The application takes in raw accelerometer

readings and applies a low-pass filter on the x-axis acceleration. It then searches for local maxima in the filtered x-axis acceleration. Local maxima between 2.5 m/s^2 and 4.5 m/s^2 are detected as steps.

Transitions Detects transitions between sitting and standing. The application monitors changes in acceleration due to gravity on the y and z axes to determine the orientation of the device. If the z-axis (up-down relative to the dog) acceleration is between $9m/s^2$ and $11m/s^2$, and the acceleration on the y-axis (front-back relative to the dog) is between $-1m/s^2$ and $1m/s^2$, the device is in a horizontal position and the robot is assumed to be in a standing posture. Similarly, if the z-axis acceleration is between $7.5m/s^2$ and $9.5m/s^2$, and the acceleration on the y-axis is between $3.5m/s^2$ and $5.5m/s^2$, the device is in an angled position and the robot is assumed to be in a sitting posture. The application detects transitions by looking for posture changes.

Headbutts Detects a sudden forward head movement. The application monitors the y-axis acceleration and searches for local minima between $-3.75\ m/s^2$ and $-6.75\ m/s^2$.

Audio Applications

We developed the following three microphone based applications.

Siren Detector Detects sirens originating from emergency vehicles. The application applies a 750 Hz high-pass filter in order to remove a significant potion of sounds that aren't sirens. The data in each window is transformed to the frequency domain using a FFT in order to extract the magnitude of the dominant frequency and the mean magnitude of all frequency bins. The ratio of the magnitude of the dominant frequency and the mean frequency is used to determine if the window contains pitched sounds. Pitched sounds between 850 Hz and 1800 Hz that last longer 650 ms are classified as sirens.

Music Journal Creates a list of all the songs heard during the day using the web services provided by Echoprint.me [7]. Audio data is partitioned into windows and passed to two branches for feature extraction. The first branch computes the variance of the amplitude over the entire window. The second branch further partitions the data into smaller windows and computes the zero crossing rate (the rate at which the signal changes from positive to negative or vice versa) for each sub-window. It then calculates the variance in zero crossing rate across the set the sub-windows. Finally, an admission control step uses thresholds (different for music and speech detection) on the extracted features to determine if an event of interest has occurred. Data is then passed to the Echoprint.me web service to identify the song.

Phrase Detection Similar to Music Journal, except different parameters are used in the wake-up condition and Google Speech API was used for speech-to-text translation.

We created a wake-up condition for each of these six applications using the processing algorithms described in section 6.3.6. Figure 6.3 shows the conceptual pipeline representation for each condition we constructed. Each one ends with an admission control step with configurable thresholds. The wake-up condition is satisfied when the relevant data or extracted features meet the admission control threshold.

6.3.8 Discussion

This section describes important questions that will need to be answered by hardware vendors to implement Sidewinder.

[7] `http://echoprint.me/`

Identifying processing algorithms. Defining the appropriate set of common algorithms that should be included in the API and executed on the low-power processor for each sensor is a key challenge. First, there is a trade-off between algorithm generality and accuracy. Simple generic algorithms can support a large set of applications, albeit no specific application is likely to experience optimal performance. Conversely, a highly specialized algorithm may provide optimal performance but is only applicable to a limited set of applications. Second, there is also a trade-off between algorithm complexity and power savings. More complex algorithms can reduce energy consumption by preventing unnecessary wake-ups due to increased accuracy. On the other hand, more complex algorithms have higher computational demands, which require a larger and hungrier peripheral processor.

While determining the complete set of algorithms to be included as part of the runtime is beyond the scope of this work, we anticipate that it will include algorithms for windowing, data filtering, feature extraction, admission control and transformations. Ideally, this set of algorithms should be standardized by the platform (ex. Android or iOS).

Access to sensor data. A related question is determining what data the sensor hub should pass to the application following a wake-up. Some applications may be interested in the raw sensor data, while others may want to use the filtered data or extracted features. Ideally, an API would allow developers to specify what data their application should receive when an event of interest occurs. Our current implementation passes a buffer of raw sensor data to the application.

Sizing. When creating the sensor node of the prototype implementation we evaluated two microcontrollers having different power consumption levels. We noticed that the lower power microcontroller was not able to run some algorithms (such as Fast Fourier Transforms) in real-time. Determining the optimal number, type and size of processors to include in the sensor hub is an open research question. Each sensor (or small group of related sensors) may be supported by its own dedicated low-power processor. Alternatively, a larger processor could be used to serve the entire sensor hub. Identifying a sweet spot between the maximum number of concurrent algorithm executions, energy budget, cost and physical size of the sensor node is an open challenge and is a decision the hardware manufacturer will have to make.

Sensor fusion. Fusing inputs from multiple sensors is a common technique used for improving the accuracy of sensing applications. Whether low power sensor hubs should include support for sensor fusion, however, is not clear. On the one hand, such support could increase energy efficiency by reducing the occurrence of unnecessary wake-ups. On the other, sensor fusion tends to be application specific and the added complexity may negate any energy benefits. The current implementation allows sensor fusion at the main processor level. Once a wake-up condition triggers and passes raw data to the application, the application has the ability to run sensor fusion algorithms on the data.

6.4 Evaluation

Our evaluation is based on a trace-driven simulation. We measured power usage for our hardware to create a power model and collected accelerometer and audio traces. This data was fed into our simulator which modeled the behavior and power consumption of our devices under various configurations and applications.

We power profiled the Google Nexus 4 in order to create a model to estimate power consumption based on the outputs from the simulator. The results of the power profile are summarized in Table 6.1. During all the measurements, the devices' screen, WiFi and GPS were turned off. While the device is

State	Average Power Consumption (mW)	Average Duration
Awake, running sensor-driven application	323	n/a
Asleep	9.7	n/a
Asleep-to-Awake Transition	384	1 second
Awake-to-Asleep Transition	341	1 second

Table 6.1: Google Nexus 4 power profile.

sleeping, its power usage is very low, consuming only 9.7 mW. While awake, the power consumption is significantly higher, averaging 323 mW. During our power measurements we noticed that additional energy is consumed during transitions between the asleep and awake states. Each transition takes about 1 second. During a wake-up transition, the average power consumption goes up to 384 mW, while during an awake-to-asleep transition the average power consumption is 341 mW.

We implemented the low-power processor on two different microcontrollers. One was a Texas Instruments (TI) MSP430 and the other a TI LM4F120. The MSP430 has the advantage of requiring very little power, consuming only 3.6 mW while awake. However, it has limited memory and cannot perform complex analysis of sensor data in real-time. In our tests, it was unable to run the FFT-based low-pass filter in real-time. The TI LM4F120 is powered by a Cortex-M4 processor. It can batch a higher number of accelerometer readings and can run all our filters in real time. However, this microcontroller has an energy footprint an order of magnitude greater than the MSP430, consuming an average of 49.4 mW while awake.

6.4.1 Trace Collection

Audio traces We collected three half-hour audio traces in different environments: an office, a coffee shop and outdoors. We used audio mixing software to add audio events of interest to the collected traces. The audio events of interest include music (5% of each trace), speech (5% of each trace), and sirens (2% of each trace). The events of interest were randomly selected from a library of audio files.

Human accelerometer traces We collected six hours of accelerometer traces from three different individuals while they perform routine daily activities: morning commute using public transit, working in a retail store, and working in an office. Between 20% and 37% of each trace is spent walking.

Robotic accelerometer traces We collected synthetic traces by having a robot perform multiple runs with a prototype smartphone attached to its back. For each run, the robot logged the start and end of each action, which we use as the ground truth for our experiments. The smartphone ran an application that kept the device always awake and continuously recorded accelerometer readings for all three axes.

To enable us to conduct controlled and repeatable experiments, we mounted the prototype smartphone on the back of an AIBO ERA 210 robot dog (see Figure 6.4). Because the robot's actions can be scripted, this setup provides an efficient and reliable way to determine ground truth. In contrast, labeling data collected from human subjects with ground truth is error prone and labor intensive.

In each run, the robot performed five different actions: standing idle, walking, sit-to-stand transitions, stand-to-sit transitions, and headbutts. We created runs with three different levels of activity. Runs in groups 1, 2 and 3 spent 90%, 50% and 10% of the time standing idle, respectively. The reminder of the time was allocated as follows: 73% for walking, 24% for transitions between sitting and standing,

Figure 6.4: Aibo robotic dog used for data collection

and 3% for headbutts. This setup allows us to experiment with detecting actions that are common, somewhat frequent, and rare. In total, the robot executed 18 different runs: 9 for group 1, 6 for group 2 and 3 for group 3. We generated more runs for groups 1 and 2 because of the lower activity levels compared to group 3. To eliminate bias, the list of actions was generated randomly for each run, based on the expected probabilities of each action.

While our robotic testbed allows us to run live experiments, we chose instead to use trace-based simulation for several reasons. First, it took the robot close to an hour to complete a single experiment. Secondly, a thorough exploration of the configuration space of the various sensing approaches we consider would have required months of continuous live experiments. Moreover, taking fine grain power consumption measurements while the robot is in motion is not trivial.

6.4.2 Configurations

We used the simulator to evaluate the recall and precision of our applications under the following configurations.

- **Duty Cycling** The applications wake-up at fixed time intervals to collect sensor data for 4 seconds and run the event detection algorithms. If an action is detected, the phone is kept awake for another 4 seconds, otherwise it goes to sleep for N seconds. N is referred to as the sleep interval. For our experiments, we use a sleep interval of 2, 5, 10, 20 and 30 seconds. As the sleep interval increases, more power is saved but recall suffers.

- **Batching** Similar to Duty Cycling, except when the phone is asleep sensor data is cached. When the device wakes, a batch of data from the sleep cycle is given to the application. We use the same sleep interval for Batching Cycling as we did for Duty Cycling.

- **Predefined Activity** This configuration simulates the Android's built-in significant motion detector. We constructed simple classifiers to wake up the device and invoke the callback method in the application when significant activity is detected (significant acceleration or sound).

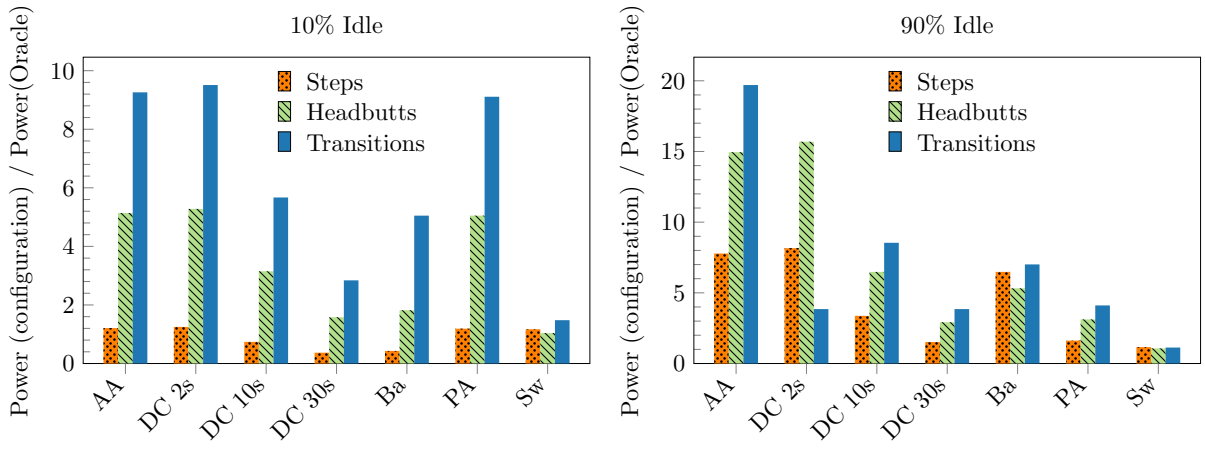


Figure 6.5: Power usage of configurations: Always Awake (AA), Duty Cycling (DC) with various sleep intervals, Batching (Ba) with 10s sleep interval, Predefined Activity (PA) and Sidewinder (Sw) relative to Oracle for synthetic accelerometer traces

- **Sidewinder based Classifier** For each of the applications, we constructed wake-up conditions to invoke the application when events of interests are detected.
- **Oracle** A hypothetical ideal implementation that only wakes up when the event of interest occurs. Such a wake-up condition would achieve perfect detection precision and recall, with the lowest possible power consumption. The difference between the power consumption of this method and the Sidewinder configuration provides an upper bound on the potential additional benefits of custom code offloading.

6.4.3 Metrics

For each sensing approach and trace, the simulator calculated the amount of sleep and awake time, the total number of wake-up events, and the recall and precision of the application. Using this data and the energy model derived from measurements of our prototype, we estimate the average power consumption. For the *Duty Cycling* experiments, the power model accounts only for the energy consumption of the Nexus 4. For *Batching* and *Predefined Activity*, the model also includes the cost of a low-power TI MSP430 microcontroller. Finally, experiments configured to use *Sidewinder* include the cost of the TI MSP430, with the exception being the siren detector which required the more powerful TI LM4F120 to run FFT in real time.

6.5 Results

In this section we present the results of simulations conducted on the accelerometer and audio traces described in Section 6.4.1. We answer the following questions:

1. How much power can be saved with more energy efficient sensing approaches?
2. How close to optimal is Sidewinder?

3. How does Sidewinder compare to Predefined Activities?

4. How well do Duty Cycling and Batching perform?

5. How representative are the accelerometer experiments on the AIBO of expected performance with humans?

Wake-up Mechanism	Sirens	Music	Phrase
Oracle	16.8	27.2	14.7
Predefined Activity	51.9	51.9	51.9
Sidewinder	63.1[8]	32.3	35.6

Table 6.2: Average power consumption (mW) for the audio applications.

Figure 6.5 presents the power usage, relative to Oracle, of replaying the synthetic accelerometer traces under the various sensing configurations. For each configuration[9], the graph presents power consumption over Oracle. Results are averages across runs of the same group. In order to make it easier to compare across approaches, we calibrated all approaches so that they all achieve 100% recall. Duty Cycling is the one approach that cannot achieve 100% recall with any reasonable sleep interval. Figure 6.6 shows the recall for Duty Cycling at 90% idle. All sensing approaches achieved similar average precision (Headbutts: 89%, Transitions: 91%, Walking: 93%).

Table 6.2 shows the average results from running the the simulations on the collected audio traces. We omitted the results for Duty Cycling and Batching because they are similar to the results from the simulations on accelerometer traces.

6.5.1 How Much Power can be Saved?

The Oracle in our work is a hypothetical ideal which is in a sleep state most of the time and only wakes up when events of interest occur. It has perfect recall, precision and the best possible power usage. Always Awake, on the other hand, never sleeps and therefore, and has the worst possible power usage. The difference in power usage between these two approaches represents the power that could be saved by better sensing approaches. Always Awake consumed on average 323mW of power. In our most demanding scenario, step detection with 10% idle, the Oracle consumed on average 266mW. For the least demanding, step detection with 10% idle, the Oracle consumed on average 16.4mW.

Conclusion: There is potential to reduce power consumption by 17.7% to 94.9% with optimal wake-ups.

6.5.2 How Close to Optimal is Sidewinder?

By comparing the performance of the Sidewinder approach to Oracle we observe that the Sidewinder approach achieves between 92.7% and 95.7% of the of the possible power savings [10] for our accelerometer-based applications. Audio applications performed similarly with saving between 85% and 98%.

[8] Includes the more powerful TI LM4F120

[9] Results for Batching are shown with a 10s sleep interval as the other results were similar to Duty Cycling

[10] $(AlwaysAwake - Sidewinder)/(AlwaysAwake - Oracle)$

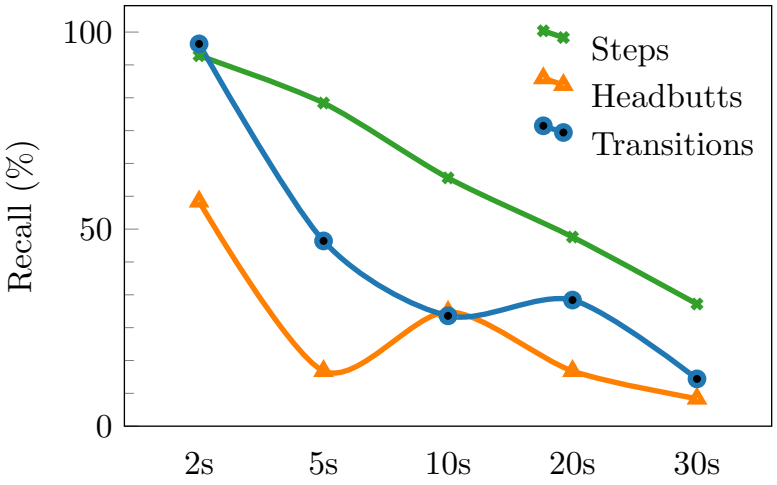


Figure 6.6: Recall for Duty Cycling on synthetic accelerometer traces with 90% idle

The suboptimal nature of wake-up conditions is illustrated by the phrase detection application. Whereas, the Oracle only wakes up when the phrase of interest occurs (<1% of each trace), our wake-up condition powers up the device every time it detects a speech segment (approximately 5% of each trace). However, even with this limitation, Sidewinder achieves 93% of the possible energy saving for this application.

Conclusion: It is possible to build a wide range of classifiers based on a set of generic processing algorithms, and that the resulting classifier achieves the large majority of available power savings. Moreover any additional power saving that custom code may achieve are likely to be very limited.

6.5.3 Sidewinder vs. Predefined Activity

To make the comparison to Predefined Activity as fair as possible, we explored the parameter space to determine the best thresholds for *significant acceleration* and *sound intensity*[11]. We chose values that minimize power consumption, while maintaining 100% detection recall. Thus the parameters used in this scenario are over-fitted to our test data and represent a best case scenario that skews the results in favor of Predefined Activity.

As expected, the power consumption resulting from the use of significant activity detectors (significant sound, significant motion) are proportional to the amount of activity in the trace and the popularity of the event of interest.

In the accelerometer experiments, Predefined Activity has similar power consumption to Sidewinder for *steps*, which is a common event, but consumes 4.7 and 6.1 times more power to detect *headbuttss* and *transitions*, which are less frequent events. In the experiments performed on the audio traces, Predefined Activity consumed 18% less power for *sirens* than Sidewinder, but 45% and 60% more power for *music journal* and *phrase detection*, respectively. Due to the higher complexity of the wake-up condition used for siren detection, the power consumption model had to account for the powerful TI LM4F120 microcontroller instead of the MSP430, which consumes an additional 40 mW.

Conclusion: a small number of predefined activities are unlikely to support efficiently a wide range of applications. This is particularly the case for applications interested in infrequent events.

[11] Two predefined activities supported by our hardware

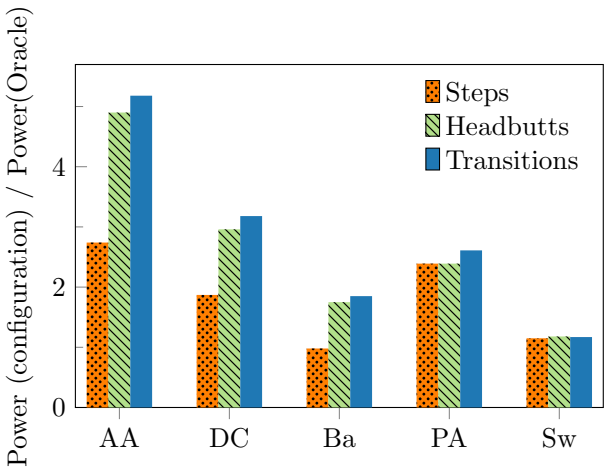


Figure 6.7: Power usage of configurations: Always Awake (AA), Duty Cycling (DC), Batching (Ba), Predefined Activity (PA) and Sidewinder (Sw) relative to Oracle for human traces

6.5.4 Sidewinder vs. Duty Cycling and Batching

Duty Cycling performs poorly. Short sleep intervals actually result in an increase in power consumption (339 mW compared to an average of 323 mW for Always Awake) due to frequent transitioning between awake and asleep states. Longer sleep intervals are more effective at saving energy, but they do so by sacrificing recall. For example, a sleep interval of 10 seconds reduces the Headbutts and Transitions recall bellow 30%.

Batching achieves perfect recall, but requires long batching intervals to achieve large energy savings. Therefore, this approach is not appropriate for applications with timeliness constrains. For example, the user of a gesture recognition application [65, 90] would not be satisfied if the application detects the performed gesture after a delay of more than a couple of seconds. We anticipate that in practice realistic batching intervals are in the order of a few seconds, depending on the sensor data acquisition rate and the size of the data buffer. Additionally, the device often wakes up to find out that no events occurred in the current batch.

Conclusion: Duty Cycling and Batching consumed 2.4 to 7.5 times more power than Sidewinder. To achieve significant power saving, Duty Cycling and Batching have to either sacrifice recall or timeliness.

6.5.5 Human Traces

Figure 6.7 shows the results from running the step detector application on traces collected from three human subjects. Since these traces are not annotated with ground truth, we use the steps detected by an *Always Awake* configuration as the baseline for determining recall. For Duty Cycling and Batching we show only a sleep interval of 10 seconds. All approaches except Duty Cycling (82%) had 100% recall.

The results from these experiments show benefits very similar to the synthetic experiments for runs with low and medium levels of activity. The Sidewinder approach achieves at least 91% of the available power saving in each of the traces.

Additionally, we note that the generic wake-up condition performs poorly. We attribute the relatively high power consumption to the fact the human subjects were performing a wide range of activities. While most of the activities were not events of interest, they resulted in unnecessary wake-ups.

6.6 Related Work

The idea of waking up a device when an event of interest occurs has been around since the inception of mobile phones. The phone's radio transceiver wakes up the device when an incoming call or a text message is received[12]. Wake on Wireless [93] extended this idea by augmenting a PDA with a low-power radio that would send a wake-up message when an incoming call is received. Similarly, Wake on WLAN [73] allows remote wake-up of wireless networking equipment.

Turducken [96] generalizes the "wake on event of interest" approach to several types of applications and to multiple components operating at increasingly small power-levels. Little Rock [83] applies Turducken's multi-tiered architecture to sensing on mobile devices. Reflex [64] complements the idea proposed by Turducken by providing a shared memory abstraction to be used by the different processors. Little Rock and Reflex expose application developers to the heterogeneous architecture. In contrast, Sidewinder hides the heterogeneous nature of the system from the application developer. Creating an application that makes use of Sidewinder does not require the developer to write low-level code for the low-power sensor hub. Instead, developers create custom wake-up conditions by configuring pipelines of commonly used algorithms. This approach increase portability, while achieving the majority of the potential power savings.

Smartphone manufacturers have started to incorporate low-power processors into their architectures, but have only implemented limited APIs that provide fixed functionality. Apple's M7 and M8 motion co-processors are used to collect, process, and store sensor data even while the main CPU is asleep and applications can retrieve historical motion data via the CoreMotion API[13]. Some recent Android devices allow batching of sensor readings [14], and the Motorola Moto X provides recognition for a small number of predefined activities that can be used as wake-up conditions[15]. While these wake-up conditions work well for some applications, they are inefficient for many other types of applications that are not interested in the set of predefined activities. In contrast, Sidewinder supports a wide variety of applications by providing developers an easy mechanism to create custom wake-up conditions.

Most of the previously noted works focused on system architecture modification in order to lower the cost of sensing. Alternative approaches have also been explored. Ace [77] is a middleware that supports continuous context-aware applications while mitigating sensing cost for acquisition of context attributes (such as AtHome and IsDriving). It achieves power savings when multiple applications request strongly correlated context attributes. Additionally, it can reduce power consumption when a "cheaper" sensor exists, which can determine the value of a different context attribute that has a strong correlation with the requested context attribute (e.g. use the accelerometer to check if the user is jogging instead of using the GPS to determine if the user is at work). A middleware such as Ace is a great example of a library that can run on top of Sidewinder and achieve additional power savings. Sensor fusion has also been an active focus of related research. Data from multiple sensors can be used to increase context-awareness in mobile devices [5, 28].

While our focus was on power-efficient acquisition of sensor data, next generation mobile perception applications face related problems regarding partitioning of application code. MAUI [16] enables fine-grained energy-aware offload of mobile application code to remote servers. Similarly, Odessa [84] uses

[12] https://developer.qualcomm.com/mobile-development/maximize-hardware/3g4g-connectivity-gobi

[13] https://developer.apple.com/library/ios/documentation/coremotion/reference/coremotion_reference/index.html

[14] http://developer.android.com/about/versions/android-4.4.html

[15] http://www.motorola.com/us/X8-Mobile-Computing-System/x8-mobile-computing-system.html

code-offloading to address the issue of processing sensor data on resource constrained mobile devices.

6.7 Conclusion

In this chapter we proposed *Sidewinder*, a new approach for continuous mobile sensing. In this approach, the platform implements common sensor data processing algorithms that execute on a low-power processor, and application developers construct wake-up conditions for events of interest by selecting among the set of pre-defined common processing algorithms and tuning their parameters. We presented an extensive evaluation showing the benefits of using Sidewinder as wake-up mechanisms for the multiple accelerometer and audio-based applications.

Our immediate future work includes developing an FPGA-based prototype, performing a thorough exploration of what algorithms should be included as part of the platform and analyzing their power and computational requirements. We would also like to explore supporting multiple concurrent applications while still maintaining predictable performance. When receiving multiple wake-up conditions, the sensor manager can attempt to improve performance by combining the pipelines that use common algorithms.

Another interesting extension includes adding "smartness" to the low-power sensor hub. Application developers may face challenges in selecting the optimal algorithms and configuration parameters for their wake-up conditions. But given feedback from the more complex algorithms running on the application level, self-learning mechanisms may be able to tune the paramers used on the wake-up conditions. It is easy to imagine an application notifying the sensor hub about wake-ups when events of interest were not actually detected (i.e. false positives). However, it will be more difficult to automatically identify events of interest missed by the wake-up condition running on the low-power node (i.e. false negatives).

Chapter 7

Conclusion

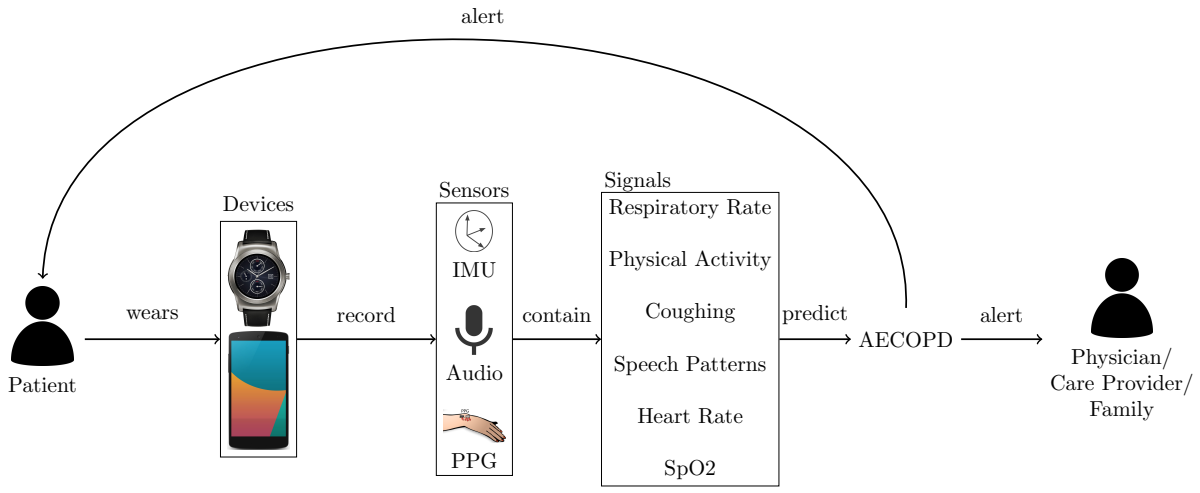


Figure 7.1: Overview of a potential complete system

COPD is a lung disease associated with "exacerbations" where the disease significantly worsens. If detected early, exacerbations can be treated early, reducing the likelihood of hospitalization. In this thesis we presented our initial steps towards a system for remote monitoring of patients with COPD. This work focused on developing methods for detecting physiological signals such as respiratory rate and coughing using existing smartwatches. Two central themes of my work were, 1) working with the limitations of existing mobile and wearable devices and 2) working with in-the-wild data. One key constraint of mobile devices is battery life. To address this, we proposed Sidewinder, a hardware architecture and programming paradigm to enable energy efficient and developer friendly continuous mobile sensing. However, without such a system available in current devices, throughout my work I employed a duty cycling scheme to record sensor data periodically throughout the day. While not entirely continuous, a duty cycling scheme provides a sample of data from throughout an individual's day. The second theme of my work is dealing with data collected in the wild. Data collected in the wild contains a large volume and variety of noise and it is often challenging to obtain associated ground truth data. These two factors combine to make working with in-the-wild data challenging as the variety of noise necessitates a large amount of labeled data if using supervised machine learning methods. In the WearBreathing work, I relied on a ground truth device that participants wore, which limited the amount of data we could collect. In contrast, in the cough detection work we did not rely on a ground truth device and were able to collect a vast amount of data. However, obtaining labels for the data became the bottleneck as it required manual, human annotation.

7.1 Future Work

As mentioned, the work presented in this thesis proposes a potential system for COPD monitoring and presents initial steps towards that system. Aside from the future work of each individual component presented as part of the system (Sidewinder, WearBreathing, CoughWatch), a comprehensive COPD monitoring solution, such as the one shown in Figure 7.1, would require further work. These works can

be categorized into physiological signals, usability and evaluation.

- Physiological Signals
  - Oxygen Saturation (SpO2): SpO2 is a measure of the proportion of oxygenated blood and has been shown to be useful for detecting exacerbations [79]. It can be measured via a Photoplethysmogram (PPG) sensor, which while available on smartwatches for measuring heart rate, is not currently used for measuring oxygen saturation. This is because SpO2 readings from a wrist are unreliable in the presence of motion artifacts [81]. A reliable SpO2 sensor on a smartwatch will be an incredibly useful for COPD monitoring.
  - Speech: Speech is a signal that we hypothesize could indicate difficulty breathing. If a smartwatch is able to reliably detect speech, subtle changes in speech patterns such as the amount of speech or duration and length of pauses between speech could be indicative of worsening respiratory function. Our preliminary work shows that current voice activity detectors (detecting whether a given segment of audio contains speech or not) are not reliable on in-the-wild smartwatch data. The first step would be building an accurate voice activity detector. Next we could extract parameters such as amount of speech, number of words spoken, length of pauses between words.
  - Heart rate and physical activity: While available on current smartwatches, the heart rate measurements taken by smartwatches have yet to be validated for patients with COPD. This is also true for other existing measure such as step count and physical activity.
  - Exacerbations: An exacerbation of COPD is a higher level signal than the other physiological signals discussed. A core motivator of this thesis is that continuous, long term monitoring of more basic physiological signals will allow establishing baselines of an individual patients health and enable detecting deviations from that baseline that could be indicative of an exacerbation.
- Usability
  - Adherence: In order to collect useful data, participants have to actually use the data collection device. This could mean completing questionnaires on a mobile phone or just putting on a smartwatch every morning. We call this adherence. Poor adherence and high drop-out rates is a common problem with mobile health studies [94]. Our interactions with participants suggest that allowing them to interact with the system by showing visualizations of their data may improve adherence rates. This is an avenue of current and future work for this project.
  - Amount of Data: Passive sensing generates a vast amount of data. Understanding who should see this data (patients, physicians, care providers, family members) and how requires a strong understanding of what each stakeholder wants to see and creating specific workflows and interactions for them.
  - Alerts: Once an exacerbation has been detected, the system should alert the patient and/or their care provider, physician or family members. Designing this alert system requires significant consideration as excessive false positives can create unnecessary worry or may lead to users ignoring the alerts while excessive false negatives will fail to improve care for patients.

- Evaluation
  - Clinical Trials: The work laid out in this thesis is aimed at developing a remote monitoring system for people with COPD. Accurate long-term monitoring of physiological signals can provide ways of detecting acute exacerbations of COPD early or even predicting them ahead of time. One way of evaluating an exacerbation detection system is to look at the precision and recall (or sensitivity and specificity) of detection. Developing such a system requires collection data that captures many exacerbations from many participants and knowing where in the data an exacerbation occurred. Then by extracting relevant signals from the data and applying either anomaly detection or some supervised time series prediction methods, we can evaluate how well we can detect exacerbations.

    However, there is another, arguably more meaningful way of evaluating the system. That is, evaluating the real world effects of deploying such a system. Here, instead of looking at precision and recall of a detector, we could look at hospital admission rates, long term mortality rates or cost to the healthcare system. Evaluating metrics such as these require clinical trials with experimental and control groups where the experimental group is equipped with our detection system and some intervention based on the detection of an exacerbation. The control group receives the current standard of care. For such a clinical trial, we not only need the exacerbation detection component, but an adequate intervention. This could be asking participants to follow up with a physician to determine the next course of action or asking participants to activate components of an action plan (e.g. take medication) that was pre-defined for them by their physician. A clinical trial such as this would evaluate how the entire system works as a whole and a successful study would show a significant difference in outcomes between the control and experimental group.

    This success, however, would be a function of not only technical aspects such as the accuracy of the sensors, signal extractors, exacerbation detection and intervention but also human factors such as whether patients consistently use the device and whether any data provided by the system cause them to change their behaviors. Based on our experience running even small scale studies, with so many factors at play, it very quickly becomes difficult to pinpoint the cause failure. Therefore, each aspect of the clinical trial should be built slowly over time and thoroughly tested to maximize the chances of success.

7.2 Monitoring of Other Health Conditions

The work presented in this thesis was motivated by COPD monitoring. However, the structure of the system presented in Figure 7.1 is likely applicable to a wide variety of health conditions by adopting the set of signals being extracted. For example, the same setup could potentially be made more relevant for monitoring people with bipolar disorder or seasonal affective disorder by adding measures of sleep and social interactions. Understanding which signals are important for which conditions requires domain knowledge and close collaboration with clinical researchers or other domain experts.

One limitation of what can be monitored with current devices is that they have to be chronic conditions. This is due to the battery limitations of wearable devices. In most of the work presented in this thesis, we relied on duty cycling while recording sensor data to provide a full days worth of battery

life. Duty cycling means we get a sample of data from throughout the patients days, but it comes at the cost of missing data. So for chronic illnesses where changes happen over longer periods, data every few minutes is sufficient to capture changes. However, it is unsuitable for acute events as they could occur during the "off" cycle. For instance, if an asthma attack or seizure happens during a period where our device is asleep, we will have missed that event. This is where low-power sensing pipelines such as those proposed in Chapter 6 come in. Being able to run algorithms on a low-power sensor hub continuously could enable detection of even acute events. For instance, the RF filter used in WearBreathing could potentially be run on a low power sensor hub continuously, and when it detects a suitable window of data, it could wake up the main processor to run the CNN extractor. With a low enough threshold, this would allow non-stop respiratory rate monitoring, which may be useful in detecting asthma attacks.

Finally, while symptoms such as respiratory rate and coughing are directly relevant to other conditions, the models we developed were built and evaluated with small, limited samples sizes and would require significant further training and testing to validate for other populations and conditions. Therefore, it is not the respiratory rate and cough detection models we developed that are the contributions of their respective works, but rather our methodology. In both works, we embraced the nature of in-the-wild data recordings and built our models to overcome the noisiness of such data as opposed to trying to limit the amount of noise.

Bibliography

[1] Justice Amoh and Kofi Odame. 2016. Deep Neural Networks for Identifying Cough Sounds. *IEEE Transactions on Biomedical Circuits and Systems* 10, 5 (Oct. 2016), 1003–1011. https://doi.org/10.1109/TBCAS.2016.2598794

[2] Yusuf A. Amrulloh, Udantha R. Abeyratne, Vinayak Swarnkar, Rina Triasih, and Amalia Setyati. 2015. Automatic Cough Segmentation from Non-Contact Sound Recordings in Pediatric Wards. *Biomedical Signal Processing and Control* 21 (Aug. 2015), 126–136. https://doi.org/10.1016/j.bspc.2015.05.001

[3] Matthias Baldauf, Schahram Dustdar, and Florian Rosenberg. 2007. A Survey on Context-Aware Systems. *International Journal of Ad Hoc and Ubiquitous Computing* 2, 4 (2007), 263–277.

[4] Álvaro Barbero and Suvrit Sra. 2014. Modular Proximal Optimization for Multidimensional Total-Variation Regularization. *arXiv:1411.0589 [math, stat]* (Nov. 2014). arXiv:math, stat/1411.0589

[5] Gregory Biegel and Vinny Cahill. 2004. A Framework for Developing Mobile, Context-Aware Applications. In *Pervasive Computing and Communications, 2004. PerCom 2004. Proceedings of the Second IEEE Annual Conference On*. IEEE, 361–365.

[6] S. S. Birring, T. Fleming, S. Matos, A. A. Raj, D. H. Evans, and I. D. Pavord. 2008. The Leicester Cough Monitor: Preliminary Validation of an Automated Cough Detection System in Chronic Cough. *European Respiratory Journal* 31, 5 (May 2008), 1013–1018. https://doi.org/10.1183/09031936.00057407

[7] J. M. Bland and D. G. Altman. 1986. Statistical Methods for Assessing Agreement between Two Methods of Clinical Measurement. *Lancet (London, England)* 1, 8476 (Feb. 1986), 307–310.

[8] P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi, and K. Tzoumas. 2015. Apache Flink: Stream and Batch Processing in a Single Engine. *IEEE Data Eng. Bull.* 38 (2015), 28–38.

[9] Peter H. Charlton, Mauricio Villarroel, and Francisco Salguiero. 2016. Waveform Analysis to Estimate Respiratory Rate. In *Secondary Analysis of Electronic Health Records*, MIT Critical Data (Ed.). Springer International Publishing, Cham, 377–390. https://doi.org/10.1007/978-3-319-43742-2_26

[10] Francine Chen, John Adcock, and Shruti Krishnagiri. 2008. Audio Privacy: Reducing Speech Intelligibility While Preserving Environmental Sounds. In *Proceedings of the 16th ACM International Conference on Multimedia (MM '08)*. ACM, New York, NY, USA, 733–736. https://doi.org/10.1145/1459359.1459472

[11] Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A Scalable Tree Boosting System. In *Proc. KDD '16*. ACM Press, San Francisco, California, USA, 785–794. https://doi.org/10.1145/2939672.2939785

[12] Zhenyu Chen, Mu Lin, Fanglin Chen, Nicholas D. Lane, Giuseppe Cardone, Rui Wang, Tianxing Li, Yiqiang Chen, Tanzeem Choudhury, and Andrew T. Campbell. 2013. Unobtrusive Sleep Monitoring Using Smartphones. In *Proceedings of the 7th International Conference on Pervasive Computing Technologies for Healthcare (PervasiveHealth '13)*. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), Venice, Italy, 145–152. https://doi.org/10.4108/icst.pervasivehealth.2013.252148

[13] Chollet, François and others. 2015. Keras. (2015).

[14] K. H. Chon, S. Dash, and K. Ju. 2009. Estimation of Respiratory Rate From Photoplethysmogram Data Using Time–Frequency Spectral Estimation. *IEEE Transactions on Biomedical Engineering* 56, 8 (Aug. 2009), 2054–2063. https://doi.org/10.1109/TBME.2009.2019766

[15] Michelle A. Cretikos, Rinaldo Bellomo, Ken Hillman, Jack Chen, Simon Finfer, and Arthas Flabouris. 2008. Respiratory Rate: The Neglected Vital Sign. *The Medical Journal of Australia* 188, 11 (June 2008), 657–659.

[16] Eduardo Cuervo, Aruna Balasubramanian, Dae-ki Cho, Alec Wolman, Stefan Saroiu, Ranveer Chandra, and Paramvir Bahl. 2010. MAUI: Making Smartphones Last Longer with Code Offload. In *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services*. ACM, 49–62.

[17] Parastoo Dehkordi, Ainara Garde, Behnam Molavi, J. Mark Ansermino, and Guy A. Dumont. 2018. Extracting Instantaneous Respiratory Rate From Multiple Photoplethysmogram Respiratory-Induced Variations. *Frontiers in Physiology* 9 (2018), 948. https://doi.org/10.3389/fphys.2018.00948

[18] Thomas Drugman, Jerome Urbain, Nathalie Bauwens, Ricardo Chessini, Carlos Valderrama, Patrick Lebecque, and Thierry Dutoit. 2013. Objective Study of Sensor Relevance for Automatic Cough Detection. *IEEE Journal of Biomedical and Health Informatics* 17, 3 (May 2013), 699–707. https://doi.org/10.1109/JBHI.2013.2239303

[19] Loubna Eltayara, Heberto Ghezzo, and Joseph Milic-Emili. 2001. Orthopnea and Tidal Expiratory Flow Limitation in Patients With Stable COPD. *Chest* 119, 1 (Jan. 2001), 99–104. https://doi.org/10.1378/chest.119.1.99

[20] J. R. Emery. 1987. Skin Pigmentation as an Influence on the Accuracy of Pulse Oximetry. *Journal of Perinatology: Official Journal of the California Perinatal Association* 7, 4 (1987), 329–330.

[21] Paul L. Enright. 2003. The Six-Minute Walk Test. *Respiratory Care* 48, 8 (Aug. 2003), 783–785.

[22] Florian Eyben, Felix Weninger, Florian Gross, and Björn Schuller. 2013. Recent Developments in openSMILE, the Munich Open-Source Multimedia Feature Extractor. ACM Press, 835–838. https://doi.org/10.1145/2502081.2502224

[23] J. F. Fieselmann, M. S. Hendryx, C. M. Helms, and D. S. Wakefield. 1993. Respiratory Rate Predicts Cardiopulmonary Arrest for Internal Medicine Inpatients. *Journal of General Internal Medicine* 8, 7 (July 1993), 354–360.

[24] Harvey Fletcher and W. A. Munson. 1933. Loudness, Its Definition, Measurement and Calculation. *The Journal of the Acoustical Society of America* 5, 2 (Oct. 1933), 82–108. https://doi.org/10.1121/1.1915637

[25] George Forman, Martin Scholz, G. Forman, and M. Scholz. 2010. Apples-to-Apples in Cross-Validation Studies: Pitfalls in Classifier Performance Measurement. *ACM SIGKDD Explorations Newsletter* 12, 1 (Nov. 2010), 49. https://doi.org/10.1145/1882471.1882479

[26] Kathleen C. Fraser, Jed A. Meltzer, and Frank Rudzicz. 2015. Linguistic Features Identify Alzheimer's Disease in Narrative Speech. *Journal of Alzheimer'ss Disease* 49, 2 (Oct. 2015), 407–422. https://doi.org/10.3233/JAD-150520

[27] Kathleen C. Fraser, Frank Rudzicz, and Graeme Hirst. 2016. Detecting Late-Life Depression in Alzheimer's Disease through Analysis of Speech and Language. Association for Computational Linguistics, 1–11. https://doi.org/10.18653/v1/W16-0301

[28] Hans W Gellersen, Albercht Schmidt, and Michael Beigl. 2002. Multi-Sensor Context-Awareness in Mobile Devices and Smart Artifacts. *Mobile Networks and Applications* 7, 5 (2002), 341–351.

[29] Jort F. Gemmeke, Daniel P. W. Ellis, Dylan Freedman, Aren Jansen, Wade Lawrence, R. Channing Moore, Manoj Plakal, and Marvin Ritter. 2017. Audio Set: An Ontology and Human-Labeled Dataset for Audio Events. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 776–780. https://doi.org/10.1109/ICASSP.2017.7952261

[30] D. R. Goldhill, A. F. McNarry, G. Mandersloot, and A. McGinley. 2005. A Physiologically-Based Early Warning Score for Ward Patients: The Association between Score and Outcome. *Anaesthesia* 60, 6 (June 2005), 547–553. https://doi.org/10.1111/j.1365-2044.2005.04186.x

[31] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press.

[32] Statistics Canada Government of Canada. 2014. Estimating the Prevalence of COPD in Canada: Reported Diagnosis versus Measured Airflow Obstruction. (March 2014).

[33] A. Guber, G. Epstein Shochet, S. Kohn-Bouzaglou, and D. Shitrit. 2018. Oxitone 1000: A First FDA Cleared Wrist-Sensor Pulse Oximeter for Continuous Patient Monitoring. In *D51. PHYSIOLOGY AND PHYSICAL ACTIVITY IN PULMONARY REHABILITATION*. American Thoracic Society, A7061–A7061. https://doi.org/10.1164/ajrccm-conference.2018.197.1_MeetingAbstracts.A7061

[34] Jono Hailstone and Andrew E. Kilding. 2011. Reliability and Validity of the Zephyr™ BioHarness™ to Measure Respiratory Responses to Exercise. *Measurement in Physical Education and Exercise Science* 15, 4 (Oct. 2011), 293–300. https://doi.org/10.1080/1091367X.2011.615671

[35] Khawar Hameed. 2003. The Application of Mobile Computing and Technology to Health Care Services. *Telematics and Informatics* 20, 2 (2003), 99–106.

[36] Tian Hao, Chongguang Bi, Guoliang Xing, Roxane Chan, and Linlin Tu. 2017. MindfulWatch: A Smartwatch-Based System For Real-Time Respiration Monitoring During Meditation. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 1, 3 (Sept. 2017), 1–19. https://doi.org/10.1145/3130922

[37] Nima Hatami, Yann Gavet, and Johan Debayle. 2018. Classification of Time-Series Images Using Deep Convolutional Neural Networks. In *Tenth International Conference on Machine Vision (ICMV 2017)*, Vol. 10696. International Society for Optics and Photonics, 106960Y. https://doi.org/10.1117/12.2309486

[38] Javier Hernandez, Daniel McDuff, and Rosalind W. Picard. 2015. BioWatch: Estimation of Heart and Breathing Rates from Wrist Motions. In *Proceedings of the 9th International Conference on Pervasive Computing Technologies for Healthcare (PervasiveHealth '15)*. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), ICST, Brussels, Belgium, Belgium, 169–176.

[39] Jong-yi Hong, Eui-ho Suh, and Sung-Jin Kim. 2009. Context-Aware Systems: A Literature Review and Classification. *Expert Systems with Applications* 36, 4 (2009), 8509–8522.

[40] Bret Hull, Vladimir Bychkovsky, Yang Zhang, Kevin Chen, Michel Goraczko, Allen Miu, Eugene Shih, Hari Balakrishnan, and Samuel Madden. 2006. CarTel: A Distributed Mobile Sensor Computing System. In *Proceedings of the 4th International Conference on Embedded Networked Sensor Systems*. ACM, 125–138.

[41] Andrey Ignatov. 2018. Real-Time Human Activity Recognition from Accelerometer Data Using Convolutional Neural Networks. *Applied Soft Computing* 62 (Jan. 2018), 915–922. https://doi.org/10.1016/j.asoc.2017.09.027

[42] Jaclyn Smith and Ashley Woodcock. 2006. Cough and Its Importance in COPD. *International Journal of Chronic Obstructive Pulmonary Disease* 1, 3 (Sept. 2006), 305–314.

[43] Álvaro Barbero Jiménez and Suvrit Sra. 2011. Fast Newton-Type Methods for Total Variation Regularization. In *Proceedings of the 28th International Conference on Machine Learning, ICML 2011, Bellevue, Washington, USA, June 28 - July 2, 2011*, Lise Getoor and Tobias Scheffer (Eds.). Omnipress, Bellevue, Washington, USA, 313–320.

[44] A. Johansson. 2003. Neural Network for Photoplethysmographic Respiratory Rate Monitoring. *Medical and Biological Engineering and Computing* 41, 3 (May 2003), 242–248. https://doi.org/10.1007/BF02348427

[45] James A. Johnstone, Paul A. Ford, Gerwyn Hughes, Tim Watson, and Andrew T. Garrett. 2012. Bioharness™ Multivariable Monitoring Device: Part. I: Validity. *Journal of Sports Science & Medicine* 11, 3 (Sept. 2012), 400–408.

[46] James A. Johnstone, Paul A. Ford, Gerwyn Hughes, Tim Watson, and Andrew T. Garrett. 2012. Bioharness™ Multivariable Monitoring Device: Part. II: Reliability. *Journal of Sports Science & Medicine* 11, 3 (Sept. 2012), 409–417.

[47] P. Kadambi, A. Mohanty, H. Ren, J. Smith, K. McGuinnes, K. Holt, A. Furtwaengler, R. Slepetys, Z. Yang, J. Seo, J. Chae, Y. Cao, and V. Berisha. 2018. Towards a Wearable Cough Detector Based on Neural Networks. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2161–2165. https://doi.org/10.1109/ICASSP.2018.8461394

[48] Haik Kalantarian and Majid Sarrafzadeh. 2015. Audio-Based Detection and Evaluation of Eating Behavior Using the Smartwatch Platform. *Computers in Biology and Medicine* 65 (Oct. 2015), 1–9. https://doi.org/10.1016/j.compbiomed.2015.07.013

[49] Diederik P. Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. *arXiv:1412.6980 [cs]* (Dec. 2014). arXiv:cs/1412.6980

[50] Predrag Klasnja, Sunny Consolvo, Tanzeem Choudhury, Richard Beckwith, and Jeffrey Hightower. 2009. Exploring Privacy Concerns about Personal Sensing. In *Proceedings of the 7th International Conference on Pervasive Computing (Pervasive '09)*. Springer-Verlag, Berlin, Heidelberg, 176–183. https://doi.org/10.1007/978-3-642-01516-8_13

[51] Clemens Kruse, Brandon Pesek, Megan Anderson, Kacey Brennan, and Hilary Comfort. 2019. Telemonitoring to Manage Chronic Obstructive Pulmonary Disease: Systematic Literature Review. *JMIR Medical Informatics* 7, 1 (March 2019), e11496. https://doi.org/10.2196/11496

[52] Sumeet Kumar, Le T. Nguyen, Ming Zeng, Kate Liu, and Joy Zhang. 2015. Sound Shredding: Privacy Preserved Audio Sensing. In *Proceedings of the 16th International Workshop on Mobile Computing Systems and Applications (HotMobile '15)*. ACM, New York, NY, USA, 135–140. https://doi.org/10.1145/2699343.2699366

[53] Lucia Kvapilova, Vladimir Boza, Peter Dubec, Martin Majernik, Jan Bogar, Jamileh Jamison, Jennifer C Goldsack, Duncan J Kimmel, and Daniel R Karlin. 2019. Continuous Sound Collection Using Smartphones and Machine Learning to Measure Cough. *Digital Biomarkers* 3, 3 (2019), 166–175.

[54] Nicholas D. Lane, Petko Georgiev, and Lorena Qendro. 2015. DeepEar: Robust Smartphone Audio Sensing in Unconstrained Acoustic Environments Using Deep Learning. In *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing - UbiComp '15*. ACM Press, Osaka, Japan, 283–294. https://doi.org/10.1145/2750858.2804262

[55] Eric C. Larson, TienJui Lee, Sean Liu, Margaret Rosenfeld, Shwetak N. Patel, E. C. Larson, T. Lee, S. Liu, M. Rosenfeld, and S. N. Patel. 2011. Accurate and Privacy Preserving Cough Sensing Using a Low-Cost Microphone. In *Proc. UbiComp '11*. ACM, 375–384.

[56] K. H. Lee, K. P. Hui, W. C. Tan, and T. K. Lim. 1993. Factors Influencing Pulse Oximetry as Compared to Functional Arterial Saturation in Multi-Ethnic Singapore. *Singapore Medical Journal* 34, 5 (Oct. 1993), 385–387.

[57] Jill Fain Lehman and Rita Singh. 2016. Estimation of Children's Physical Characteristics from Their Voices. 1417–1421. https://doi.org/10.21437/Interspeech.2016-146

[58] Daniyal Liaqat, Mohamed Abdalla, Pegah Abed-Esfahani, Moshe Gabel, Tatiana Son, Rober Wu, Andrea S. Gershon, Frank Rudzicz, and Eyal de Lara. 2019. WearBreathing: Real World

Respiratory Rate Monitoring Using Smartwatches. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 3, 2, Article Article 56 (June 2019), 22 pages. https://doi.org/10.1145/3328927

[59] Daniyal Liaqat, Silviu Jingoi, Eyal de Lara, Ashvin Goel, Wilson To, Kevin Lee, Italo De Moraes Garcia, and Manuel Saldana. 2016. Sidewinder: An Energy Efficient and Developer Friendly Heterogeneous Architecture for Continuous Mobile Sensing. In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '16)*. ACM, New York, NY, USA, 205–215. https://doi.org/10.1145/2872362.2872398

[60] Daniyal Liaqat, Ebrahim Nemati, Mahbubar Rahman, and Jilong Kuang. 2017. A Method for Preserving Privacy during Audio Recordings by Filtering Speech. In *2017 IEEE Life Sciences Conference (LSC)*. 79–82. https://doi.org/10.1109/LSC.2017.8268148

[61] Daniyal Liaqat, Robert Wu, Salaar Liaqat, Eyal de Lara, Andrea S. Gershon, and Frank Rudzicz. 2020. The Ground Truth Trade-Off in Wearable Sensing Studies. *arXiv:2001.09738 [cs]* (Jan. 2020). arXiv:cs/2001.09738

[62] Ryan Libby. 2009. A Simple Method for Reliable Footstep Detection in Embedded Sensor Platforms. (2009).

[63] Felix Xiaozhu Lin, Zhen Wang, and Lin Zhong. 2015. K2: A Mobile Operating System for Heterogeneous Coherence Domains. *ACM Trans. Comput. Syst.* 33, 2 (June 2015), 4:1–4:27. https://doi.org/10.1145/2699676

[64] Xiaozhu Lin, Zhen Wang, Robert LiKamWa, and Lin Zhong. 2012. Reflex: Using Low-Power Processors in Smartphones without Knowing Them. *Proc. Conf. Architectural Support for Programming Languages and Operating Systems (ASPLOS)* (March 2012).

[65] Jiayang Liu, Lin Zhong, Jehan Wickramasuriya, and Venu Vasudevan. 2009. uWave: Accelerometer-Based Personalized Gesture Recognition and Its Applications. *Pervasive and Mobile Computing* 5, 6 (2009), 657–675.

[66] Xing Liu, Tianyu Chen, Feng Qian, Zhixiu Guo, Felix Xiaozhu Lin, Xiaofeng Wang, and Kai Chen. 2017. Characterizing Smartwatch Usage in the Wild. In *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services - MobiSys '17*. ACM Press, Niagara Falls, New York, USA, 385–398. https://doi.org/10.1145/3081333.3081351

[67] P. C. Loizou. 2005. Speech Enhancement Based on Perceptually Motivated Bayesian Estimators of the Magnitude Spectrum. *IEEE Transactions on Speech and Audio Processing* 13, 5 (Sept. 2005), 857–869. https://doi.org/10.1109/TSA.2005.851929

[68] K. Venu Madhav, M. Raghu Ram, E. Hari Krishna, Nagarjuna Reddy Komalla, and K. Ashoka Reddy. 2013. Robust Extraction of Respiratory Activity From PPG Signals Using Modified MSPCA. *IEEE Transactions on Instrumentation and Measurement* 62, 5 (May 2013), 1094–1106. https://doi.org/10.1109/TIM.2012.2232393

[69] Nicolas Maisonneuve, Matthias Stevens, Maria E Niessen, Peter Hanappe, and Luc Steels. 2009. Citizen Noise Pollution Monitoring. In *Proceedings of the 10th Annual International Conference*

on Digital Government Research: Social Networks: Making Connections between Citizens, Data and Government. Digital Government Society of North America, 96–103.

[70] Nicolas Maisonneuve, Matthias Stevens, Maria E Niessen, and Luc Steels. 2009. NoiseTube: Measuring and Mapping Noise Pollution with Mobile Phones. In *Information Technologies in Environmental Engineering*. Springer, 215–228.

[71] K. McGuinness, K. Holt, R. Dockry, and J. Smith. 2012. P159 Validation of the VitaloJAK™ 24 Hour Ambulatory Cough Monitor. *Thorax* 67, Suppl 2 (Dec. 2012), A131–A131. `https://doi.org/10.1136/thoraxjnl-2012-202678.220`

[72] S. E. Minson, B. A. Brooks, C. L. Glennie, J. R. Murray, J. O. Langbein, S. E. Owen, T. H. Heaton, R. A. Iannucci, and D. L. Hauser. 2015. Crowdsourced Earthquake Early Warning. *Science Advances* 1, 3 (April 2015), e1500036–e1500036. `https://doi.org/10.1126/sciadv.1500036`

[73] Nilesh Mishra, Kameswari Chebrolu, Bhaskaran Raman, and Abhinav Pathak. 2006. Wake-on-Wlan. In *Proceedings of the 15th International Conference on World Wide Web*. ACM, 761–769.

[74] Alyn H. Morice, Eva Millqvist, Kristina Bieksiene, Surinder S. Birring, Peter Dicpinigaitis, Christian Domingo Ribas, Michele Hilton Boon, Ahmad Kantar, Kefang Lai, Lorcan McGarvey, David Rigau, Imran Satia, Jacky Smith, Woo-Jung Song, Thomy Tonia, Jan W. K. van den Berg, Mirjam J. G. van Manen, and Angela Zacharasiewicz. 2019. ERS Guidelines on the Diagnosis and Treatment of Chronic Cough in Adults and Children. *European Respiratory Journal* (Jan. 2019). `https://doi.org/10.1183/13993003.01136-2019`

[75] Vivian Genaro Motti and Kelly Caine. 2015. Users' Privacy Concerns about Wearables. In *Financial Cryptography and Data Security*, Michael Brenner, Nicolas Christin, Benjamin Johnson, and Kurt Rohloff (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 231–244.

[76] Vinod Nair and Geoffrey E. Hinton. 2010. Rectified Linear Units Improve Restricted Boltzmann Machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning (ICML'10)*. Omnipress, Madison, WI, USA, 807–814.

[77] Suman Nath. 2012. Ace: Exploiting Correlation for Energy-Efficient and Continuous Context Sensing. In *Proc. of the 10th Conference on Mobile Systems, Applications, and Services (MobiSys)*. ACM, 29–42.

[78] Neal Patwari, Joey Wilson, Sai Ananthanarayanan, Sneha K Kasera, and Dwayne R Westenskow. 2014. Monitoring Breathing via Signal Strength in Wireless Networks. *IEEE Transactions on Mobile Computing* 13, 8 (2014), 1774–1786.

[79] Claudio Pedone, Domenica Chiurco, Simone Scarlata, and Raffaele Antonelli Incalzi. 2013. Efficacy of Multiparametric Telemonitoring on Respiratory Outcomes in Elderly People with COPD: A Randomized Controlled Trial. *BMC Health Services Research* 13 (March 2013), 82. `https://doi.org/10.1186/1472-6963-13-82`

[80] Cuong Pham. 2016. MobiCough: Real-Time Cough Detection and Monitoring Using Low-Cost Mobile Devices. In *Intelligent Information and Database Systems*, Ngoc Thanh Nguyen, Bogdan Trawiński, Hamido Fujita, and Tzung-Pei Hong (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 300–309.

[81] Caleb Phillips, Daniyal Liaqat, Moshe Gabel, and Eyal de Lara. 2019. Wrist02 – Reliable Peripheral Oxygen Saturation Readings from Wrist-Worn Pulse Oximeters. *arXiv:1906.07545 [cs, eess]* (June 2019). arXiv:cs, eess/1906.07545

[82] Davy Preuveneers and Yolande Berbers. 2008. Mobile Phones Assisting with Health Self-Care: A Diabetes Case Study. In *Proceedings of the 10th International Conference on Human Computer Interaction with Mobile Devices and Services*. ACM, 177–186.

[83] Bodhi Priyantha, Dimitrios Lymberopoulos, and Jie Liu. 2011. Littlerock: Enabling Energy-Efficient Continuous Sensing on Mobile Phones. *Pervasive Computing, IEEE* 10, 2 (2011), 12–15.

[84] Moo-Ryong Ra, Anmol Sheth, Lily Mummert, Padmanabhan Pillai, David Wetherall, and Ramesh Govindan. 2011. Odessa: Enabling Interactive Perception Applications on Mobile Devices. In *Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services*. ACM, 43–56.

[85] Meeralakshmi Radhakrishnan, Sharanya Eswaran, Sougata Sen, Vigneswaran Subbaraju, Archan Misra, and Rajesh Krishna Balan. 2016. Demo: Smartwatch Based Shopping Gesture Recognition. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services Companion (MobiSys '16 Companion)*. ACM, New York, NY, USA, 115–115. `https://doi.org/10.1145/2938559.2938572`

[86] Mickael Rouvier, Grégor Dupuy, Paul Gay, Elie Khoury, Teva Merlin, and Sylvain Meignier. 2013. An Open-Source State-of-the-Art Toolbox for Broadcast News Diarization. Idiap.

[87] Noah Rubio, Richard A Parker, Ellen M Drost, Hilary Pinnock, Christopher J Weir, Janet Hanley, Leandro C Mantoani, William MacNee, Brian McKinstry, and Roberto A Rabinovich. 2017. Home Monitoring of Breathing Rate in People with Chronic Obstructive Pulmonary Disease: Observational Study of Feasibility, Acceptability, and Change after Exacerbation. *International Journal of Chronic Obstructive Pulmonary Disease* Volume 12 (April 2017), 1221–1231. `https://doi.org/10.2147/COPD.S120706`

[88] Elliot Saba. 2018. *Techniques for Cough Sound Analysis*. Ph.D. Dissertation. University of Washington.

[89] Nazir Saleheen, Supriyo Chakraborty, Nasir Ali, Md Mahbubur Rahman, Syed Monowar Hossain, Rummana Bari, Eugene Buder, Mani Srivastava, and Santosh Kumar. 2016. mSieve: Differential Behavioral Privacy in Time Series of Mobile Sensor Data. In *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing (UbiComp '16)*. ACM, New York, NY, USA, 706–717. `https://doi.org/10.1145/2971648.2971753`

[90] Thomas Schlömer, Benjamin Poppinga, Niels Henze, and Susanne Boll. 2008. Gesture Recognition with a Wii Controller. In *Proceedings of the 2nd International Conference on Tangible and Embedded Interaction*. ACM, 11–14.

[91] John W. Severinghaus and Shin O. Koh. 1990. Effect of Anemia on Pulse Oximeter Accuracy at Low Saturation. *Journal of Clinical Monitoring* 6, 2 (April 1990), 85–88. `https://doi.org/10.1007/BF02828282`

[92] T. Shany, S. J. Redmond, M. R. Narayanan, and N. H. Lovell. 2012. Sensors-Based Wearable Systems for Monitoring of Human Movement and Falls. *IEEE Sensors Journal* 12, 3 (March 2012), 658–670. https://doi.org/10.1109/JSEN.2011.2146246

[93] Eugene Shih, Paramvir Bahl, and Michael J Sinclair. 2002. Wake on Wireless: An Event Driven Energy Saving Strategy for Battery Operated Devices. In *Proc. of the 8th Conference on Mobile Computing and Networking(MobiCom)*. ACM, 160–171.

[94] Sara Simblett, Ben Greer, Faith Matcham, Hannah Curtis, Ashley Polhemus, José Ferrão, Peter Gamble, and Til Wykes. 2018. Barriers to and Facilitators of Engagement With Remote Measurement Technology for Managing Health: Systematic Review and Content Analysis of Findings. *Journal of Medical Internet Research* 20, 7 (Dec. 2018), e10480. https://doi.org/10.2196/10480

[95] Joren Six, Olmo Cornelis, and Marc Leman. 2014. TarsosDSP, a Real-Time Audio Processing Framework in Java. In *Proceedings of the 53rd AES Conference (AES 53rd)*.

[96] Jacob Sorber, Nilanjan Banerjee, Mark D. Corner, and Sami Rollins. 2005. Turducken: Hierarchical Power Management for Mobile Devices.. In *Proc. of the 3rd Conference on Mobile Systems, Applications, and Services (MobiSyS)*. Seattle, WA.

[97] Brian Stasak, Julien Epps, Nicholas Cummins, and Roland Goecke. 2016. An Investigation of Emotional Speech in Depression Classification. 485–489. https://doi.org/10.21437/Interspeech.2016-867

[98] Xiao Sun, Zongqing Lu, Wenjie Hu, and Guohong Cao. 2015. SymDetector: Detecting Sound-Related Respiratory Symptoms Using Smartphones. In *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing (UbiComp '15)*. ACM, New York, NY, USA, 97–108. https://doi.org/10.1145/2750858.2805826

[99] Xiao Sun, Li Qiu, Yibo Wu, Yeming Tang, and Guohong Cao. 2017. SleepMonitor: Monitoring Respiratory Rate and Body Position During Sleep Using Smartwatch. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 1, 3 (Sept. 2017), 1–22. https://doi.org/10.1145/3130969

[100] Vinayak Swarnkar, Udantha R. Abeyratne, Anne B. Chang, Yusuf A. Amrulloh, Amalia Setyati, and Rina Triasih. 2013. Automatic Identification of Wet and Dry Cough in Pediatric Patients with Respiratory Diseases. *Annals of Biomedical Engineering* 41, 5 (May 2013), 1016–1028. https://doi.org/10.1007/s10439-013-0741-6

[101] John Sweller. 2011. CHAPTER TWO - Cognitive Load Theory. Psychology of Learning and Motivation, Vol. 55. Academic Press, 37–76. https://doi.org/10.1016/B978-0-12-387691-1.00002-8

[102] Theodoros Giannakopoulos. 2014. Silence Removal in Speech Signals - File Exchange - MATLAB Central - MathWorks File Exchange. http://www.mathworks.com/matlabcentral/fileexchange/28826-silence-removal-in-speech-signals.

[103] Christopher C Tsai, Gunny Lee, Fred Raab, Gregory J Norman, Timothy Sohn, William G Griswold, and Kevin Patrick. 2007. Usability and Feasibility of PmEB: A Mobile Phone Application for Monitoring Real Time Caloric Balance. *Mobile networks and applications* 12, 2-3 (2007), 173–184.

[104] Eldad Vizel, Mordechai Yigla, Yulia Goryachev, Eyal Dekel, Vered Felis, Hanna Levi, Isaac Kroin, Simon Godfrey, and Noam Gavriely. 2010. Validation of an Ambulatory Cough Detection and Counting Application Using Voluntary Cough under Different Conditions. *Cough* 6, 1 (May 2010), 3. https://doi.org/10.1186/1745-9974-6-3

[105] Rui Wang, Min S. H. Aung, Saeed Abdullah, Rachel Brian, Andrew T. Campbell, Tanzeem Choudhury, Marta Hauser, John Kane, Michael Merrill, Emily A. Scherer, Vincent W. S. Tseng, and Dror Ben-Zeev. 2016. CrossCheck: Toward Passive Sensing and Detection of Mental Health Changes in People with Schizophrenia. In *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing (UbiComp '16)*. ACM, 886–897. https://doi.org/10.1145/2971648.2971740

[106] Rui Wang, Fanglin Chen, Zhenyu Chen, Tianxing Li, Gabriella Harari, Stefanie Tignor, Xia Zhou, Dror Ben-Zeev, and Andrew T. Campbell. 2014. StudentLife: Assessing Mental Health, Academic Performance and Behavioral Trends of College Students Using Smartphones. In *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing (UbiComp '14)*. ACM, 3–14. https://doi.org/10.1145/2632048.2632054

[107] Rui Wang, Gabriella Harari, Peilin Hao, Xia Zhou, and Andrew T. Campbell. 2015. SmartGPA: How Smartphones Can Assess and Predict Academic Performance of College Students. In *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing*. ACM, 295–306.

[108] Tom M. A. Wilkinson, Gavin C. Donaldson, John R. Hurst, Terence A. R. Seemungal, and Jadwiga A. Wedzicha. 2004. Early Therapy Improves Outcomes of Exacerbations of Chronic Obstructive Pulmonary Disease. *American Journal of Respiratory and Critical Care Medicine* 169, 12 (June 2004), 1298–1303. https://doi.org/10.1164/rccm.200310-1443OC

[109] Robert Wu, Daniyal Liaqat, Eyal de Lara, Tatiana Son, Frank Rudzicz, Hisham Alshaer, Pegah Abed-Esfahani, and Andrea S. Gershon. 2018. Feasibility of Using a Smartwatch to Intensively Monitor Patients with Chronic Obstructive Pulmonary Disease: Prospective Cohort Study. *JMIR mHealth and uHealth* 6, 6 (2018), e10046. https://doi.org/10.2196/10046

[110] Rui Xia and Yang Liu. 2016. DBN-ivector Framework for Acoustic Emotion Recognition. In *Interspeech 2016*. 480–484. https://doi.org/10.21437/Interspeech.2016-488

[111] Blaise Agüera y Arcas, Beat Gfeller, Ruiqi Guo, Kevin Kilgour, Sanjiv Kumar, James Lyon, Julian Odell, Marvin Ritter, Dominik Roblek, Matthew Sharifi, and Mihajlo Velimirović. 2017. Now Playing: Continuous Low-Power Music Recognition. *arXiv:1711.10958 [cs, eess]* (Nov. 2017). arXiv:cs, eess/1711.10958

[112] Jian Bo Yang, Minh Nhut Nguyen, Phyo Phyo San, Xiao Li Li, and Shonali Krishnaswamy. 2015. Deep Convolutional Neural Networks on Multichannel Time Series for Human Activity Recognition. In *Proceedings of the 24th International Conference on Artificial Intelligence (IJCAI'15)*. AAAI Press, Buenos Aires, Argentina, 3995–4001.

[113] M. Zaharia, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M. J. Franklin, A. Ghodsi, J. Gonzalez, S. Shenker, and I. Stoica. 2016. Apache Spark: A Unified Engine for Big Data Processing. *Commun. ACM* 59, 11 (Oct. 2016), 56–65. https://doi.org/10.1145/2934664

[114] B. Zhao, H. Lu, S. Chen, J. Liu, and D. Wu. 2017. Convolutional Neural Networks for Time Series Classification. *Journal of Systems Engineering and Electronics* 28, 1 (Feb. 2017), 162–169. https://doi.org/10.21629/JSEE.2017.01.18