

Design, Optimization, and Applications of Wearable IoT Devices

by

Ganapati Manjunath Bhat

A Dissertation Presented in Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

Approved May 2020 by the
Graduate Supervisory Committee:

Umit Y. Ogras, Chair
Chaitali Chakrabarti
Angelia Nedić
Radu Marculescu

ARIZONA STATE UNIVERSITY

August 2020

ABSTRACT

Movement disorders are becoming one of the leading causes of functional disability due to aging populations and extended life expectancy. Diagnosis, treatment, and rehabilitation currently depend on the behavior observed in a clinical environment. After the patient leaves the clinic, there is no standard approach to continuously monitor the patient and report potential problems. Furthermore, self-recording is inconvenient and unreliable. To address these challenges, wearable health monitoring is emerging as an effective way to augment clinical care for movement disorders.

Wearable devices are being used in many health, fitness, and activity monitoring applications. However, their widespread adoption has been hindered by several adaptation and technical challenges. First, conventional rigid devices are uncomfortable to wear for long periods. Second, wearable devices must operate under very low-energy budgets due to their small battery capacities. Small batteries create a need for frequent recharging, which in turn leads users to stop using them. Third, the usefulness of wearable devices must be demonstrated through high impact applications such that users can get value out of them.

This dissertation presents solutions to solving the challenges faced by wearable devices. First, it presents an open-source hardware/software platform for wearable health monitoring. The proposed platform uses flexible hybrid electronics to enable devices that conform to the shape of the user's body. Second, it proposes an algorithm to enable recharge-free operation of wearable devices that harvest energy from the environment. The proposed solution maximizes the performance of the wearable device under minimum energy constraints. The results of the proposed algorithm are, on average, within 3% of the optimal solution computed offline. Third, a comprehensive framework for human activity recognition (HAR), one of the first steps towards a solution for movement disorders is presented. It starts with an online learning framework for HAR. Experiments on a low power IoT device (TI-CC2650 MCU)

with twenty-two users show 95% accuracy in identifying seven activities and their transitions with less than 12.5 mW power consumption. The online learning framework is accompanied by a transfer learning approach for HAR that determines the number of neural network layers to transfer among uses to enable efficient online learning. Next, a technique to co-optimize the accuracy and active time of wearable applications by utilizing multiple design points with different energy-accuracy trade-offs is presented. The proposed technique switches between the design points at runtime to maximize a generalized objective function under tight harvested energy budget constraints. Finally, we present the first ultra-low-energy hardware accelerator that makes it practical to perform HAR on energy harvested from wearable devices. The accelerator consumes 22.4 μJ per operation using a commercial 65 nm technology. In summary, the solutions presented in this dissertation can enable the wider adoption of wearable devices.

*Dedicated to my parents Manjunath Bhat, Anasuya Bhat
and brother Goutam Bhat*

ACKNOWLEDGMENTS

This dissertation is the result of efforts over the last six years of graduate study and over 20 years of education. The last six years have been one of the most productive times of my life and it would not have been possible without the support of faculty, post-docs, graduate students, friends, and family. First and foremost, I would like to thank my advisor, Prof. Umit Y. Ogras, for the guidance, patience, and advice he has offered me over the years of my graduate study. His insightful recommendations on writing, time management, communication, and networking helped me grow as a researcher and person. I am also grateful for his help and support in my academic job search. His thoughtful comments and advice on the interview process helped me greatly in securing an academic position. Prof. Ogras did everything to ensure his students' success, which is something I will strive to achieve in my career.

I would like to thank Prof. Chaitali Chakrabarti, Prof. Angelia Nedić, and Prof. Radu Marculescu for being part of my Ph.D. defense committee. Their insightful comments helped me in improving this dissertation. I am also thankful to Prof. Partha Pratim Pande, Prof. Janardhan Rao Doppa, Prof. Prabhat Mishra, Prof. Daniel Bliss, Prof. Hyung Gyu Lee, and Dr. Hyunseok Lee for their inputs and guidance in my research over the past few years. I am also thankful for working with Dr. Suat Gumussoy and for his continued mentorship.

I am grateful for the interesting discussions, words of encouragement, and lighter moments with friends and colleagues at ASU: Dr. Ujjwal Gupta, Dr. Jaehyun Park, Dr. Samet Arda, Md. Muztoba, Sumit K. Mandal, Manoj Babu, Darshan Sathyamurthy, Ranadeep Deb, Anish Krishnakumar, Yigit Tuncel, Sizhe An, Alper Goksoy, Toygun Basaklar, and Shruti Narayana. I also enjoyed the numerous eLab trips that I took with other members of the group.

I am also thankful to Semiconductor Research Corporation, National Science Foundation (NSF CAREER Award CNS-1651624), Defense Advanced Research Projects Agency (DARPA Young Faculty Award (YFA) Grant D14AP00068) for funding the research presented

in this dissertation. I would also like to thank Google for funding the initial years of my graduate study.

Finally, I would like to thank my parents, Manjunath Bhat and Anasuya Bhat, and brother, Goutam Bhat, for their constant love, support, and encouragement over the years. It is their love and encouragement that motivated me to pursue graduate studies and contribute to research. I am also deeply grateful to my partner, Catherine Spiers, for her unconditional support during my studies and many road trips we did together. I also thank my cousins and friends back in India, with whom I have numerous happy moments.

TABLE OF CONTENTS

	Page
LIST OF TABLES	xii
LIST OF FIGURES	xiv
CHAPTER	
1 INTRODUCTION	1
1.1 Contributions	2
1.2 Summary of Publications	5
2 <i>OPENHEALTH</i> : OPEN SOURCE PLATFORM FOR WEARABLE HEALTH	
MONITORING	8
2.1 Introduction	8
2.2 Wearable Health: Challenges and Solutions	10
2.3 OpenHealth Vision	14
2.4 OpenHealth Release	16
2.4.1 The Base Hardware	16
2.4.2 The Base Firmware	19
2.4.3 Public Release	20
2.5 Example Application Domains	21
2.5.1 Human Activity Recognition (HAR)	21
2.5.2 Gesture Recognition	22
2.6 Summary and Future Directions	22
3 NEAR-OPTIMAL ENERGY ALLOCATION FOR ENERGY HARVESTING	
IOT DEVICES	24
3.1 Introduction	24
3.2 Related Work	28

CHAPTER	Page
3.3 Preliminaries and Overview	29
3.4 Optimal Energy Management	32
3.4.1 Problem Formulation	32
3.4.2 Optimal Closed-Form Solution with Relaxed Constraints	34
3.4.3 Near-Optimal Runtime Solution	35
3.4.3.1 Uncertainty in Expected Energy Values	35
3.4.3.2 Perturbation of the Allocated Energy Values	36
3.4.3.3 User Activity and Minimum Energy Constraint	38
3.4.3.4 Summary of the Proposed Algorithm	39
3.5 Experimental Evaluation	39
3.5.1 Experimental Setup	39
3.5.2 Energy Allocation Over Time	41
3.5.3 Comparison to Offline Optimization	43
3.6 Summary	44
4 ONLINE HUMAN ACTIVITY RECOGNITION USING LOW-POWER WEARABLE DEVICES	48
4.1 Introduction	48
4.2 Related Work and Novelty	53
4.3 Human Activity Recognition Dataset	56
4.3.1 Wearable System Setup	56
4.3.2 User Studies	57
4.3.3 Dataset Description	58
4.3.4 Flow for Using the w-HAR Dataset	59
4.3.5 Comparisons with Existing HAR Datasets	61

CHAPTER	Page
4.4 Feature Set and Classifier Design	62
4.4.1 Goals and Problem Statement	62
4.4.2 Sensor Data Segmentation	63
4.4.3 Feature Generation	67
4.5 Classifier Design	70
4.5.1 Supervised Learning for State Classification	70
4.5.2 Proposed NN Classifier Design.....	71
4.6 Online Learning for Human Activity Recognition	73
4.6.1 Online Learning with Policy Gradient.....	74
4.6.2 Online Updates with Incremental Supervised Learning.....	77
4.7 Experimental Evaluation and Discussions	78
4.7.1 Experimental Setup	78
4.7.2 Neural Network Design Space Exploration	79
4.7.3 Accuracy Analysis of the Neural Network	81
4.7.3.1 Confusion Matrix	81
4.7.3.2 Comparison with Other Classifiers	82
4.7.3.3 Robustness of the NN Classifier.....	82
4.7.4 Online Learning with new users	83
4.7.5 Power, Performance and Energy Evaluation	86
4.8 Summary	88
5 TRANSFER LEARNING FOR HUMAN ACTIVITY RECOGNITION USING REPRESENTATIONAL ANALYSIS OF NEURAL NETWORKS	89
5.1 Introduction	89
5.2 Related Research	92

CHAPTER	Page
5.3 Human Activity Recognition Framework	94
5.3.1 Experimental Datasets	95
5.4 Transfer Learning for Human Activity Recognition	95
5.4.1 Flow of the Proposed Transfer Learning Approach	95
5.4.2 Clustering Users with Distinct Activity Patterns	96
5.4.3 Baseline Classifier Training for each User Cluster	98
5.4.4 Analysis of Distance Between Trained Networks	99
5.4.5 Transferring the NN and Fine-tuning	103
5.5 Evaluations	104
5.5.1 Accuracy Analysis	104
5.5.2 Training Time, Loss and Convergence Analysis	106
5.6 Summary	107
6 <i>REAP</i> : RUNTIME ENERGY-ACCURACY OPTIMIZATION FOR ENERGY HARVESTING IOT DEVICES	108
6.1 Introduction	108
6.2 Related Work	111
6.3 Runtime Energy-Accuracy Optimization	112
6.3.1 Preliminaries	112
6.3.2 Optimization Problem Formulation	113
6.3.3 Runtime Optimization Algorithm	114
6.4 Human Activity Recognition Case Study	115
6.4.1 Background and Baseline Implementation	115
6.4.2 Pareto-Optimal Design Points	117
6.5 Experimental Evaluation	120

CHAPTER	Page
6.5.1 Experimental Setup	120
6.5.2 Expected Accuracy and Active Time Analysis	121
6.5.3 Accuracy – Active Time Trade-off Analysis	123
6.5.4 Case Study using Real Solar Energy Data	124
6.6 Summary	126
7 AN ULTRA-LOW ENERGY HUMAN ACTIVITY RECOGNITION ACCEL- ERATOR	127
7.1 Introduction	127
7.2 Related Work	129
7.3 The Proposed Baseline HAR Engine	131
7.3.1 Input Data	132
7.3.2 Preprocessing the Raw Sensor Data	132
7.3.3 Feature Generation	134
7.3.4 Single-Level Baseline DNN Classifier	135
7.4 Activity-Aware 2-Level HAR Engine	137
7.4.1 Two-Class SVM Classifier	138
7.4.2 Decision Tree (DT) Classifier for Static Activities	139
7.4.3 DNN Classifier for Dynamic Activities	140
7.5 Low-Power Optimizations	140
7.5.1 Clock and Data Gating	140
7.5.2 Power Gating	143
7.5.3 Use of Low-Power Classifiers	144
7.6 Experimental Evaluation	145
7.6.1 Experimental Setup	145

CHAPTER	Page
7.6.2 Design Area	146
7.6.2.1 Baseline HAR Engine	146
7.6.2.2 Activity-Aware 2-Level HAR Engine	147
7.6.3 Accuracy Evaluation	148
7.6.3.1 Baseline HAR Engine	148
7.6.3.2 Activity-Aware 2-Level HAR Engine	151
7.6.4 Power Consumption Evaluation	153
7.6.4.1 Baseline HAR Engine	153
7.6.4.2 Activity-Aware 2-Level HAR Engine	154
7.6.5 Performance Evaluations	158
7.7 Summary	160
8 CONCLUSION AND FUTURE DIRECTIONS	161
8.1 Future Directions	163
REFERENCES	165

LIST OF TABLES

Table	Page
3.1. Summary of the Major Parameters	33
3.2. Parameter Values Used during Evaluations	40
4.1. List of Activities Used in the HAR Framework	57
4.2. Experimental Protocol for the HAR Dataset	58
4.3. Summary of the Number of Segments in Each Activity	59
4.4. Comparison with Existing HAR Datasets	62
4.5. Confusion Matrix for 18 Training Users	81
4.6. Comparison of Accuracy for Different Classifiers	82
4.7. Comparison of Accuracy Improvement with Online Learning	86
4.8. Execution Time, Power and Energy Consumption	87
6.1. Summary of Symbols Used in the Optimization Problem	113
6.2. Accuracy, Execution Time, Power, and Energy Consumption of Different Human Activity Recognition Design Points	119
7.1. Comparison of the Proposed HAR Engine to Relevant Custom Designs Reported in the Literature	130
7.2. Parameters Used for Deep Neural Network Training	149
7.3. Confusion Matrix for the Baseline Classifier	150
7.4. Confusion Matrix for the Level 1 SVM Classifier. The Static and Dynamic Activ- ities Are Classified with over 99% Accuracy	151
7.5. Confusion Matrix for the Activity-Aware HAR Engine for Static Activities	152
7.6. Confusion Matrix for the Activity-Aware HAR Engine for Dynamic Activities	152
7.7. Power Consumption Summary for the Baseline HAR Engine at $F = 100\text{kHz}$, $V =$ 1.0 V	154

Table	Page
7.8. Power Consumption Summary for the Activity-Aware HAR Engine at $F = 100$ kHz and $V = 1.0$ V	155
7.9. Summary of Latency of the Blocks in the Baseline HAR Engine and the Activity-Aware HAR Engine	159

LIST OF FIGURES

Figure	Page
2.1. Challenges of Wearable Health Platforms and Proposed Strategies	11
2.2. (a) Current Practice of Healthcare for Movement Disorders, (b) Envisioned Solution with the Use of Open Source Wearable Platform	13
2.3. Overview of the Wearable Platform	17
2.4. (a) Wearable Device in Flexible Form Factor (b) Screenshot of the Smartphone App Used to Control the Device	18
3.1. The Proposed Hardware Architecture and Energy Harvesting Framework. The Energy Harvesting Unit Channels the Generated Current between the IoT Device and Battery. Our Proof of Concept Prototype Uses PV-Cell as the Ambient Energy Source, but the Proposed Framework Can Work with Multiple Energy Sources. . .	25
3.2. Illustration of the Battery Level Computation for $T = 24$ Hr Horizon.	30
3.3. Illustration of the Application Utility Function.	32
3.4. Prototype (a) Front View, (b) Back View	40
3.5. Energy Allocation in January without Learning the User Pattern.	42
3.6. Energy Allocation in July without Learning the User Pattern.	42
3.7. Energy Allocation in January after Learning the User Pattern.	42
3.8. Energy Allocation in July after Learning the User Pattern.	42
3.9. Comparison of the Proposed Solution to Offline Optimization for Three Users over 12 Months.	44
4.1. Wearable System Setup, Sensors, and the Low-Power IoT Device [160]. We Knitted the Textile-Based Stretch Sensor to a Knee Sleeve to Accurately Capture the Leg Movements.....	50

Figure	Page
4.2. Flow-Chart for Using the W-HAR Data Set to Test New Algorithms or Reproduce the Results Presented in This Chapter	60
4.3. Illustration of the Segmentation Algorithm	64
4.4. Illustration of the Sensor Data Segmentation	67
4.5. Visualization of Segmentation for All Activities in the HAR Framework	67
4.6. Design Space Exploration for Neural Network Configuration	72
4.7. The NN Used for Activity Classification and Online Learning.	73
4.8. Comparison of Accuracy with the Number of Neurons	80
4.9. Comparison of Accuracy with Different Combinations of Users for Training	83
4.10. Comparison of Reinforcement Learning and Incremental Learning	84
5.1. Comparison of Stretch Sensor [117] Data of Four Users for a Single Step during Walk. There Is a Significant Change in Both the Range of Values and Data Patterns. The Grey Dashed Lines Show Different Instances of the Same Activity, While the Red Line Shows a <i>Representative</i> Activity Window for Each User.	90
5.2. Overview of the Transfer Learning Approach for HAR	96
5.3. The Architecture of CNN. The Layers Annotated at the Bottom Are Used in CCA Distance.	98
5.4. The Accuracy of the CNNs Tested with Different UCs for the W-HAR Dataset. The Red Star Shows the Accuracy of the UC Used Training While the Triangles Show Cross-UC Accuracy.	99
5.5. The CCA Distance between CNNs Trained with (a) UC 1, (b) UC 2, (c) UC 3, and (d) UC 4 from the W-HAR Dataset When Tested on All the Four UCs.	100
5.6. The CCA Distance between CNNs Trained with (a) UC 1, (b) UC 2, (c) UC 3, and (d) UC 4 from the UCI HAR Dataset When Tested on All the Four UCs.	101

Figure	Page
5.7. The CCA Distance between CNNs Trained with (a) UC 1 and UC 2, (b) UC 1 and UC 3, (c) UC 1 and UC 3 from the W-HAR Dataset When Tested on All the Four UCs.....	102
5.8. The CCA Distance between CNNs Trained with (a) UC 1 and UC 2, (b) UC 1 and UC 3, (c) UC 1 and UC 3 from the UCI HAR Dataset When Tested on All the Four UCs.	102
5.9. The CCA Distance between CNNs Trained with (a) UC 1 and UC 2, (b) UC 1 and UC 3, (c) UC 1 and UC 3 from the UCI HAPT Dataset When Tested on All the Four UCs.	102
5.10. Comparison of Accuracy between Original and Fine-Tuned CNN for the W-HAR Dataset.	103
5.11. Comparison between Original and Fine-Tuned NN for 200 UCs.	104
5.12. Transfer Learning Improvement Analysis: (a) Training Time, (b) Loss.....	104
5.13. Comparison between Original and Fine-Tuned NN for 100 UCs from the UCI HAR Dataset.	105
5.14. Transfer Learning Improvement Analysis: (a) Training Time, (b) Loss for the UCI HAR Dataset.	105
5.15. Comparison between Original and Fine-Tuned NN for 100 UCs from the UCI HAPT Dataset.	105
5.16. Transfer Learning Improvement Analysis: (a) Training Time, (b) Loss for the UCI HAPT Dataset.	105
6.1. Overview of the Human Activity Recognition Application.	116
6.2. The Knobs Used to Obtain Design Points with Different Energy-Accuracy Trade-Offs.....	117

Figure	Page
6.3. The Energy-Accuracy Trade-Off of Various Design Points. The Dashed Line Connects the Selected 5 Design Points.	118
6.4. Energy Consumption Distribution of DP1 over One-Hour Activity Period T_P . Total Energy Consumption Is 9.9 J.	119
6.5. (a) Expected Accuracy of <i>REAP</i> and Design Points. (b) The Active Time of Each DP Normalized to <i>REAP</i>	122
6.6. The Objective Value $J(T)$ of Static Design Points (Equation 6.1) Normalized to $J(T)$ of <i>REAP</i> with $\alpha = 2$	123
6.7. Performance (i.e., $J(T)$) Achieved by <i>REAP</i> normalized to DP1, DP3, and DP5 during the Month of September 2015. Error Bars Represent the Range of Improvement.	125
7.1. Architecture of the Baseline HAR Engine.	133
7.2. Architecture of the Activity-Aware HAR Engine.	138
7.3. A Representative Timing Diagram for the Activation of the Blocks in the Proposed HAR Engine. The Active Times of Feature and Classifier Clocks Are Detailed in Table 7.9.	141
7.4. Clock Gating Control Logic.	142
7.5. Power and Clock Domains in the Proposed Design.	143
7.6. Floorplan of the Baseline HAR Engine.	147
7.7. Floorplan of the Activity-Aware 2-Level HAR Engine.	147
7.8. Area of the Major Blocks Used in the Baseline HAR Engine.	147
7.9. Area of the Major Blocks Used in the Hierarchical 2-Level HAR Engine.	147
7.10. Power Consumption as a Function of Voltage.	156

Figure	Page
7.11. Comparison of Power Consumption of the Baseline and Activity-Aware HAR Engines	157

Chapter 1

INTRODUCTION

About 15% of the world's population lives with a disability according to the annual world report on disability [173]. Moreover, 100 to 190 million individuals face significant difficulties in functioning. State-of-the-art methodologies for diagnosis, treatment, and rehabilitation of this population rely on evaluations by medical professionals in a clinical environment [53]. However, as soon as the patient leaves the clinic, it is not possible to monitor their symptoms due to the lack of standard approaches [53]. The quality of life of this population can be improved significantly with the help of *wearable internet-of-things (IoT) devices* that combine sensing, processing, and wireless communication [48, 75].

Wearable sensors and mobile health applications are emerging as attractive solutions to augment clinical treatment and enable telepathic diagnostics [44, 46, 53]. Wearable devices have been recently used for monitoring of patients in a free-living home environment [98]. This capability allows doctors to understand the progression of symptoms over time [39]. Wearable devices have also shown promising results in the diagnosis and management of many movement disorders. For instance, studies in [132, 172] use wearable sensors and machine learning algorithms to identify essential tremor in patients. Wearable technology has also been widely used in the diagnosis and treatment of Parkinson's disease patients [38, 178]. Despite these promising results, widespread adoption of wearable sensors and devices has been limited [53, 107].

Recent research has focused on identifying the reasons that hinder the widespread adoption of wearable devices despite their potential in improving healthcare [53, 120]. Based on these, challenges to the widespread adoption of wearable devices can be classified into three

major categories. First, conventional rigid devices are uncomfortable and awkward to wear for long periods. Therefore, users prefer not to wear them in public [120]. Second, small-form-factor IoT devices must operate under extreme energy constraints, since large batteries are prohibitive [53]. Finally, the value of wearable IoT devices must be demonstrated by high-impact applications to expedite their adoption [53]. This dissertation addresses these challenges by making contributions in wearable device design, energy-neutral operation, and a comprehensive human activity recognition framework, as outlined in the next section.

1.1 Contributions

Wearable IoT devices have recently attracted significant attention due to advances in sensing, low-power processing, communication, and radio technologies [75, 97]. In particular, flexible hybrid electronics (FHE) technology offers great potential for sensor-rich wearable applications [21, 84]. FHE technology combines the performance advantages of conventional CMOS technologies and the form-factor advantages of flexible electronics. Our first contribution uses FHE technology to design a flexible wearable device for health monitoring. The device consists of a TI-CC2650 MCU, Invensense MPU-9250 sensor, and other components mounted on a flexible substrate. Since we target recharge-free operation, we use a flexible PV cell and energy harvesting circuitry to harvest energy from ambient light. We envision that the wearable prototype and its extensions can be used to create an open-source hardware/software ecosystem for health monitoring [16].

The limited battery capacity of wearable devices has led to the study of energy harvesting. Solar energy harvesting using PV-cells is one of the most promising techniques adopted by many recent studies [3, 125, 137]. Ambient energy harvesting necessitates the development of algorithms to efficiently manage the harvested energy such that device lifetime can be maxi-

mized. To this end, Kansal et al. [83] propose the concept of energy-neutral operation where the energy used by the device in any given period is equal to the harvested energy. Algorithms for energy-neutral operation are proposed in [83, 167]. However, these solutions do not consider application requirements in their algorithms. Therefore, our second contribution proposes a recharge-free solution for wearable devices [23]. We use a dynamic programming approach to enable optimal allocation of the harvested energy. We first calculate the optimal energy allocation using a closed-form formula. Then, we use a novel runtime algorithm that revises the allocations as a function of variations in the harvested energy. This contribution aims to address the challenge of frequent recharging of wearable devices.

Our third contribution proposes an online learning algorithm for human activity recognition (HAR). HAR aims to identify user activity, such as standing, walking, and jogging, by processing data acquired from sensors. HAR is important for personalized health care as one of the first steps in the treatment of movement disorders is to understand the daily activities of patients [44]. Therefore, HAR using wearable devices has attracted significant research attention [29, 131]. Smartphones have also been used for HAR recently [153]. These approaches use offline training and online inference that does not scale well to a large number of users. Therefore, there is a need to enable online learning for HAR. At the same time, training from scratch for all users can be slow and computationally intensive. To avoid this overhead, we develop a two-step framework for HAR [15]. This framework allows us to perform efficient online updates to a HAR classifier for new users. As part of this work, we collected activity data from 22 users while performing seven activities. The dataset has been released to the public as part of this dissertation.

One of the challenges in online learning for HAR is determining the appropriate information to transfer from classifiers learned on one set of users to a different set of users. In the context of neural networks, we need to carefully determine the number of layers to transfer

between users. The transfer ensures that the general features are preserved among users while avoiding overfitting to a single user. To address this issue, we employ transfer learning to adapt convolutional neural networks (CNN) trained offline to new users. Using representational analysis of CNNs [113], we show that the first few layers of the CNN provide features that are general for all the users. Using this insight, we transfer the CNN weights to new users and perform fine-tuning for *only* the deeper layers of the network. Evaluations using three datasets show that our approach achieves up to 43% accuracy improvement when compared to accuracy without using transfer learning. We also reduce the training time by 66% while maintaining the same accuracy as training from scratch.

Next, we combine the HAR framework and the energy allocation algorithms to enable a runtime energy-accuracy optimization method for energy harvesting IoT devices. These devices need to maximize their accuracy and active time under a tight energy budget imposed by the battery and form-factor constraints. To this end, we consider energy harvesting devices that run on a limited energy budget to recognize user activities over a given period. We propose a technique to co-optimize the accuracy and active time by utilizing multiple design points with different energy-accuracy trade-offs. The proposed technique switches between these design points at runtime to maximize a generalized objective function under tight harvested energy budget constraints. We experimentally validate the proposed approach using a custom prototype based on TI Sensortag [159] IoT board and real energy harvesting data.

Finally, we present *the first fully integrated custom hardware accelerator* for HAR that enables operation on harvested energy. It integrates *all steps of HAR*, i.e., reading the raw sensor data, generating features, and activity classification using a deep neural network (DNN). We synthesize the hardware engine using the commercial 65 nm low power technology node. Extensive evaluations show that the hardware engine consumes 22.4 μJ per operation, which is the lowest energy consumption reported in the literature.

In summary, this dissertation makes the following contributions:

- Wearable IoT devices using flexible hybrid electronics [16, 21, 26],
- Energy-neutral operation through optimal energy harvesting and management [23],
- Online learning framework and open-source dataset for human activity recognition [15],
- A transfer learning framework for human activity recognition,
- Runtime energy-accuracy co-optimization for energy harvesting IoT devices [14],
- An ultra-low-energy hardware accelerator for human activity recognition [25]

The rest of this dissertation describes each of these contributions in more detail. Specifically, Chapter 2 describes our open-source wearable platform for health monitoring. Chapter 3 details our algorithm for near-optimal energy allocation in wearable devices. Chapters 4 and 5 describe the online learning and transfer learning frameworks for the human activity recognition application, respectively. Chapter 6 presents our runtime energy-accuracy optimization framework for wearable devices. The ultra-low-energy hardware accelerator for HAR is presented in Chapter 7. Finally, Chapter 8 concludes the dissertation and provides directions for future work.

1.2 Summary of Publications

The work presented in this dissertation is based on research manuscripts written by the author [14–16, 21, 23, 25]. In addition to the design, optimization, and applications of wearable devices, the author has also worked on the reliability analysis of FHE devices [17, 59, 161]. Specifically, the author worked on developing models to estimate the stress experienced by FHE devices under various bending conditions. This modeling helps in reducing the number of test patterns that have to be performed mechanically, leading to significant savings in test time.

The author has also worked extensively on thermal and resource management in mobile platforms. Mobile devices, especially smartphones, are making a profound impact on multiple areas of human life, including communication, education, and health. In order to meet the increasing performance requirement of applications, mobile processors integrate multiple CPU cores, GPUs, and specialized processors in a small form factor, which leads to an increase in the power density. Higher power density, in turn, leads to increased junction and skin temperatures. A high junction temperature degrades reliability while a high skin temperature deteriorates the user experience.

Power and temperature form a positive feedback loop that can lead to a thermal runaway in an unstable system. Therefore, there is a strong need for a formal analysis of the power consumption-temperature dynamics in mobile devices. The author's work in [18] presents a theoretical analysis of the power-temperature dynamics in mobile systems. It reduces the thermal hotspots in a mobile platform to a single-input-single-output system to provide the region of convergence of the power-temperature dynamics in the system. It also provides an analytical method to determine the steady-state temperature of the system at runtime.

We extend the analysis to a multiple-input-multiple-output system in [20]. Then, we use the thermal stability models to design a control algorithm that manages the temperature of the system without affecting the performance of the application. Experiments on the Odroid-XU3 board [72] show that the control algorithm regulates the temperature with a minimal loss in performance when compared to the default thermal governors.

The stability analysis of mobile processors allows us to understand the steady-state dynamics of the system. At the same time, we must ensure that the temperature of the system does not violate the thermal constraints under dynamically changing workloads. To this end, the author's work in [19] provides a case study of the thermal behavior in two modern mobile processors. We also propose a predictive thermal management algorithm that uses predictions of

future temperature to make dynamic frequency management decisions such that the thermal limit is not violated [24]. Experiments on the Odroid-XU3 board [72] show that our algorithm successfully regulates the maximum temperature and decreases the temperature violations by one order of magnitude while also reducing the total power consumption on average by 7% compared with the default solution.

Along with thermal management, modern mobile platforms must manage the number of active cores and their frequencies optimally as per the changing requirements of applications [22, 109]. This problem is challenging because the number of possible configurations grows exponentially, and it is infeasible to search over the large number of configurations. Existing heuristics are also not efficient because they rely on resource utilization, which does not provide insight into workload characteristics. To address these issues, the author contributed to approaches that enable runtime resource management with Pareto-optimal configuration selection [66] and imitation learning [108, 109]. These approaches provide significant improvements in performance per watt when compared to the default interactive governor.

Chapter 2

OPENHEALTH: OPEN SOURCE PLATFORM FOR WEARABLE HEALTH MONITORING

2.1 Introduction

Movement disorders are becoming one of the leading causes of functional disability as a result of an aging population and extended life expectancy [52]. For example, Parkinson’s disease (PD), essential tremor (ET), epilepsy, and stroke affect more than 70 million people worldwide [44]. Diagnosis and treatment of movement disorders currently rely on tests and observations made by specialists in a medical facility, who prescribe medication and therapy based on these observations [107]. However, clinical visits, which are typically weeks apart, capture only a snapshot of the symptoms [54]. The low frequency of visits introduces complications in therapy decisions, since symptoms vary over time, and patients’ recall accuracy is not reliable [120]. Moreover, access to highly-trained specialists can be challenging in many parts of the world.

Wearable sensors and mobile health applications are emerging as attractive solutions to augment clinical treatment and enable telepathic diagnostics [44, 53, 54, 120]. Wearable technology allows for continuous monitoring of user movement in a free-living home environment. This capability helps in capturing the progression of symptoms that change over time. Furthermore, it enables evaluating the prescribed therapy on an individual basis [74, 107, 110]. Similarly, wearable sensors and smartphones have shown promising results in the diagnosis of ET [44] and detecting seizures in epilepsy [120]. Studies have also shown that both patients and health professionals (HPs) value the interactive information available from wearable

monitoring. Hence, wearable sensors coupled with telepathic diagnostics can greatly improve health care [74, 120].

Despite the promising results demonstrated so far, widespread adoption of wearable technology is hindered by both technology and adaptation challenges. The International Parkinson and Movement Disorders Society Task Force on Technology identifies the major challenges as non-compatible platforms, the clinical relevance of the “big data” acquired by sensors, and wide-spread/long-term deployment of new technologies [53]. According to the task force, open source projects can help in addressing these challenges by providing a common platform, with standardized hardware (HW) and software (SW) tools, driven by burning clinical needs. Several open source solutions for medical devices have been proposed recently. The work in [116] surveys open source devices for infusion pumps, brain-computer interfaces, CT scanners, and physiological monitoring. Among these, the e-Health¹ sensor platform is the most relevant for movement disorders, as it provides sensors for monitoring motion. However, the e-Health sensor platform comes in a large form factor, making it unsuitable for long-term wearable use.

The goal of this chapter is to discuss the major barriers to the widespread deployment of wearable health technology and present the *OpenHealth* framework as a potential solution. *OpenHealth* is an open source HW/SW platform for wearable health monitoring. *Our vision is to bridge the gap between isolated research activities, health professionals, and technology developers by facilitating research using a common platform, standards, and data sets.* Our open source release² includes all the HW and SW files of the OpenHealth platform, which includes energy harvesting circuitry, a modular sensor hub, processing hardware, a wireless modem, software drivers, and application-programming interfaces (API). Our initial application

¹<https://www.cooking-hacks.com/documentation/tutorials/ehealth-biometric-sensor-platform-arduino-raspberry-pi-medical>

² <https://sites.google.com/view/openhealth-wearable-health/home>

area is activity monitoring for movement disorder patients. Future versions of our open source platform will include applications, such as fall detection and seizure detection in epilepsy. We also provide reference implementations and data sets for human activity and gesture recognition applications. This feature aims to enable researchers who focus on applications to use *OpenHealth* without dealing with hardware details. All the hardware design files, software repository, and data sets are released under the GNU General Public License.

The rest of this chapter is organized as follows. Section 2.2 overviews the challenges faced by wearable technologies and our solution strategies. Our vision for the operation of *OpenHealth* and implementation details of our current release are discussed in Section 2.3 and Section 2.4, respectively. Finally, Section 2.6 summarizes our future directions.

2.2 Wearable Health: Challenges and Solutions

We classify the barriers to widespread adaptation of wearable health technologies as adaptation and technical challenges, as detailed in Figure 2.1. The adaptation challenges include social and user-specific barriers that prevent widespread deployment. In contrast, the technical challenges include barriers faced by the designers of the wearable platforms.

Adaptation Challenge 1 - Comfort: A substantial number of patients who participated in previous studies have reported feeling self-conscious when using wearable devices. They anticipate that the wearable device might be a probable cause of stigmatization [82, 120]. The users also expressed feeling embarrassment and awkwardness when wearing the sensor in public. It was stressful to even wear the sensor for some patients [82, 120]. To address this challenge, we propose devices based on flexible hybrid electronics, as illustrated in Figure 2.4(a) and detailed in Section 2.4.1. This approach enables physically flexible or stretchable devices that can blend in with clothes, such as a knee sleeve [15].

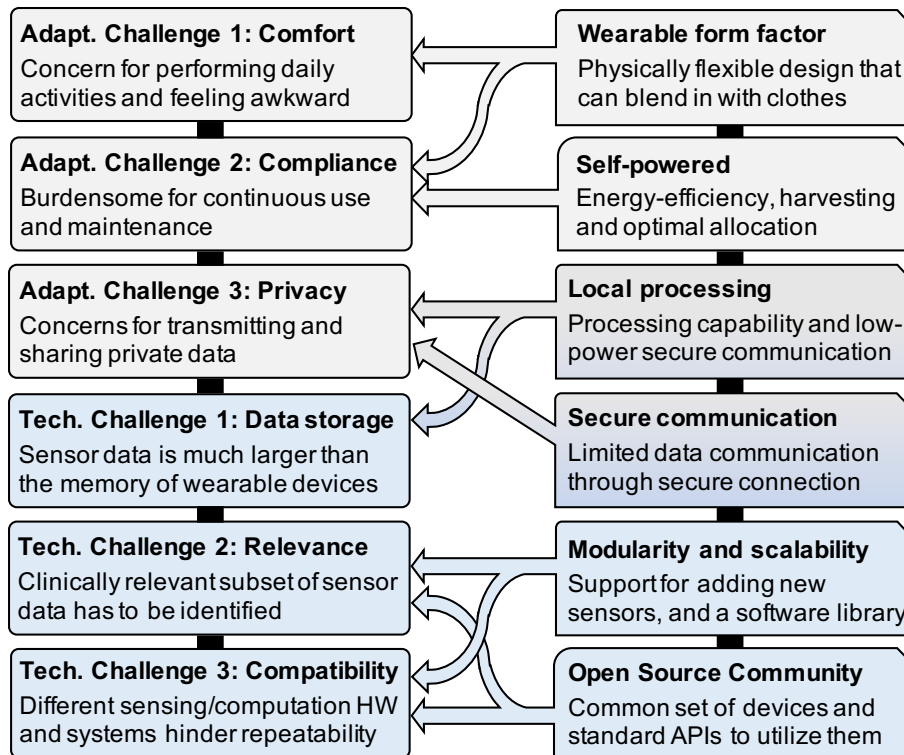


Figure 2.1: Challenges of wearable health platforms and proposed strategies

Adaptation Challenge 2 - Compliance: Participants in prior studies reported difficulty in using and charging the device regularly. It is difficult for a patient to charge the device since it may involve taking the device off and wearing it on again [120]. Larger and bulky batteries help in increasing the lifetime of the device, but they also make the device uncomfortable to wear. Studies have also shown that many patients find wearable devices uncomfortable or burdensome and stop using them after some time [44]. Thus, the device should be able to operate autonomously with minimum human intervention. Therefore, *OpenHealth* includes energy harvesting and dynamic energy allocation, which can eliminate battery charging requirements [23]. Furthermore, the *OpenHealth* platform operates autonomously to facilitate use without human intervention. More specifically, it automatically turns on, manages the power states, and communicates with a host device, such as a smartphone, when it senses motion, as described in

Section 2.4.1. The physically flexible form factor also promotes compliance, as illustrated in Figure 2.1.

Adaptation Challenge 3 - Privacy: Prior studies have shown that data privacy and security are among the primary concerns about using wearable devices for health monitoring [120]. Raw sensor data could be transferred to the cloud for the identification of technology-based objective measures (TOMs). Raw data transfer can cause security pitfalls as the sensor data contains sensitive information about the patient’s health. The first solution to this concern is processing the user-specific data locally to the maximum extent possible. For example, motion data from PD patients are processed locally to extract clinically relevant information. Then, *only the processed data* is transmitted through a secure channel *only to the health professional in charge*.

Technical Challenge 1 - Data Storage: Wearable sensors can collect a large amount of data. For instance, a 3-axis accelerometer alone can collect more than 5MB of data in one hour, while the local storage capacity of wearable devices is in the order of few megabytes. Hence, long-term storage of raw data is not sustainable. Since transmitting the raw data would quickly deplete the battery, it is not a viable option either. Therefore, the proposed solution provides local processing capability, as well as a library of signal processing and machine learning algorithms, as detailed in Sections 2.4.1 and 2.4.2. This strategy also benefits the privacy challenge.

Technical Challenge 2 - Relevance of Sensor Data: Large amounts of sensor data do not necessarily mean that all the data are clinically relevant [53]. In fact, a high volume of data can dilute its direct applicability [107]. Hence, sensors and algorithms should effectively extract relevant information for individualized patient treatment [44, 107]. We address this challenge through two mechanisms. First, the proposed platform features a modular sensor hub that allows adding new sensors for specific use-cases. Furthermore, the proposed platform provides hardware support and built-in functions, such as a variety of filtering and bio-marker gener-

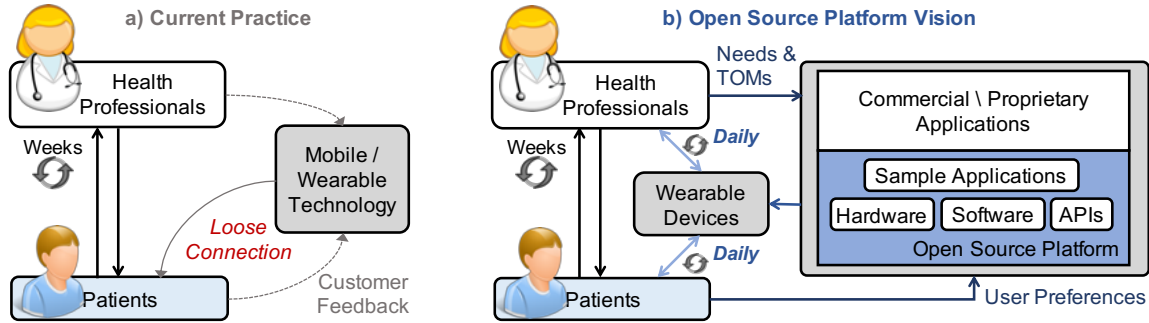


Figure 2.2: (a) Current practice of healthcare for movement disorders, (b) Envisioned solution with the use of open source wearable platform

ation algorithms. Second, the health professionals are principal members of the open source community. They communicate the clinical needs and TOMs with the developers, as shown in Figure 2.2 and detailed in Section 2.3.

Technical Challenge 3 - Compatibility: The International Parkinson and Movement Disorders Society Task Force on Technology emphasizes that the majority of technology development efforts operates within its own “islands of expertise”, with limited compatibility among the systems [53]. Since devices from distinct manufacturers may give non-compatible results, it is difficult to integrate data provided by different devices. In order to bridge this gap, we propose an open source design methodology where the wearable devices are derived from a common base platform, as illustrated in Figure 2.2(b) and detailed in Section 2.4. The compatibility of the proposed solution is improved by using the same underlying hardware, preprocessing software, and standard interfaces. This also facilitates comparing results from different research groups. Finally, open source solutions can constitute the foundation for commercial products, which add new proprietary intellectual property (IP) on top of the commonly used solutions.

Reliability and robustness: In addition to addressing the adaptation and technical challenges, we must ensure that wearable devices are reliable and robust. Reliable and robust design of the device ensures that the device remains operational when subjected to stress during normal

use by patients. Common causes of stress for wearable devices include bending, rolling, and folding by patients as a result of their activities [65]. We simulate different bending patterns in a finite element simulator (COMSOL) to test the reliability of the device before the manufacturing process.

The device must also be able to continuously sample the sensors, generate the features, and notify a caregiver in case emergencies. This is especially important for life-threatening emergencies such as seizures in patients with epilepsy. In our proposed solution, we aim at continuous energy-neutral operation by incorporating energy harvesting and a backup battery in the *OpenHealth* platform [23]. Furthermore, we perform on-device processing of the sensor data such that the latency of transferring the sensor data to a host device can be avoided.

2.3 OpenHealth Vision

Patients with movement disorders primarily interact with health professionals during office visits, which are typically weeks apart. Recently, smartphone apps and fitness trackers have been employed to monitor patients' symptoms during their daily life [54], as depicted in Figure 2.2(a). Although this is a useful starting point, these devices are not designed to take into consideration the patients' needs. Instead, the patients and health professionals provide feedback after using the device. Hence, they do not address the social and technical challenges, such as comfort, compliance, and clinical relevance, as discussed in Section 2.2. Consequently, the connection between the patients' needs and device capabilities is loose.

OpenHealth aims to provide a comprehensive framework that enables a tighter and systematic interaction between all the stakeholders in wearable health monitoring, as illustrated in Figure 2.2(b). Open source communities can bridge isolated research efforts, health professionals, and HW/SW developers to address the social and technical challenges [53]. In this

framework, health professionals provide needs and clinically relevant TOMs to the developers. TOMs are defined as technology-based objective measures provided by device-based clinical tests conducted in a standardized environment to have an objective assessment of specific behavior related to a movement disorder [53]. TOMs can also include the tests self-administered by patients to monitor symptoms in everyday life. TOMs help the health professionals in assessing patient symptoms such that the quality of care can be improved [53]. The second input consists of the preferences of the patients, who are the end users of the wearable devices, as shown in Figure 2.2(b). The preferences include materials used in the device, form factor, and battery life. These inputs from HPs and patients ensure that the wearable devices developed meet the requirements of the users from the onset, rather than relying on customer feedback.

The open source hardware and software is developed with the inputs from health professionals and users to address their requirements and needs. The *OpenHealth* platform also includes standard APIs, reference applications, and data sets. These APIs can be used by researchers to develop their own applications to detect and monitor TOMs without mastering the hardware design. An open source platform is important because it enables compatibility and standard comparisons of data. It can also enable the generation of common data sets for movement disorders, analogous to data sets such as the MNIST database³ for image recognition. This can give a boost to research in the area of movement disorders. In addition to the base open source platform, third parties can develop their own commercial applications as extensions to the base platform, as illustrated in Figure 2.2(b).

The final step in the OpenHealth architecture is the real-world usage of wearable devices by patients and clinicians for health monitoring — the wearable device supplements the existing office visits, which could be weeks apart. In our *OpenHealth* vision, the wearable device provides daily feedback of TOMs and other relevant parameters to both patients and their HPs.

³<http://yann.lecun.com/exdb/mnist/>

The daily feedback allows HPs to monitor the symptoms of their patients in real-time, allowing them to make better therapeutic decisions. Similarly, patients can benefit by having access to daily feedback about their symptoms from both HPs and the algorithms on the wearable device. As a result of this daily feedback, large improvements to the quality of life of patients can be made.

2.4 OpenHealth Release

The main components of the *OpenHealth* platform are shown in Figure 2.3. The hardware stack in the base platform consists of the most commonly used sensors, a microcontroller unit (MCU), radio, and energy harvesting circuitry. Similarly, the base software stack consists of the real-time operating system (RTOS), sensor APIs, communication services, and reference applications. In addition to the base platform, the wearable platform can be extended with additional sensors, algorithms, and applications.

2.4.1 The Base Hardware

Processing Unit: Texas Instruments (TI) CC2650 MCU⁴ is the main processing unit in our base hardware. It consists of an ARM Cortex-M3 core with an operating frequency of 47 MHz. The MCU includes 20 KB of SRAM and 128 KB of programmable flash. In addition to the main Cortex-M3 core, it also includes a low power sensor controller that can run autonomously from the rest of the system. The sensor controller can be used to monitor sensors while the rest of the system is in a low power sleep state. The device is always on to ensure autonomous operation, but it waits in low power mode until it detects active motion. When motion is de-

⁴<http://www.ti.com/product/CC2650>

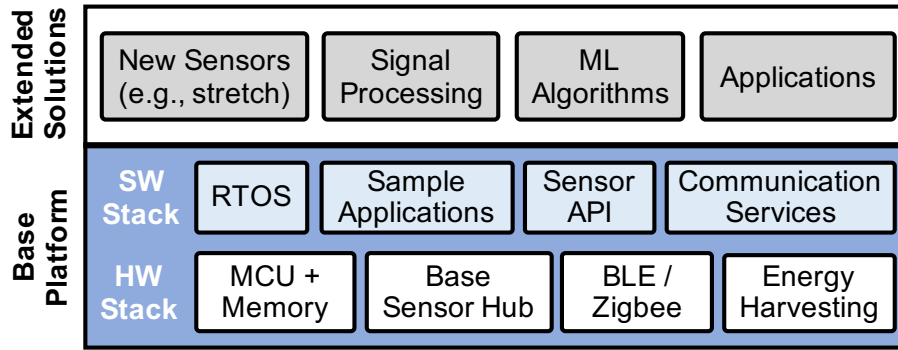
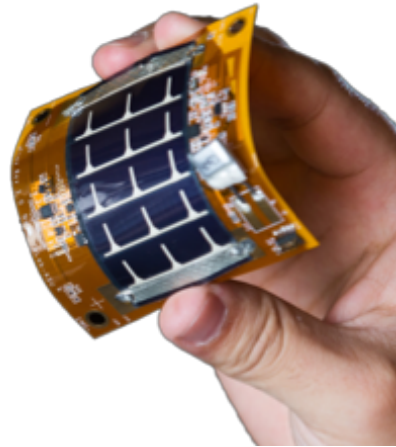


Figure 2.3: Overview of the wearable platform

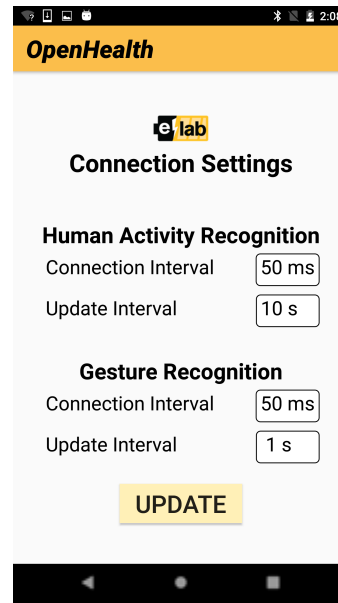
tected, the device wakes up and starts collecting sensor data. Then, the MCU executes the target application, such as human activity or gesture recognition and gesture recognition. The outputs are transmitted to a host device, such as a smartphone, without any user intervention after setting the connection settings with the host device, as illustrated in Figure 2.4(b). These settings control how often the wearable device synchronizes with the host. As mentioned before, on-board processing capability allows us to perform the processing of TOMs in real-time, thus eliminating the need to transfer the raw data.

Sensor Unit: The sensor unit in our base wearable platform consists of the Invensense MPU-9250⁵ motion sensor and electromyography (EMG) sensors. The MPU consists of a three-axis accelerometer and a three-axis gyroscope. They are used to track the motion of the user wearing the device. Similarly, the EMG sensor is used to record the electrical activity produced by the muscles of the user. The sensors are connected to the MCU using an SPI interface, which makes it easy to interface additional sensors in future extensions. We plan to add more sensors such as galvanic skin response and blood oxygen sensors in future versions of the device. We note that adding new sensors may require changes in layout and power supply architecture on the device.

⁵<https://www.invensense.com/products/motion-tracking/9-axis/mpu-9250/>



(a)



(b)

Figure 2.4: (a) Wearable device in flexible form factor (b) Screenshot of the smartphone app used to control the device

Communication: We use the integrated RF core in the TI CC2650 as the main communication module. It consists of a dedicated ARM Cortex-M0 to support communication tasks. The RF core can support the Bluetooth Low Energy (BLE) and ZigBee protocols using a 2.4 GHz RF transceiver. The BLE interface encrypts all the data that is sent over the air, thus ensuring the security and privacy of the user data. Moreover, BLE and Zigbee can be used to create a network of devices to make the platform scalable.

Energy Harvesting Unit: We use energy harvesting as the primary source of energy in our wearable platform to enable sustainable operation. In particular, we use solar energy harvesting with the help of a PV-cell, as shown in Figure 2.4. The PV-cell is connected to a maximum power point tracking charger that charges the Lithium-ion battery mounted on the device. The battery is used to store the harvested energy such that it can sustain autonomous operation when the harvested energy is below the energy requirement of the device [125].

Form factor: The base hardware can be manufactured in both rigid and flexible forms. Figure 2.4(a) shows the device in a flexible form factor. The flexible form factor is easy to wear as a patch on the body, which makes it comfortable to use for longer periods. We focus on incorporating a flexible form factor since flexible electronics technology has the potential to change the landscape in computing using stretchable and bendable platforms. Flexible hybrid electronics (FHE) [65, 118] can enable powerful computing abilities in a stretchable and bendable form factor by taking advantage of the computing abilities of rigid processors. As a result, FHE technology allows us to perform local processing of TOMs while maintaining a form factor that is comfortable and easy to wear as a patch. The rigid form factor is useful when a more rugged operation is required, such as mounting the device on a shoe. Both the form factors use PV-cells to harvest energy from the ambient light, fulfilling the energy harvesting requirement.

2.4.2 The Base Firmware

Operating System: The primary operating system consists of a thread-based, real-time operating system (RTOS) from Texas Instruments. The RTOS is responsible for scheduling and maintaining all the software tasks running in the system. The RTOS also provides drivers such as I2C, SPI, and UART to interface with the peripherals and sensors connected to the MCU. We employ the SPI interface to connect the motion sensors to the MCU.

Sensor API: The sensor API functions as the intermediate interface between low-level drivers and the application. In our software architecture, the sensor API provides functionality to control the sensors using the I2C or SPI interfaces. Specifically, it provides standard functions that allow the application to control the registers of the sensors and read data from each sensor. The modular nature of the sensor API interface allows the developers to new sensors easily.

Communication services: The communication services consist of the BLE and Zigbee proto-

col stacks. The protocol stacks run on the RF core in the MCU, independently of the application. Whenever the application has to send data to another device, it sends a message to the stack, which transmits the message over the air. All the data transferred using BLE and ZigBee is encrypted to ensure the security of the data.

Ease of development: We use the TI CC2650 MCU to ensure that tools needed for development of software for the wearable platform are easily accessible. Hence, *OpenHealth* users can use the software development kits, debugging kits, and RTOS libraries available for the TI MCU. In addition to the software tools provided by TI, we provide standard APIs to use the sensors in the *OpenHealth* platform. Finally, we plan to create a forum that will enable the users of the platform to discuss new applications, designs, and solutions to potential issues.

2.4.3 Public Release

The design files and firmware for the base platform are available for download at the *OpenHealth* web page⁶. In addition to HW design files, the public release also contains the base firmware that includes the RTOS, sensor API, communication libraries. To facilitate development effort and new research, we also include the reference applications and anonymous data sets collected for these applications. Users of the wearable platform will also be able to upload new data sets, thus helping in the creation of a common data repository for movement disorders. We plan to maintain an *OpenHealth* Wiki-page that will describe the steps required for manufacturing the base platform and reference applications, as well as answering frequently asked questions.

⁶<https://sites.google.com/view/openhealth-wearable-health/home>

2.5 Example Application Domains

The *OpenHealth* platform can be used to implement a variety of wearable applications ranging from fitness tracking to continuous health monitoring of patients with movement disorders. One of our focus application areas is the diagnosis and monitoring of patients with PD. For example, body motion analysis, response to therapy, and motor fluctuation monitoring of PD can be achieved with 3-axis accelerometers and gyroscopes available in the base platform. This section illustrates two reference applications included with the *OpenHealth* release. Adding more sensors like sweat sensors, heart rate sensors, and EEG can enable monitoring of nonmotor symptoms and progression [53].

2.5.1 Human Activity Recognition (HAR)

One of the first steps in the treatment of movement disorders is to understand what the patients are doing [44]. This objective is achieved through HAR algorithms, which aim to identify the user activity, such as standing, walking, and jogging, by processing sensor data. Since efficient HAR implementations can provide valuable insights to both health professionals and patients, we include it as one of the two reference applications. We provide a summary of the application here, while the full details of the application are present in Chapter 4. To implement our HAR application, we employ the base platform, which contains the MPU-9250 motion sensor, along with a wearable stretch sensor [15]. We use a combination of accelerometer and stretch sensors since it provides 10% higher recognition accuracy than using either sensor alone. The 3-axis accelerometer in the motion sensor is placed near the ankle to capture the swing of one of the legs. The stretch sensor is used on a knee sleeve to capture the degree of bending of the knee. During this work, we performed with 22 users and collected data for

seven activities and transitions between them. Then, the data is used to train a programmable neural network that can recognize these activities in real-time. Our experimental evaluation shows that we achieve accuracy greater than 90% for all the activities, as shown in Chapter 4. The application consumes about 12.5 mW power for recognizing a single activity. *The user data, as well as the implementation of the HAR application, are included in the OpenHealth release.*

2.5.2 Gesture Recognition

Gesture recognition is another application that is very useful in the context of health monitoring. Gesture recognition can be used in applications such as gesture-based control and interaction with assistive devices. We implemented a gesture recognition application that uses the accelerometer sensor in the base platform. The device is mounted on the wrist of the user to record the accelerometer data when the user is performing a gesture. We then use a NN classifier to recognize gestures, such as *up*, *down*, *left*, and *right*, which can be used to control assistive devices. Experimental evaluations using seven users show that the wearable device can recognize gestures with an accuracy of 98.6% while having an active power consumption of about 10 mW. Our implementation of the gesture recognition application [122–124] and the corresponding data set is available with the release of the wearable device.

2.6 Summary and Future Directions

This chapter presented the *OpenHealth* platform for open source health monitoring. It discussed the need for wearable health monitoring and the challenges faced by wearable devices before their widespread adoption. Then, it presented the hardware and software details of

the *OpenHealth* platform. Finally, we provided example applications for human activity and gesture recognition using the proposed wearable platform.

In order to assess whether the wearable device is able to address the challenges and meet our vision, we will conduct extensive user studies with movement disorder patients. We also plan to design custom SoC with reconfigurable NN accelerators to reduce the power footprint into the μW range. Furthermore, we will integrate additional modalities of energy harvesting, such as body heat and body motion. These will be key enablers to sustainable and maintenance-free operation.

Chapter 3

NEAR-OPTIMAL ENERGY ALLOCATION FOR ENERGY HARVESTING IOT DEVICES

3.1 Introduction

Advances in low power sensor, processor and wireless communication technologies enable a wide range of wearable applications. For instance, small form factor and low-cost IoT devices offer a great potential for non-invasive healthcare services that are not limited to any specific time or place [9, 42]. Exciting, and possibly pervasive, applications include health monitoring, activity tracking, and gesture-based control [126]. However, small form factor and low cost constraints severely limit the battery capacity. Therefore, harvesting ambient energy and optimal energy allocation are crucial for the success of wearable IoT devices.

Energy limitation is one of the major problems faced by wearable applications. Bulky batteries are heavy and inflexible, while small printed batteries have modest ($3.16\text{--}29.6\text{ mWh/cm}^2$) capacity [92, 95, 170], which requires frequent charging. Therefore, it is imperative to exploit ambient energy sources such as light, motion, and heat. Recent studies show that photovoltaic (PV) cells can provide 0.1 mW/cm^2 (indoor) – 100 mW/cm^2 (outdoor) power [164]. Similarly, human motion and heat can generate 0.73 mW/cm^3 [61] and 0.76 mW/cm^2 power at $\Delta T = 10\text{ K}$ [85], respectively. Energy harvesting can be particularly effective for wearable devices, since they are inherently personalized. For example, the device can easily learn the *expected* energy generation and consumption patterns based on daily activities. Therefore, we adopt energy harvesting as the primary source. At the same time, the intermittent nature and current source behavior of the energy sources necessitate an energy storage element, such as a battery

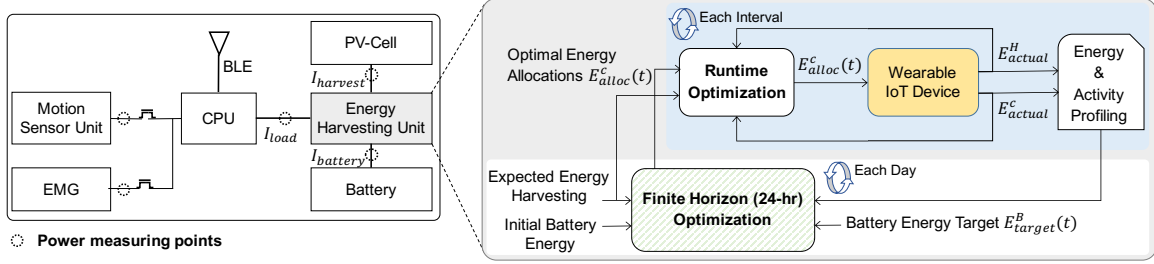


Figure 3.1: The proposed hardware architecture and energy harvesting framework. The energy harvesting unit channels the generated current between the IoT device and battery. Our proof of concept prototype uses PV-cell as the ambient energy source, but the proposed framework can work with multiple energy sources.

and super capacitance [137]. In this chapter, we utilize flexible rechargeable batteries as a reinforcement to provide a smooth quality of service and backup, in case the harvested energy falls significantly below expectations. The batteries we employ offer 148 mWh capacity at a $12 \times 35 \text{ mm}^2$ footprint, have 2 mm thickness and weigh 1.7 g [130].

The primary goal of this chapter is to provide *recharge-free* wearable IoT devices that maximize the quality of service (QoS). To achieve this goal, we propose a dynamic energy optimization framework with a finite time horizon. The proposed framework channels the generated power between the battery and the IoT device, while enforcing minimum and target energy constraints to guarantee recharge-free operation. The fundamental components of the proposed framework are illustrated in Figure 3.1 and described below.

Inputs and objective: The inputs to our optimization framework are the initial battery energy and the expected energy harvested pattern. In addition, we also specify the minimum battery level allowed at any point in time, and the battery energy target at the end of the day. The minimum energy constraint ensures that the battery always has a reserve to perform emergency tasks. Similarly, the energy level target ensures that the battery will have a desired level of charge at the end of each day. Our goal is to optimize the work performed by the IoT device, called the utility, under the battery level constraints. We measure the utility using an increasing

function of the energy allocated to the IoT device. This choice captures the fact that more energy allocation would lead to a larger utility. At the same time, it is more general than simply maximizing the allocated energy itself, since allocating more energy may have a diminishing rate of return.

Dynamic optimization with 24-hour horizon: The first component of the proposed solution is a finite horizon dynamic optimization formulation, as represented by the green patterned box in Figure 3.1. We set the finite time horizon as 24 hours since the energy harvesting pattern and user activities are repeated on a daily basis with potential day-to-day variations. The 24-hour horizon is divided into equal intervals, e.g., one hour or one-minute epochs. We derive a closed-form solution that gives the optimal energy allocations for each time interval during the day by using Karush-Kuhn-Tucker (KKT) conditions [11, 31, 91]. The optimality of this solution is guaranteed if the expected energy harvesting pattern matches with the actual generated energy. However, there are inter-day and inter-interval variations in the generated energy due to environmental conditions. Therefore, we also need to perturb the energy allocations computed using expected values.

Perturbation in each interval: The energy allocations computed at the beginning of each day deviate from their optimal values due to uncertainties in the harvested energy and load conditions. Therefore, we also perform runtime optimization by taking the differences in the expected and actual energy values into account. For example, suppose that the one-day horizon is divided into 24 one-hour intervals, and the energy harvested during the first hour is less than the assumed value. We compute this difference at the end of the first hour. Then, we reflect the difference in the energy allocations computed for the rest of the intervals on that day. In this way, the deviations from the optimal allocation are rectified at every interval. As a result, we continuously adapt to the changes in the environmental conditions with negligible runtime overhead.

Learning the daily patterns: Throughout the day, we keep track of harvested energy in each interval and use this data to find the *expected* energy harvesting pattern. Similarly, user motion patterns reveal low and high activity periods (e.g., sleep and exercise times). Daily averages of this data are fed to the proposed framework. Then, this data is used to guide the energy allocations, such as allocating minimum energy during sleep, as described in Section 3.4.3.

We demonstrate the proposed framework using the hardware prototype presented in Section 3.5.1. Our prototype employs flexible PV-cells to harvest energy from ambient light. The effectiveness of our optimization algorithm is evaluated for different user activities and energy harvesting patterns obtained from an online database [4, 163]. The proposed runtime algorithm is near-optimal, since the actual harvested energy in a given interval may be different from the expected value. Therefore, we compare our results with the maximum achievable utility computed using an oracle and an offline optimization algorithm [63, 64]. We show that the utility obtained by our runtime optimization approach is within 3% of the optimal utility, which is not feasible since it assumes an oracle. Moreover, our results converge to the optimal solution as the difference between the harvested energy and its expected value diminishes.

The major contributions of this chapter are as follows:

- We present a closed-loop solution for finding the optimal energy consumption of a self-powered IoT device when the amount of harvested energy is known a priori,
- Since the actual harvested and consumed energy may differ from their expected values, we propose a novel runtime algorithm with constant time complexity for setting the energy consumption in a finite horizon,
- We demonstrate that our results are, on average, within 3% of the optimal solution computed offline for a wide range of practical scenarios using a hardware prototype. We also show that the proposed algorithm incurs negligible power consumption and execution time penalty.

The rest of the chapter is organized as follows: We review the related work in Section 3.2. We present the preliminaries and the proposed algorithm in Section 3.3 and Section 3.4, respectively. Finally, we discuss the experimental results in Section 3.5 and summarize the conclusions in Section 3.6.

3.2 Related Work

Wearable IoT devices have recently attracted significant attention due to advances in sensing, low-power processing, communication protocol, and radio technologies [71, 97]. In particular, flexible hybrid electronics technology offers great potential for sensor-rich wearable applications [21, 65, 84].

The limited battery capacity of wearable devices has led to the study of energy harvesting. Major components of an energy harvesting system are the energy source, storage, harvesting circuit, and harvesting-aware power management [137, 155]. Solar energy harvesting using PV-cells is one of the most promising techniques adopted by many recent studies [3, 125, 137]. Body heat and motion can also generate energy with the help of thermoelectric [76, 157] and piezoelectric sensors [77, 142], respectively.

Energy harvesting aware power management for wireless sensor nodes has been studied extensively in recent years [60, 83, 167]. In particular, the work in [83] presents a general framework for including energy harvesting in power management decisions. The authors maximize the duty cycle of a sensor node using a linear program formulation. To avoid solving a linear program at runtime, the authors also present a low-complexity heuristic to solve the linear program. Similarly, a linear quadratic tracking based algorithm that adapts the duty cycle of the sensor node is presented in [167]. The authors minimize the deviation of the battery level

from a specified target. However, these solutions do not consider the application requirements when tuning the duty cycle of the nodes.

Concurrent task scheduling and dynamic voltage frequency scheduling is proposed to increase the lifespan of energy harvesting systems in [102]. At the beginning of each time interval, their algorithm refines the solar irradiance estimation and adjusts the task scheduling, but it is unable to correct future energy allocations. A design-time capacity planning and runtime adjustment method to achieve long-term recharge-free operation is presented in [33]. The method derives the battery capacity that can satisfy uninterrupted operation for a year. During runtime, the duty ratio of the device is changed based on the daily operation history. However, this approach only reacts to the harvested energy variations, thus leaving room for improvement.

In wearable IoT applications, energy can be optimized by considering user activity and application characteristics. Our proposed approach learns the energy harvesting and user activity patterns. We first calculate the optimal energy allocation using a closed-form formula, assuming the expected harvesting pattern. Then, we propose a novel runtime algorithm that both revises the optimal allocation dynamically and redistributes the slack from the previous intervals.

3.3 Preliminaries and Overview

We divide the one-day horizon into T equal intervals. For example, the battery energy illustration in Figure 3.2 assumes $T = 24$, i.e., each interval is one hour long. The proposed approach does not put any constraints on the level of granularity, provided that the overhead of the runtime energy allocation calculations is negligible⁷.

Energy constraints: The battery energy at the beginning of any interval t is denoted as E_t^B

⁷ Our implementation runs with one-minute intervals without any significant overhead.

for $0 \leq t \leq T - 1$. The proposed approach can work with multiple ambient sources such as a PV-cell, thermoelectric generator, and a piezoelectric device. In our experiments, we use a commercial PV-cell as the ambient energy source [57]. Suppose that the harvested and consumed energies in interval t are given by E_t^H and E_t^c , respectively. As illustrated in Figure 3.2, the battery energy dynamics can be expressed as:

$$E_{t+1}^B = E_t^B + \eta_t E_t^H - E_t^c, \quad 0 \leq t \leq T - 1 \quad (3.1)$$

where η_t is used to model the losses of the battery and power management circuitry, including the PV cell and voltage converters. The efficiency is time-varying since it is a function of generated current. Regardless of the harvested energy, the IoT device should have enough reserves to perform an emergency task, such as detecting a fall and sending an emergency signal. Therefore, we set a minimum battery level constraint E_{min} . Similarly, we constrain the energy level at the end of the day from below, such that there is sufficient reserve for the next day. Hence, the constraints on the battery energy level are given as:

$$E_T^B \geq E_{target} \quad \text{and} \quad E_t^B \geq E_{min} \quad \forall t \quad 0 \leq t \leq T - 1 \quad (3.2)$$

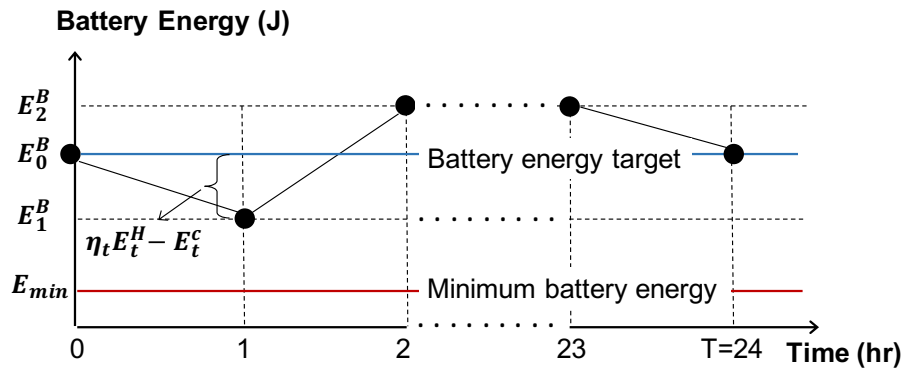


Figure 3.2: Illustration of the battery level computation for $T = 24$ hr horizon.

Driver applications and the utility function: Although the proposed framework does not depend on any particular application, we consider health monitoring and activity tracking as the

driver applications. We monitor the user activity using a motion sensor unit that integrates an accelerometer and a gyroscope. We also employ circuitry for real-time acquisition of physiological signals such as electromyography (EMG) and electrocardiogram (ECG). These signals are sampled and processed by a microcontroller unit (MCU). The processing results are transmitted to a personal device, such as a smartphone, using Bluetooth Low Energy (BLE) protocol.

The energy requirement of the target application is determined primarily by three factors. The first one is the active power consumption $P_{act}(f_t)$ as a function of the processing speed f_t during interval t . In our driver applications, this includes sampling the sensors, processing the data in real-time, and potentially transmitting data through BLE connection. The other factors are the duty ratio ρ_t , i.e., the percentage of time the application is active, and the idle power consumption P_{idle} . With these definitions, the average application power consumption in a given interval can be written as:

$$P_t = [\rho_t P_{act}(f_t) + (1 - \rho_t) P_{idle}] \quad (3.3)$$

A given target application needs a minimum duty ratio ρ_{min} and operating frequency f_{min} to accomplish its performance requirements. For example, it may need to guarantee a certain number of measurements per unit time. We use these requirements to compute the minimum energy M_E that should be allocated for each period. Allocating more energy can improve the QoS by delivering higher throughput, while less energy allocation means lower QoS. We define a utility function that expresses the quality of service in terms of M_E to capture this behavior. For illustration, a linear utility function $E_t^c - M_E$ is plotted in Figure 3.3. In general, allocating more energy has a diminishing rate of return, while allocation under M_E degrades quality at a faster rate. Hence, we employ a parameterized and generalized the utility function that captures this behavior as illustrated in Figure 3.3:

$$u(E_t^c) = \ln \left(\frac{E_t^c}{M_E} \right)^\alpha \quad (3.4)$$

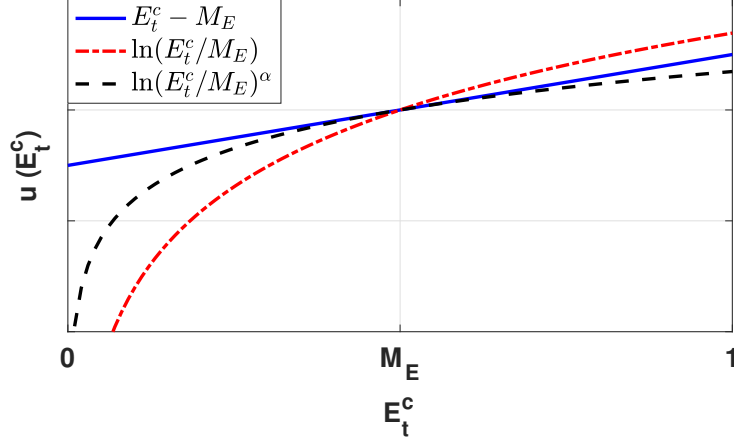


Figure 3.3: Illustration of the application utility function.

where the parameter α is used to tune the utility function for a specific user or application. We note that the algorithm presented next works with *any utility function that is concave and increasing*. The major parameters used in this chapter are summarized in Table 3.1.

3.4 Optimal Energy Management

3.4.1 Problem Formulation

Our goal is to maximize the utility over a one-day horizon under the energy constraints explained in Section 3.3. Hence, we can formulate the optimization problem using Equations 3.1–3.4 as follows:

$$\begin{aligned}
 & \text{maximize} && U(E_0^c, E_1^c \dots E_T^c) = \sum_{t=0}^{T-1} \beta^t \ln \left(\frac{E_t^c}{M_E} \right)^\alpha \\
 & \text{subject to} && E_{t+1}^B = E_t^B + \eta_t E_t^H - E_t^c && 0 \leq t \leq T - 1 \\
 & && E_{t+1}^B \geq E_{min} && 0 \leq t \leq T - 1 \\
 & && E_T^B \geq E_{target}
 \end{aligned} \tag{3.5}$$

Table 3.1: Summary of the major parameters

Symbol	Description
T	Number of control intervals in the finite horizon
$\beta > 0$	Discounting factor for utility
E_{min}, E_{target}	Minimum and target battery energy constraints
P_t	Power consumption of the IoT device in interval t
ρ_t, f_t	Duty ratio and frequency of the IoT device in interval t
M_E	Minimum energy required for positive utility
α	A positive parameter to control the shape of the utility function
E_t^H, E_t^c	Harvested and consumed energy in interval t
E_t^B	Battery energy at the beginning of interval t
Δ_t^h, Δ_t^c	Deviation from the expected values of harvested and consumed energy

In this formulation, we compute the total utility as the sum of the utilities in each interval. A positive discount factor $0 < \beta^t \leq 1$ is added to enable bias against distant intervals.

The optimal solution to the problem given in Equation 3.5 can be found offline using dynamic programming [12]. However, it requires solving a set of T nonlinear equations, which is computationally expensive for a runtime algorithm. Furthermore, it relies on the knowledge of the energy that will be harvested in the future intervals (i.e., E_t^H for $0 \leq t \leq T - 1$). In what follows, we propose a two-step solution based on two insights that enable us to overcome these challenges. The proposed solution leads to a near-optimal runtime algorithm with a complexity of $\mathcal{O}(1)$, i.e., the complexity does not grow with the time horizon or the number of intervals.

3.4.2 Optimal Closed-Form Solution with Relaxed Constraints

The proposed solution relies on two key insights:

Key insight-1: We can derive a closed-form analytical solution to this optimization problem if we *tentatively* ignore the minimum energy constraint. Obviously, the revised solution is not guaranteed to satisfy the minimum energy constraint $E_t^B \geq E_{min}$. However, we can enforce it at runtime at the expense of loss in optimality. Therefore, we find the closed-form solution at the beginning of each day. Then, the energy allocations are adjusted at the beginning of each interval, as described in Section 3.4.3.

Key insight-2: We cannot rely on the knowledge of the energy that will be harvested or consumed throughout the day. However, we can learn the expected patterns by profiling the generated energy during each time interval. This enables us to derive the optimal allocation for each interval at the beginning of each day by utilizing the expected values. Similarly, the actual energy consumption may be different from the optimal allocation, as detailed in Section 3.4.3. Therefore, we compare the actual generated and consumed energies with their expected values. Then, we use the difference to perturb the energy allocations for the remaining intervals, as described in Section 3.4.3. Since we relax the E_{min} constraint, there may be time intervals during which the battery level drops below the minimum threshold. Furthermore, the proposed approach can over- or under-allocate energy due to unexpected changes in the harvested energy, unlike an Oracle-based offline optimization. However, these effects do not propagate beyond one interval, since the proposed approach rectifies over- and under-allocations at the beginning of the next control interval. For the continuity of the discussion, we first summarize the closed-form solution with relaxed constraints below.

Closed-form solution: When we relax the minimum energy constraint and assume expected values for the harvested energy, the optimal energy allocation for each interval can be found as

follows:

$$\text{First interval : } E_0^c = \frac{E_0^B - E_{target} + \sum_{t=0}^{T-1} \eta_t E_t^H}{1 + \beta + \beta^2 + \dots + \beta^{T-1}} \quad (3.6)$$

$$\text{Subsequent intervals : } E_{t+1}^c = \beta E_t^c \quad 0 \leq t \leq T - 1$$

The derivation is presented in the Appendix. Note that the denominator can be computed a priori, and the total expected energy that will be harvested is available through profiling. Therefore, this closed-form equation enables us to compute the energy allocations with constant time complexity. Next, we explain how we employ this solution to design a runtime algorithm.

3.4.3 Near-Optimal Runtime Solution

This section presents our novel algorithm that builds on top of the closed-form solution given by Equation 3.6. The proposed algorithm perturbs the optimal allocations found using the expected energy values and enforces the minimum energy constraints at runtime.

3.4.3.1 Uncertainty in Expected Energy Values

The actual energy harvested at runtime may differ from the expected value due to factors such as environmental conditions. Efficiency in storing the harvested energy also adds to the uncertainty, since it varies with the load. We represent the difference between the actual energy generation and the expected value by Δ_t^H . $\Delta_t^H > 0$ ($\Delta_t^H < 0$) means that actual energy harvested during interval t is larger (smaller) than the expected value for that interval.

An IoT device uses the energy allocation target for a given interval t to compute the average power consumption allowed in that interval. Then, it finds the duty ratio and operating frequency using Equation 3.3, as summarized in Section 3.4.3.4. However, the actual energy consumed at the end of the interval may be different from the target. We subtract the actual

consumption from the allocated energy to find the difference Δ_t^c . Similar to the difference in the harvested energy, $\Delta_t^c > 0$ means a surplus, $\Delta_t^c < 0$ means that more energy than the allocated target is consumed. Hence, the difference between the expected energy accumulation and the actual values is:

$$\Delta_t = \Delta_t^H + \Delta_t^c \quad 0 \leq t \leq T - 1 \quad (3.7)$$

When Δ_t is positive, the energy surplus can be used during the remaining intervals. Otherwise, the consumed energy is more than the allocated target. Therefore, the deficit should be reflected in the remaining intervals.

3.4.3.2 Perturbation of the Allocated Energy Values

We need to adjust two quantities to account for the unpredictable dynamic variations. First, the optimal solution given in Equation 3.6 needs to be corrected in light of the new data available at the end of each interval. Second, the over or under expenditure in the previous interval should be distributed to future intervals.

Correcting the Future Allocations: Suppose that we adjust the optimal allocation at the beginning of the time interval t . The difference in expected and actual energy accumulated over earlier intervals $\{\Delta_0, \Delta_1, \dots, \Delta_{t-1}\}$ are known at this point. Therefore, the adjusted allocation for an interval t can be found using Equation 3.6 as:

$$E_t^c = \beta^t \frac{E_0^B - E_{target} + \sum_{k=0}^{T-1} \eta_k E_k^H + \sum_{k=0}^{t-1} \Delta_k}{1 + \beta + \beta^2 + \dots + \beta^{T-1}}$$

Since we are interested in a computationally efficient recursive solution, we can re-arrange the

terms to express E_t^c in terms of E_{t-1}^c and Δ_{t-1} only:

$$E_t^c = \beta \left(\beta^{t-1} \frac{E_0^B - E_{target} + \sum_{k=0}^{T-1} \eta_k E_k^H + \sum_{k=0}^{t-2} \Delta_k}{\sum_{k=0}^{T-1} \beta^k} + \frac{\beta^{t-1} \Delta_{t-1}}{\sum_{k=0}^{T-1} \beta^k} \right)$$

$$E_t^c = \beta \left(E_{t-1}^c + \frac{\beta^{t-1} \Delta_{t-1}}{\sum_{k=0}^{T-1} \beta^k} \right) \quad (3.8)$$

Hence, Equation 3.8 corrects the future allocations based on the most up-to-date energy generation and consumption information after each interval.

Redistributing the Surplus/Deficit: In addition to correcting the future allocations, we need to account for deviations from the revised optimal values in the past intervals. For example, assume that the optimal allocation for interval $t - 1$ was computed as 10 mAh, but the harvested energy in interval $t - 1$ turned out to be significantly lower than the expected value. Suppose that the optimal allocation in interval $t - 1$ is corrected as 6 mAh in light of the new measurements. Equation 3.8 corrects the future allocations, but it does not claim back 4 mAh overspent in the previous interval. In other words, Equation 3.8 alone does not make up for over-consumption or reclaim the underutilized energy allocations in the *previous intervals*. Therefore, we need to distribute Δ_{t-1} to the remaining intervals $[t, T - 1]$. A straightforward uniform distribution is not sufficient since any adjustment introduced at time t affects the future allocations due to the recursive rule in Equation 3.8.

Suppose that we add a correction term to Equation 3.8 as follows:

$$E_t^c = \beta \left(E_{t-1}^c + \frac{\beta^{t-1} \Delta_{t-1}}{\sum_{k=0}^{T-1} \beta^k} \right) + a_t \Delta_{t-1}$$

where a_t is a normalization coefficient that will ensure that the perturbations in the remaining intervals will add up to precisely Δ_{t-1} . By grouping the terms with Δ_{t-1} , we obtain:

$$E_t^c = \beta E_{t-1}^c + \left(\frac{\beta^t}{\sum_{k=0}^{T-1} \beta^k} + a_t \right) \Delta_{t-1} \quad (3.9)$$

Since the perturbation term will be multiplied with β in each future interval (due to the βE_{t-1}^c term), the sum of the perturbations from the current interval through the last one can be written

as:

$$\sum_{k=t}^{T-1} \beta^{k-t} \left(\frac{\beta^t}{\sum_{k=0}^{T-1} \beta^k} + a_t \right) \Delta_{t-1} = \Delta_{t-1}$$

By solving this equation, we can find a_t as:

$$a_t = \begin{cases} \frac{1-\beta}{1-\beta^{T-t}} - \frac{\beta^t}{\sum_{k=0}^{T-1} \beta^k} & 0 < \beta < 1 \\ \frac{1}{T-t} - \frac{1}{T} & \beta = 1 \end{cases} \quad (3.10)$$

3.4.3.3 User Activity and Minimum Energy Constraint

Profiling the energy consumption and user activity reveal specific periods with low or high activities. For example, it is possible to identify sleep and exercise periods. The proposed approach enables us to easily introduce new equality constraints based on this information. More precisely, we set $E_t^c = M_E$ for intervals t that fall during the sleep duration. Similarly, one can allocate a certain maximum value during expected exercise periods. We note that over-allocation does not have a significant drawback since unutilized allocations are distributed to future periods. However, under-allocation may hurt the utility if the interval duration is long (e.g., one hour). Therefore, low activity regions should be selected conservatively. Since these constraints can be introduced as pre-allocation, they do not change the formulation.

The final consideration is enforcing the minimum energy constraint. Equation 3.9 can cause the battery energy drain below E_{min} , since this constraint was relaxed to find a closed-form solution. Therefore, we project the remaining battery energy E_{t+1}^B at runtime using Equation 3.1 and compare it against E_{min} before committing to a solution. If there is a violation, we allocate the maximum energy that satisfies $E_{t+1}^B = E_{min}$. That is, the allocation becomes:

$$E_t^c = \begin{cases} \beta E_{t-1}^c + \left(\frac{\beta^t}{\sum_{k=0}^{T-1} \beta^k} + a_t \right) \Delta_{t-1} & E_{t+1}^B \geq E_{min} \\ E_t^B + \eta_t E_t^H - E_{min} & otherwise \end{cases} \quad (3.11)$$

where E_{t+1}^B and a_t and are given by Equations 3.1 and 3.10, respectively.

3.4.3.4 Summary of the Proposed Algorithm

We conclude this section with a step-by-step description of the runtime operation:

1. *At the beginning of each day:* Compute the allocation for the first interval E_0^c using Equation 3.6.
2. *For each interval $0 \leq t \leq T - 1$:* Divide the energy allocation E_t^c by the interval duration to find the target power consumption P_t . Then, use Equation 3.3 to find the duty ratio ρ_t . If there are multiple allowed frequency levels f_t , we use the most energy-efficient f_t . However, any feasible combination is acceptable.
3. *During each interval $0 \leq t \leq T - 1$:* Keep track of actual harvested and consumed energy. Compute Δ_t at the end of the interval by finding the difference between the expected and measured values.
4. *Before the start of each interval $1 \leq t \leq T - 1$:* Use Equation 3.11 to find the next allocation E_t^c . If $t = T - 1$ stop, otherwise increment t and go to step 2.

3.5 Experimental Evaluation

3.5.1 Experimental Setup

IoT Device Parameters: We employ the prototype shown in Figure 3.4 to demonstrate the proposed algorithm under realistic scenarios. It consists of an MPPT charger (TI BQ25504 [158]), a microprocessor (TI CC2650 [160]), a motion sensor unit (InvenSense MPU-9250 [79]), and EMG circuitry. We use a PV-cell from FlexSolarCells SP3-37 [57] as the energy-harvesting device and a 12 mAh Li-Po battery GMB 031009 [62] as the storage element. We have probes to measure the power consumption of different components, as illustrated in Figure 3.1. These

measurements are used to validate the power model given in Equation 3.3 as a function of the duty ratio and frequency. We also determined the IoT device parameters, such as E_{min} and M_E , listed in Table 3.2, based on these measurements.

Table 3.2: Parameter values used during evaluations

Parameter	Value	Parameter	Value
E_{min}	0.75 mAh	E_{target}	8 mAh
P_{idle}	2.2 mW	M_E	0.6 mAh
T	24	α	1

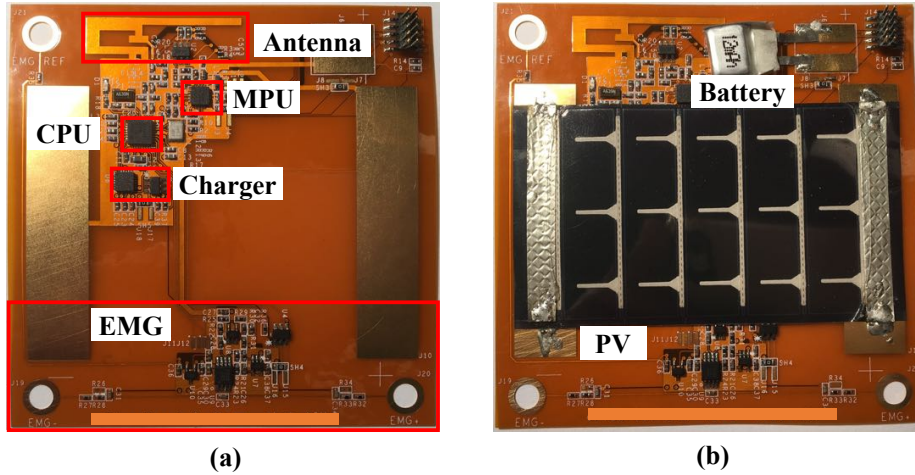


Figure 3.4: Prototype (a) front view, (b) back view

Energy Harvesting Model: The harvested energy is determined by the PV-cell and the radiation intensity, which is a function of observation time and location. I-V characteristics of SP3-37 are measured by varying the radiance from 100 to 1000 W/m^2 with the help of a halogen lamp. Then, this empirical data is used to model the maximum generated power as a function of radiation. This model enables us to compute the harvested energy when the radiation is known. To find the radiation, we first estimate the position of the sun at a given date and time using Sandia’s Ephemeris model [145]. Then, we convert the position information

to radiation using Ineichen’s model [78]. These three models are used by our algorithm to predict the energy that will be harvested during the day. We compare our results to an optimal offline algorithm implemented using the CVX package [64] and an oracle. The oracle uses the actual radiation, which is measured at every minute on the NREL Solar Radiation Research Laboratory’s baseline measurement system [4].

User Activity Model: The energy consumption varies as a function of the user activity. To evaluate a wide range of scenarios, we use different user activity patterns from the American Time Use Survey conducted by the US Department of Labor [163]. This survey contains the time a user spends on various activities. In our evaluations, we use five activity categories $\{sleep, work, exercise, leisure, others\}$. When the user is asleep, we allocate M_E to the corresponding interval. Otherwise, we use the proposed approach to find the optimal allocation.

3.5.2 Energy Allocation Over Time

We first illustrate the operation of the proposed algorithm for a specific user and date. Figure 3.5 shows the energy-harvesting profile, battery energy, and optimal allocations on January 1st for user-1. The energy harvesting profile (blue \circ markers) shows that there is little to none energy generation until 8 AM. During this period, the allocated energy (red \square markers) is supplied by the battery. The energy stored in the battery drops continuously (green \triangle markers) due to the lack of harvested energy. Once the harvested energy exceeds the energy allocated within an interval (around 10 AM), the battery energy starts recovering. We observe that our results match very closely with the result of the offline optimization that uses an oracle (dotted lines). We do not see a significant difference in the allocated energy throughout the day since the battery capacity is sufficient to absorb the variation in the harvested energy. However, we observe a dramatically different behavior for July, as shown in Figure 3.6. The peak harvested

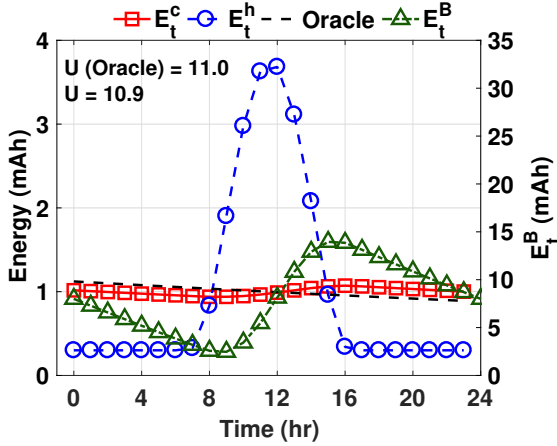


Figure 3.5: Energy allocation in January without learning the user pattern.

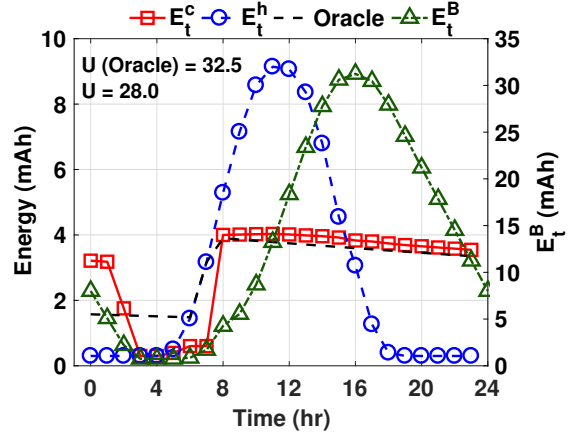


Figure 3.6: Energy allocation in July without learning the user pattern.

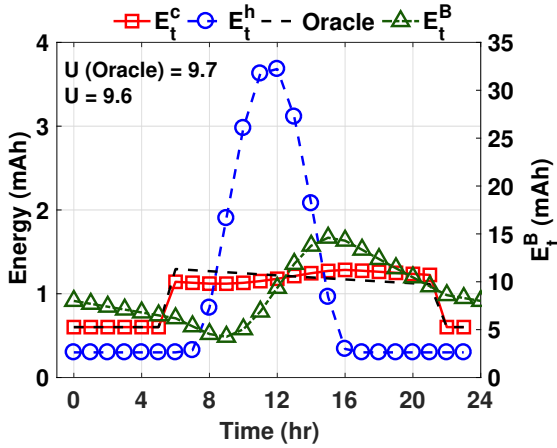


Figure 3.7: Energy allocation in January after learning the user pattern.

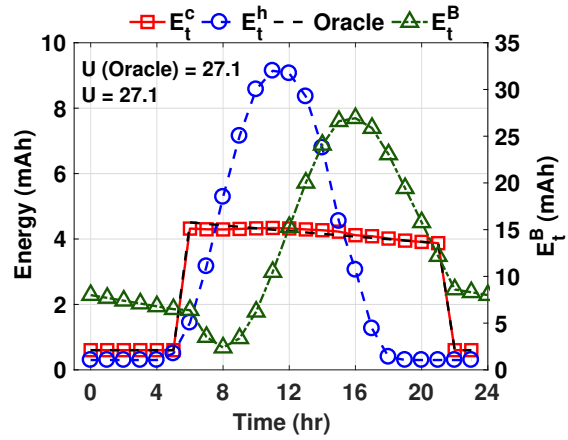


Figure 3.8: Energy allocation in July after learning the user pattern.

energy is about $2.5\times$ larger in July than January (~ 3.5 mAh versus ~ 9 mAh), and it spans a wider range. Therefore, the proposed algorithm allocates aggressively at the beginning of the day, relying on the energy that will be generated later. However, it hits the minimum battery energy constraint at 4 AM, unlike the offline optimization that accounts for E_{min} from the beginning. As soon as the battery energy drops to E_{min} , the proposed algorithm starts allocating *sub-optimally* only the harvested energy to the IoT device. The sub-optimal allocation continues until the harvested energy becomes sufficiently large to power the IoT device and charge

the battery (8 AM). While the allocation during the rest of the day closely follows the optimal allocation, the IoT device is under-powered from 3 AM to 8 AM. As a result, the loss in utility with respect to the oracle is larger compared to that obtained for January. This demonstrates the cost of neglecting the minimum energy constraint at the beginning of the day.

Next, we analyze the results on the same days by taking the user activity into account. We identify the periods of low activity, primarily the intervals categorized as *sleep*, and constrain the allocations in those intervals as $E_t^c = M_E$. We add the same constraint to the offline optimization for fairness. Comparing Figure 3.5 to Figure 3.7 shows that the algorithm starts allocating less energy at night. As a result, more energy is reserved for higher activity intervals, which leads to more than 30% increase in the utility during those intervals. Like before, the results match very closely with the offline optimization results. Incorporating user activity leads to even more savings in the results obtained for July. When we account for user activity, the proposed algorithm does not over-allocate at the early hours since there is little activity at night. Therefore, the battery energy does not hit to E_{min} , and our results coincide with the oracle results, as shown in Figure 3.8.

3.5.3 Comparison to Offline Optimization

Improving the duty ratio is an important end goal. Therefore, this section compares the duty ratio obtained with the proposed approach against the offline optimization results, which employ an oracle. We performed the comparisons for three different users from the US Department of Labor [163] database over 12 months. Figure 3.9 summarizes the normalized duty ratio (our results divided by the offline optimization results). We observe that the duty ratio provided by our approach is, on average, within 1% of the duty ratio achieved by the oracle. Moreover, the largest loss in optimality in the duty is less than 5%. We observe a bigger loss

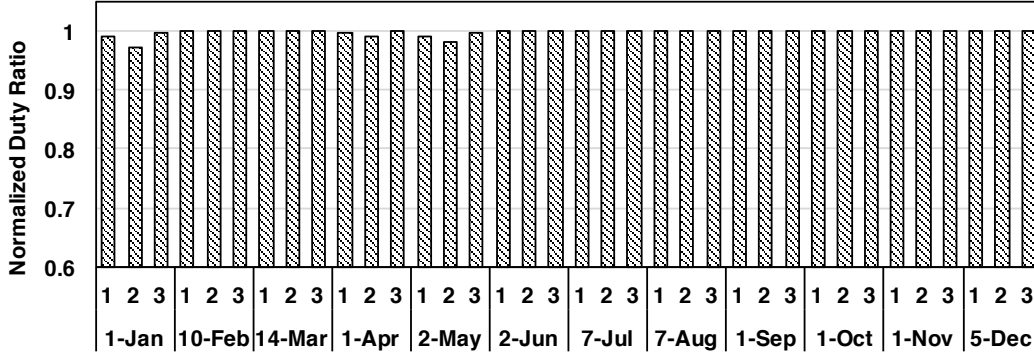


Figure 3.9: Comparison of the proposed solution to offline optimization for three users over 12 months.

under two conditions. First, when the variation between the expected and actual energy generation is large, the results of the proposed algorithm degrade, as anticipated. Second, when the peak-to-peak variation in the harvested energy becomes comparable to the battery capacity ($\sim 25\%$ of E_{target}), the proposed algorithm hits the E_{min} target, as shown in Figure 3.6.

3.6 Summary

Wearable IoT devices have great potential to enable health monitoring, activity tracking, and gesture-based control applications. However, they face severe energy limitations due to weight and cost constraints. Therefore, harvesting energy from ambient sources, such as light and body heat, and using it optimally is critical for their success. This chapter presented a near-optimal runtime algorithm for self-powered wearable IoT devices. The proposed approach is based on two observations that lead to near-optimal results with constant time complexity. First, we obtain a closed-form solution for the optimization problem by relaxing the minimum battery energy constraint. Then, we use the expected energy that will be harvested throughout the day to solve the relaxed finite-horizon optimization problem. Finally, we account for the

deviations from the expected values and enforce the minimum energy constraints at runtime. We demonstrate that our results are, on average, within 3% of optimal values computed offline using an oracle. The results degrade as the peak-to-peak variation in the harvested energy and deviation from the expected values increase. However, the degradation in the utility is small when the battery capacity can absorb the peak-to-peak variations.

Appendix: Derivation of Equation 3.6

To solve the optimization problem given in Equation 3.5, we first evaluate the Lagrangian of the objective function as:

$$L = \sum_{t=0}^{T-1} \beta^t \left[\ln \left(\frac{E_t^c}{M_E} \right)^\alpha - \lambda_t (E_{t+1}^B + E_t^c - E_t^B - \eta_t E_t^H) \right] + \sum_{t=0}^{T-1} \beta^t \mu_t [E_{t+1}^B - E_{min}] + \beta^{T-1} \mu_{T-1} [E_T^B - E_{target}] \quad (3.12)$$

Using the Lagrangian, we can write the first-order conditions as:

$$\frac{\partial L}{\partial E_t^c} : \quad \beta^t \left[\frac{\alpha M_E}{E_t^c} - \lambda_t \right] = 0 \quad 0 \leq t \leq T-1 \quad (3.13)$$

$$\frac{\partial L}{\partial E_{t+1}^B} : \quad -\beta^t \lambda_t + \beta^t \mu_t + \lambda_{t+1} \beta^{t+1} = 0 \quad 0 \leq t \leq T-1 \quad (3.14)$$

$$\frac{\partial L}{\partial E_T^B} : \quad -\beta^{T-1} \lambda_{T-1} + \beta^{T-1} \mu_{T-1} = 0 \quad i.e., \lambda_{T-1} = \mu_{T-1} \quad (3.15)$$

In addition to the first-order conditions above, the Karush-Kuhn-Tucker (KKT) conditions are $\mu_t \geq 0, \lambda_t \geq 0$, and:

$$\mu_t (E_{t+1}^B - E_{min}) = 0 \quad 0 \leq t \leq T-1 \quad (3.16)$$

$$\mu_{T-1} (E_T^B - E_{target}) = 0 \quad (3.17)$$

Since $u(E_t^c)$ is concave, Equations 3.13-3.17 give the necessary and sufficient conditions for the optimality [91]. We present a step-by-step solution below.

1. Boundary Condition: Lagrangian multipliers λ_t can be found using Equation 3.13 as :

$$\lambda_t = \frac{\alpha M_E}{E_t^c} \quad 0 \leq t \leq T - 1 \quad (3.18)$$

Combining this relation with Equation 3.15, we can conclude that $\lambda_{T-1} = \mu_{T-1} \neq 0$. Hence, the complementary slackness given by Equation 3.17 implies $E_T^B = E_{target}$.

2. Recursion over E_t^c : We can use Equation 3.14 to derive a recursion rule for λ_t and combine it with Equation 3.18 as follows:

$$\begin{aligned} \beta \lambda_{t+1} &= \lambda_t - \mu_t & 0 \leq t \leq T - 1 \\ \frac{\alpha \beta M_E}{E_{t+1}^c} &= \frac{\alpha M_E}{E_t^c} - \mu_t & 0 \leq t \leq T - 1 \end{aligned} \quad (3.19)$$

We plug the boundary condition $E_T^B = E_{target}$ to this recursive relation. Then, the energy allocations in earlier intervals can be solved using Equation 3.5 and the KKT condition given by Equation 3.16.

Solving T nonlinear equations at runtime is not efficient. However, *tentatively ignoring* the minimum energy constraint (*key insight 1*), enables us to eliminate μ_t from Equation 3.19. That is, we can set $\mu_t = 0$ in equations 3.16 and 3.19. Similarly, E_t^H is not known a priori, but we use the expected values (*key insight 2*). The proposed algorithm presented in Section 3.4.3 enables us to make up for these choices at runtime.

3. Closed-form Solution: After setting $\mu_t = 0$, $0 \leq t \leq T - 1$, Equation 3.19 reduces to:

$$E_{t+1}^c = \beta E_t^c \quad (3.20)$$

We can re-arrange the battery energy dynamics in Equation 3.5, and combine with this relation as follows:

$$\begin{aligned} E_0^c &= E_0^B - E_1^B + \eta_0 E_0^H, \quad \beta E_0^c = E_1^B - E_2^B + \eta_2 E_2^H, \dots \\ \beta^{T-1} E_0^c &= E_{T-1}^B - E_{target} + \eta_{T-1} E_{T-1}^H \end{aligned}$$

Note that E_{target} in the last equation comes from the boundary condition. When we summing up these T equations, $E_1^B - E_{T-1}^B$ cancel each other. Hence, we find E_0^c as:

$$E_0^c = \frac{E_0^B - E_{target} + \sum_{t=0}^{T-1} \eta_t E_t^H}{1 + \beta + \beta^2 + \dots + \beta^{T-1}} \quad (3.21)$$

Combining Equation 3.20 and Equation 3.21 gives the closed-form solution summarized in Equation 3.6. \square

Chapter 4

ONLINE HUMAN ACTIVITY RECOGNITION USING LOW-POWER WEARABLE DEVICES

4.1 Introduction

Advances in wearable electronics has the potential to disrupt a wide range of health applications [49, 115]. For example, diagnosis and follow-up for many health problems, such as motion disorders, depend currently on the behavior observed in a clinical environment. Specialists analyze the gait and motor functions of patients in a clinic and prescribe therapy accordingly. As soon as the patient leaves the clinic, there is no way to continuously monitor the patient and report potential problems [53, 128]. Another high-impact application area is obesity-related diseases, which claim about 2.8 million lives every year [8, 174]. Automated tracking of physical activities of overweight patients, such as walking, offers tremendous value to health specialists since self-recording is inconvenient and unreliable.

There has been growing interest in human activity recognition with the prevalence of low-cost motion sensors and smartphones. For example, accelerometers in smartphones are used to recognize activities such as stand, sit, lay down, walking, and jogging [5, 67, 93]. This information is used for rehabilitation instruction, fall detection of the elderly, and reminding users to be active [80, 168]. Furthermore, activity tracking also facilitates physical activity, which improves the wellness and health of its users [30, 34, 87]. The successful design of activity recognition algorithms depends critically on the availability of sensor data that captures the activities of interest. Research studies typically employ wearable inertial sensors or smartphones to collect the data while the users are performing the activities of interest. The data is then used

to train and evaluate algorithms for activity recognition. However, the data is rarely made publicly available [112]. As a result, it is difficult to reproduce the results and obtain comparisons with existing approaches. Therefore, there is a critical need for open-source datasets that provide a common platform for HAR research. This chapter presents the wearable HAR (w-HAR) dataset to address the need for open-source datasets.

HAR techniques can be broadly classified based on when training and inference take place. Early work collects the sensor data before processing. Then, both classifier design and inference are performed offline [10]. Hence, they have limited applicability. Most recent work trains a classifier offline and processes the sensor data online to infer the activity [5, 153]. However, to date, there is no technique that can perform *both online training and inference*. Online training is crucial since it needs to adapt to new, and potentially large number of, users who are not involved in the training process. To this end, *this chapter presents the first HAR technique that continues to train online to adapt to its user*.

The vast majority, if not all, of recent HAR techniques employ smartphones. The major motivations behind this choice are their widespread use and easy access to integrated accelerometer and gyroscope sensors [168]. We argue that smartphones are not suitable for HAR for three reasons. First, patients cannot always carry a phone as prescribed by the doctor. Even when they have the phone, it is not always in the same position (e.g., at hand or in pocket), which is typically required in these studies [35, 153]. Second, mobile operating systems are not designed for meeting real-time constraints. For example, the Parkinson’s Disease Dream Challenge [111] organizers shared raw motion data collected using iPhones in more than 30K experiments. According to the official spec, the sampling frequency is 100 Hz. However, the actual sampling rate varies from 89 Hz to 100 Hz, since the phones continue to perform many unintended tasks during the experiments. Due to the same reason, the power consumption is

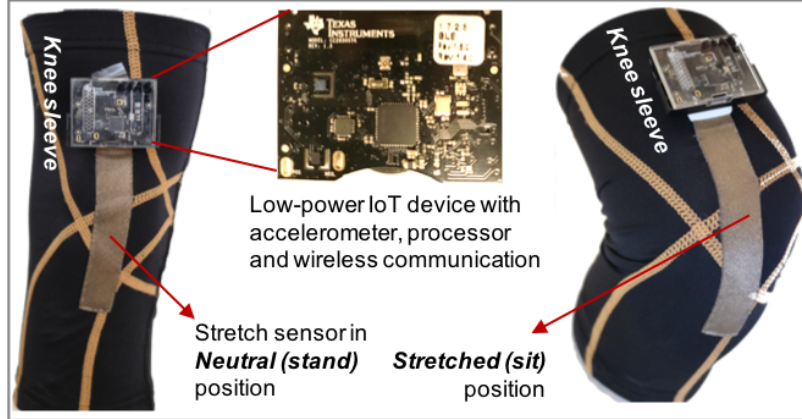


Figure 4.1: Wearable system setup, sensors, and the low-power IoT device [160]. We knitted the textile-based stretch sensor to a knee sleeve to accurately capture the leg movements

in the order of watts (more than $100\times$ of our result). Finally, researchers are limited to sensors integrated in the phones, which are not specifically designed for human activity recognition.

This chapter first presents wearable HAR (w-HAR), an opensource dataset for HAR. Our dataset is collected using the wearable system shown in Figure 4.1. The setup shown in Figure 4.1 integrates an IMU and textile-based wearable stretch sensors to provide two modalities of motion data. In contrast, other HAR datasets [6, 112, 140, 179] typically use accelerometers and gyroscopes as their primary sensors. However, accelerometers and gyroscopes are notoriously noisy, leading to challenges in data segmentation and classification. The stretch sensor provides low-noise motion data that allows us to generate non-uniform activity segments ranging from one to three seconds. Using the wearable setup, we first perform extensive data collection with 22 user subjects. We record the IMU (accelerometer and gyroscope) and stretch sensor data of each user while they perform activities in the set $\{jump, lie\ down, sit, stand, stairs\ down, stairs\ up, walk\}$. Then, we manually label the data such that it can be used to train machine learning algorithms. w-HAR is the first dataset in the literature that includes both IMU and stretch sensor data. The dataset has been publicly released along with this paper to enable further research on activity recognition algorithms.

We provide three versions of the dataset such that users can choose the most appropriate version for their application. The first version includes the raw data obtained from the sensors without any pre-processing. This version is most useful when users want to develop their own segmentation and pre-processing algorithms for HAR, along with feature generation and classifier design. The second version of the dataset uses the segmentation algorithm in [15] to generate variable-length segments. This version allows users to develop their own features and classifiers. Finally, the third version provides the set of features used in our work, such that users can focus solely on classifier design.

In addition to the dataset, we also present a comprehensive framework for designing human activity recognition classifiers. Our framework consists of the following steps:

Segmentation: The first step in our framework is to divide the data into segments such that each segment contains a single activity. Most prior studies on HAR divide the sensor data fixed length windows [8, 93] or smoothen noisy accelerometer data over long durations [35] (detailed in Section 4.2). However, this is not ideal as a single window may contain portions of multiple activities, leading to challenges in classification. In contrast, we develop a novel algorithm that uses data from the stretch sensor data to generate variable length activities windows, ensuring that each window contains a single activity.

Feature Generation: The stretch sensor accurately captures the periodicity in the motion. Hence, its fast Fourier transform (FFT) reveals invaluable information about activity in different frequency bands. Therefore, we choose the leading coefficients as features in our classification algorithm. Unlike the stretch sensor, the accelerometer data is known to be noisy. Therefore, we employ the approximation coefficients of its discrete wavelet transform (DWT) to capture the behavior as a function of time. We note that our HAR framework only uses acceleration data from the IMU, while both accelerometer and gyroscope data are included in our public dataset.

Design Space Exploration for Offline Classifier: It is critical to choose a classifier that is both robust and resource-efficient. Resource efficiency is important to ensure that the classifier can execute on wearable devices with power and memory constraints. To this end, we start with commonly used classifiers such as neural networks, random forest, support vector machine (SVM) and k-nearest neighbor (k-NN). Among these, we focus on neural networks, since they can be easily updated online using both reinforcement learning (RL) [156] and supervised learning techniques with low overhead. Once we choose neural networks as our classifier, we perform a design space exploration (DSE) to determine the appropriate structure for the network. The DSE helps us in ensuring that the classifier is robust to multiple sets of users while satisfying the requirement of low resource requirements to run on the wearable device.

Online Learning: State-of-the-art approach for HAR typically train classifiers online and only perform the activity classification online [5, 153]. This approach is not scalable when the device is used by new users with potentially different activity characteristics. Therefore, we also perform online training of the classifiers such that it can adapt to new users who are not involved in the training process. We make use of two approaches to continuously update the weights of the neural network as a function of the feedback available from users. When users can provide the actual activity performed, we use incremental supervised learning updates. Otherwise, we use the policy gradient algorithm [156]. Experiments with our dataset show that these algorithms can improve the accuracy by as much as 40% of unseen users.

Finally, this chapter is the first to provide a detailed power consumption and performance break-down of sensing, processing, and communication tasks. We implement the proposed framework on the TI-CC2650 MCU [160] and present an extensive experimental evaluation using data from nine users and a total of 2614 activity windows. Our approach provides 95% overall recognition accuracy with 27.60 ms processing time, 1.13 mW sensing, and 11.24 mW computation power consumption.

In summary, the major contributions of this chapter are as follows:

- An activity recognition dataset with accelerometer and stretch sensor data from 22 users
- A novel technique to segment the sensor data non-uniformly as a function of the user motion,
- Design space exploration of neural networks to choose the structure of the offline classifier such that it is robust to input from different users
- An online learning algorithm using incremental supervised learning that provides faster convergence when compared to reinforcement learning
- Experimental validation of the DSE and online learning algorithms on a low-power wearable device using the w-HAR dataset

The rest of the chapter is organized as follows. We review the related work in Section 4.2. Then, we present the w-HAR dataset in Section 4.3. The feature generation and classifier design techniques are presented in Section 4.4 and Section 4.5, respectively. Section 4.6 presents the two online learning algorithms. Finally, the experimental results are presented in Section 4.7, and our conclusions are summarized in Section 4.8.

4.2 Related Work and Novelty

Human activity recognition has been an active area of research due to its applications in health monitoring, patient rehabilitation, and promoting physical activity among the general population [8, 29, 30]. Advances in sensor technology have enabled activity recognition to be performed using body-mounted sensors [131]. Typical steps for activity recognition using sensors include data collection, segmentation, feature extraction, and classification.

Several prior studies have presented datasets for HAR [6, 112, 140, 179]. Most of the datasets presented earlier focus on acquiring data from smartphones and performing HAR on

them. For instance, Micucci et al. [112] present a dataset for HAR using accelerometers on smartphones. The dataset includes data from thirty subjects and nine activities of daily living. The authors also present a review of other publicly available datasets using smartphones. Wearable sensors have also been used in HAR datasets since multiple devices can be easily mounted on different parts of the body. For instance, the Opportunity dataset presented in [140] uses multiple inertial measurement units and accelerometers to collect data from four users. Similarly, Zhang et al. [179] use a single motion sensing unit to obtain data from 14 users. While these datasets are useful for HAR, they primarily contain data from accelerometers, which is known to be noisy. As a result, studies using these datasets resort to fixed-length windows, instead of creating a window for each activity. In contrast, the dataset presented in this paper includes data from a textile-based stretch sensor and IMU, which allows us to create variable-length segments tailored to each activity.

HAR studies typically use a fixed window length to infer the activity of a person [8, 93]. For instance, the studies in [8, 93] use 10-second windows to perform activity recognition. Increasing the window duration improves accuracy [29] since it provides richer data about the underlying activity. However, transitions between different activities cannot be captured with long windows. Moreover, fixed window lengths rarely capture the beginning and end of an activity. This leads to inaccurate classification as the window can have features of two different activities [29]. A recent approach proposes action segmentation using a step detection algorithm on the accelerometer data [35]. Since the accelerometer data is noisy, they need to smoothen the data using a one-second sliding window with a 0.5-second overlap. Hence, this approach is not practical for low-cost devices with limited memory capacity. Furthermore, the authors state that there is a strong need for better segmentation techniques to improve the accuracy of HAR [35]. To this end, we present a robust segmentation technique that produces windows whose sizes vary as a function of the underlying activity.

Most existing studies employ statistical features such as mean, median, minimum, maximum, and kurtosis to perform HAR [8, 93, 129]. These features provide useful insight, but there is no guarantee that they are representative of all activities. Therefore, a number of studies use all the features or choose a subset of them through feature selection [129]. Fast Fourier transform and more recently discrete wavelet transform have been employed on accelerometer data. For example, the work in [35] computes the 5th order DWT of the accelerometer data. Eventually, it uses only a few of the coefficients to calculate the wavelet energy in the 0.625 - 2.5 Hz band. In contrast, we use only the approximation coefficients of a single level DWT with $O(N/2)$ complexity. Unlike prior work, we do not use the FFT of the accelerometer data, since it entails significant high-frequency components without clear implications. In contrast, we employ leading FFT coefficients of the stretch sensor data, since it gives a very good indication of the underlying activity.

Early work on HAR used wearable sensors to perform data collection while performing various activities [10]. This data is then processed offline to design the classifier and perform the inference. However, offline inference has limited applicability since users do not get any real-time feedback. Therefore, recent work on HAR has focused on implementation on smartphones [5, 30, 73, 153]. Compared to wearable HAR devices, smartphones have limited choice of sensors and high power consumption. In addition, results on smartphones are harder to reproduce due to the variability in different phones, operating systems, and usage patterns [34, 153].

Finally, existing studies on HAR approaches employ commonly used classifiers, such as k-NN [58], support vector machines [58], decision trees [135], and random forest [58], which are *trained offline*. In strong contrast to these methods, the proposed framework is the first to enable *online training*. We first train a neural network offline to generate an initial implementation of the HAR system. Then, we use reinforcement learning or incremental supervised learning at runtime to improve the accuracy of the system for new users. We envision that

these algorithms will enable personalized HAR devices that adapt continuously to the unique activity pattern of their users.

4.3 Human Activity Recognition Dataset

The availability of datasets is crucial for human activity recognition research. Therefore, we open source our dataset to enable further research in this area. The dataset in this chapter is the first to integrate readings from a wearable stretch sensor and an inertial motion unit (IMU). The presence of the stretch sensors adds an additional modality that allows us to create variable-length segments. The variable-length segments make it easier for the classifiers to recognize activities, as we show in the experiments. In this section, we describe the details of the data collection setup, protocol, user demographics, and the labeling process.

4.3.1 Wearable System Setup

We use a combination of Invensense-9250 IMU and stretch sensors to collect the data, as shown in Figure 4.1. The IMU is integrated into the TI-CC2650 Sensortag device, and the stretch sensor is another discrete module. We mount the IMU on the right ankle of the user since this allows for a maximum swing of the sensor [67]. The stretch sensor is sewed to a knee sleeve, as shown in Figure 4.1. During the experiment, the user wears the sleeve on the knee to capture the knee movements while performing the activities. Both the sensor devices are equipped with the Bluetooth low energy (BLE) protocol for communication. Using the BLE protocol, the sensors transmit the data to a smartphone which stores the data to a file. In our future work, we plan to integrate the IMU and the stretch sensor into a single device such that a single stream of data can be transmitted. To synchronize the data from the sensors, we record

Table 4.1: List of activities used in the HAR framework

• Jump (J)	• Lie Down (L)	• Sit (S)
• Stand (St)	• Walk (W)	• Stairs Up (SU)
• Stairs Down (SD)	• Transition (T) between the activities	

the wall clock time for each data sample from the sensors. Then, using the offset between the sensors, we align the sensor readings using the approach in [154].

Wearable System Sensor Parameters: We sample the IMU at 250 Hz and the stretch sensor at 25 Hz. These sampling frequencies are sufficient to capture the frequency of human movements, which are in the order of a few Hz. We use a significantly higher frequency for the accelerometer since it typically exhibits higher noise. Therefore, the higher sampling frequency allows us to smooth and sub-sample the data using a moving average filter while preserving the data signatures.

4.3.2 User Studies

We obtain motion data from 22 users with the wearable device setup. The users are recruited using the snowball sampling technique. Each user signed a consent form as approved by the institutional review board at Arizona State University. We experiment with a total of 22 users (consisting of 14 males and 8 females), with ages 20–45 years and heights 150–180 cm. The set of activities performed by the users is summarized in Table 4.1. Each user performs a series of experiments shown in Table 4.2. In addition to this protocol, we also perform experiments where the users are free to perform any activities they choose. Next, we perform the labeling of the dataset, as described below.

Data Labeling: After collecting the data from the users, we use the segmentation algorithm to divide the data into variable-length windows. Then, the generated windows are analyzed

Table 4.2: Experimental protocol for the HAR dataset

Experiment 1	Experiment 2	Experiment 3	Experiment 4
	Stand 10 s		
Stand 30 s	Sit 30 s	Stand 10 s	Stand 10 s
Jump 3 times	Stand 10 s	Walk 40 steps	Jump 3 times
Stand 30 s	Jump 3 times	Stand 10 s	Walk 40 steps
	Sit 30 s		Sit 20 s
Experiment 5	Experiment 6	Experiment 7	
Stand 10 s	Stand 10 s	Stand 10 s	
Sit 10 s	Walk down	Walk up stairs	
Lie down 30 s	stairs	Stand 10 s	
Sit 10 s	Stand 10 s		

by four human experts to assign the labels. The same labels are also assigned to each sample in the raw data before segmentation such that we know the user’s activity in each sampling period. Moreover, we revisit the assigned labels during the testing phase of the HAR classifier to ensure that the errors made by the classifier are not due to mislabeling.

4.3.3 Dataset Description

After labeling the data, we generate three versions of the dataset for public release as follows.

Raw Data: This version of the contains the raw data obtained from the stretch and IMU (accelerometer + gyroscope) sensors without any pre-processing. We synchronize the stretch and IMU data such that the time indices for both sensors are aligned. Consequently, users do not have to run any synchronization algorithms on the data. The raw data version of w-HAR is ideal for researchers who want to design their own algorithms for all steps of HAR from segmentation to classification.

Segmented Data: The segmented dataset uses the segmentation algorithm proposed in [15]

Table 4.3: Summary of the number of segments in each activity

Activity	Segments	Activity	Segments
Jump	458	Walk	2007
Lie down	474	Stairs up	109
Sit	696	Stairs down	99
Stand	620	Transition	277

and summarized in Section 4.4.2 to generate variable-length activity segments. This version is suitable for users who want to focus on feature generation and classification algorithms. Breakdown of the total segments of each activity is summarized in Table 4.3.

Feature Data: We also release the features used [15] as part of w-HAR. The features included in this version are summarized in Section 4.4.3. The feature data version allows users to focus on developing classifiers for HAR and obtain reproducible comparisons among different algorithms.

In summary, w-HAR includes a total of 4740 segments with a total duration of about 3 hours. The dataset and the corresponding source code for our algorithms are available for download at our GitHub page: <https://github.com/gmbhat/human-activity-recognition>.

4.3.4 Flow for Using the w-HAR Dataset

This section describes the flow for incorporating the w-HAR dataset for either developing new algorithms or results reported in this chapter, as shown in Figure 4.2. The first step after obtaining the dataset is to segment the raw data into windows. To this end, users can either use the windows provided along with the dataset or develop their own segmentation algorithm, as shown using paths 1a and 1b, respectively. The next step is to generate features for each window generated by the segmentation algorithm. Here, the users are free to generate their

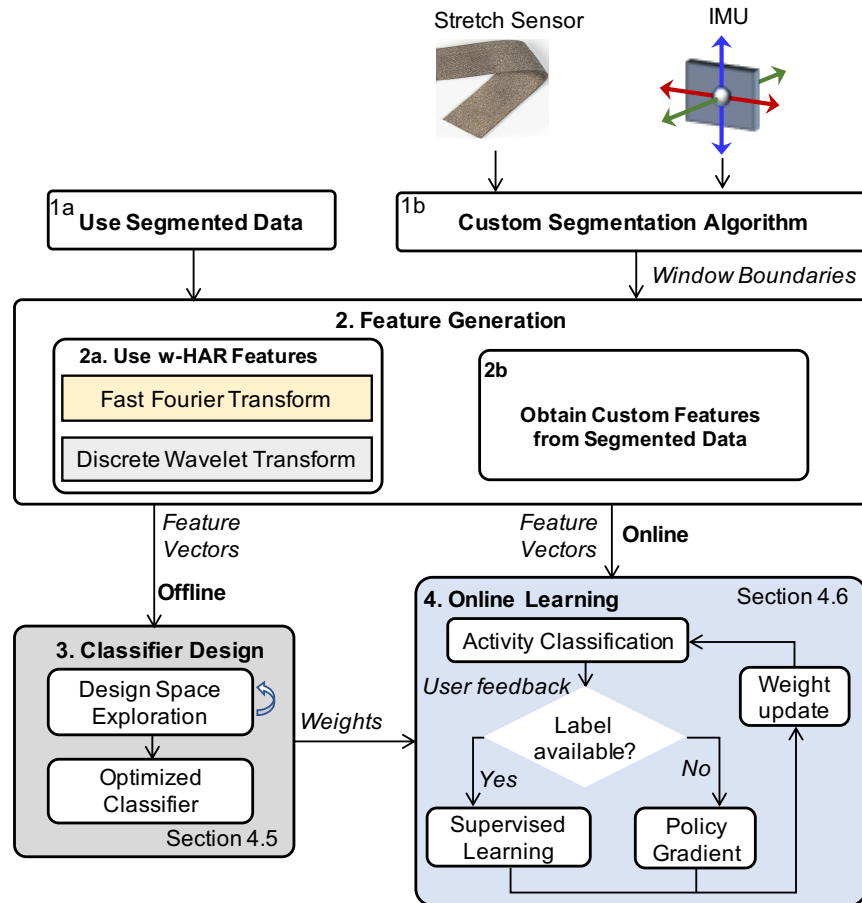


Figure 4.2: Flow-chart for using the w-HAR data set to test new algorithms or reproduce the results presented in this chapter

own features or use the baseline feature set provided with the dataset. The feature data and the labels are then used to design a classifier for activity classification. The classifier design involves a design space exploration for determining the optimal classifier, as shown in Figure 4.2. We also provide a baseline neural network classifier such that it is easy to obtain reproducible comparisons with new approaches. Finally, the last step of the design flow is to use the classifier designed offline to identify activities at runtime. In this step, user feedback is used to update the weights of the classifier to improve the accuracy of the classifier continuously. As shown in Figure 4.2, our implementation of the framework uses either reinforcement learning

or incremental supervised learning depending on the level of user feedback available. We envision that further research on HAR using our dataset will enable new online learning algorithms for personalized activity recognition and healthcare. Next, we describe the segmentation algorithm used to generate the segmented data in w-HAR. We also go over the feature set that is released as part of w-HAR.

4.3.5 Comparisons with Existing HAR Datasets

This section presents a comparative analysis of our dataset with other publicly available HAR datasets. We choose the datasets that focus on the activities that are common to our dataset and use either smartphones or wearables as their data collection device. Table 4.4 summarizes the major characteristics of the datasets. All the previous datasets use a single modality of sensing, i.e., the accelerometer. In contrast, the proposed dataset is the first one to integrate data from accelerometer and stretch sensor. Furthermore, our dataset also includes gyroscope data for use by other researchers in their algorithms. Having data only from the accelerometer limits previous datasets to fixed-length windows, as shown in the last column of the table. The only exception is DU-MD dataset [143], which reports variable-length segments. However, variable-length segments in DU-MD are obtained manually, thus making the approach unsuitable for runtime algorithms. The proposed dataset overcomes this problem by using the stretch sensor data to enable variable-length segments that can be obtained at runtime. We acknowledge that the number of user subjects in our dataset is lower compared to some previous studies. We plan to resolve this by continuing to augment our dataset in the future.

Table 4.4: Comparison with existing HAR datasets

Dataset	Device	Sensors	No. of Subjects	No. of Activities	Variable-length Segments
DU-MD [143]	Wearable	Accelerometer	33	7	Yes, manually
Shoaib et al. [152]	Smartphone	Accelerometer	10	7	No
UCI HAR [6]	Smartphone	Accelerometer	30	6	No
Ugulino et al. [162]	Wearable	Accelerometer	5	4	No
UniMiB SHAR [112]	Smartphone	Accelerometer	30	9	No
USC-HAD [179]	Wearable	Accelerometer	14	12	No
WISDM [93]	Smartphone	Accelerometer	29	6	No
w-HAR	Wearable	Accelerometer, Stretch sensor	22	7	Yes

4.4 Feature Set and Classifier Design

4.4.1 Goals and Problem Statement

The goal of the proposed HAR framework is to recognize the seven common daily activities listed in Table 4.1 and the transitions between them *in real-time* with *more than 90% accuracy under mW power range*. These goals are set to make the proposed system practical for daily use. The power consumption target enables day-long operation using ultra-thin lithium polymer cells [51].

The stretch sensor is knitted to a knee sleeve, and the IoT device with a built-in accelerometer is attached to it, as shown in Figure 4.1. All the processing outlined in Figure 4.2 is performed locally on the IoT device. More specifically, the streaming stretch sensor data is processed to generate segments ranging from one to three seconds (Section 4.4.2). Then, the raw accelerometer and stretch data in each window are processed to produce the features used by the classifier (Section 4.4.3). Finally, these features are used for both online inference (Section 4.5) and online learning (Section 4.6). Since communication energy is significant, *only the*

recognized activity and time stamps are transmitted to a gateway, such as a phone or PC, using Bluetooth whenever they are nearby (within 10m). The following sections provide a theoretical description of the proposed framework without tying them to specific parameter values. These parameters are chosen to enable a low-overhead implementation using streaming data. The actual values used in our experiments are summarized in Section 4.7.1 while describing the experimental setup.

4.4.2 Sensor Data Segmentation

Activity windows should be sufficiently short to capture transitions and fast movements, such as fall and jump. However, short windows can also waste computation time and power for idle periods, such as sitting. Furthermore, a fixed window may contain portions of two different activities, since perfect alignment is not possible. Hence, activity-based segmentation is necessary to maintain high accuracy with minimum processing time and power consumption.

To illustrate the proposed segmentation algorithm, we start with the snapshot in Figure 4.3 from our user studies. Both the 3-axis accelerometer and stretch sensor data are preprocessed using a moving average filter similar to prior studies. The unit of acceleration is already normalized to gravitational acceleration. The stretch sensor outputs a capacitance value that changes as a function of its state. This value ranges from around 390 pF (neutral) to close to 500 pF when it is stretched [117]. Therefore, we normalize the stretch sensor output by subtracting its neutral value and scaling by a constant: $s(t) = [s_{raw}(t) - \min(s_{raw})]/S_{const}$. We adopted $S_{const} = 8$ to obtain a comparable range to accelerometer readings. First, we note that the 3-axis accelerometer data exhibits significantly larger variations compared to the normalized stretch capacitance. Therefore, decisions based on accelerations are prone to false hits [35]. In

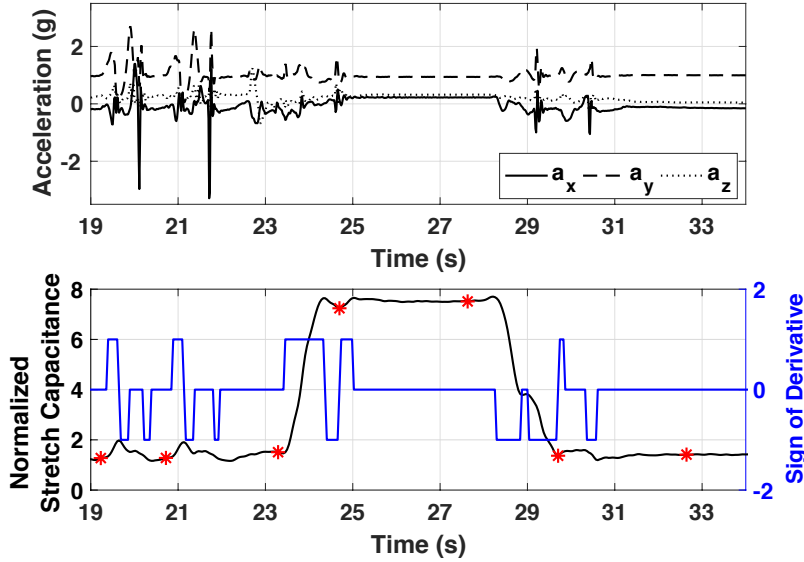


Figure 4.3: Illustration of the segmentation algorithm

contrast, we propose a robust solution that generates the segments specified with red * markers in Figure 4.3.

The boundaries between different activities can be identified by detecting the deviation of the stretch sensor from its neutral value. For example, the first segment in Figure 4.3 corresponds to a step during walk. The sensor value starts increasing from a local minimum to a peak at the beginning of the step. The beginning of the second segment ($t \approx 21$ s) exhibits similar behavior since it is another step. Although the second step is followed by a longer neutral period (the user stops and sits to a chair at $t \approx 23$ s), the beginning of the next segment is still marked by a rise from a local minimum. In general, we can observe a distinct minimum (fall followed by rise as in walk) or a flat period followed by a rise (as in walk to sit) at the boundaries of different activity windows. Therefore, the proposed segmentation algorithm monitors the derivative of the stretch sensor to detect the activity boundaries, as outlined in Algorithm 1.

We employ the 5-point derivative formula given below to track the trend of the sensor

Algorithm 1: Segmentation Algorithm

```
1 Input: Stretch Sensor Samples
2 Initialize: FIFO buffer  $S$  of length 5 to store the stretch samples
3 Initialize: FIFO buffer  $D$  of length 3 to store the trend of the derivative
4 Initialize:  $trend_{prev} \leftarrow \text{flat}$ 
5 Initialize:  $trend \leftarrow \text{flat}$ 
6 for Time index  $t = 1, 2, 3, \dots$  do
7   Append sample  $s(t)$  to buffer  $S$ 
8    $s'(t) \leftarrow$  Five-point derivative using Equation 4.1
9   Append  $s'(t)$  to buffer  $D$ 
10  if ( $D(0) > 0$  and  $D(1) > 0$  and  $D(2) > 0$ ) then
11     $trend \leftarrow \text{increasing}$ 
12  end
13  else if ( $D(0) < 0$  and  $D(1) < 0$  and  $D(2) < 0$ ) then
14     $trend \leftarrow \text{decreasing}$ 
15  end
16  else if ( $D(0) = 0$  and  $D(1) = 0$  and  $D(2) = 0$ ) then
17     $trend \leftarrow \text{flat}$ 
18  end
19  if One second since last segment then
20    if ( $trend_{prev} = \text{flat}$  and  $trend = \text{increasing}$ ) OR
21      ( $trend_{prev} = \text{decreasing}$  and  $trend = \text{increasing}$ ) then
22      Mark a new segment at time index  $t$ 
23    end
24  if Three seconds since last segment then
25    Mark a new segment at time index  $t$ 
26  end
27   $trend_{prev} \leftarrow trend$ 
28 end
```

value:

$$s'(t) = \frac{s(t-2) - 8s(t-1) + 8s(t+1) - s(t+2)}{12} \quad (4.1)$$

where $s(t)$ and $s'(t)$ are the stretch sensor value and its derivative time step t , respectively. When the derivative is positive, we know that the stretch value is *increasing*. Similarly, a negative value means a decrease, and $s'(t) = 0$ implies a flat region. Looking at a single data point can catch sudden peaks and lead to false alarms. To improve the robustness, one can

look at multiple consecutive data points before determining the trend. In our implementation, we conclude that the *trend* changes only if the last three derivatives consistently signal the new trend (lines 10–18 in Algorithm 1). For example, if the current trend is flat, we require that the derivative is positive for three consecutive data points to filter glitches in the data point. Whenever we detect that the trend changes from flat or decreasing to positive, we produce a new segment. The final consideration in the segmentation algorithm is to bound the length of the windows from above and below. In order to bound the length of a segment from below, we start looking for new segments only after a fixed amount of time (one second in our implementation) has passed since the last window. The minimum window length check is performed by the *if* statement in line 19 of Algorithm 1. Lower bounding the window size saves computation time and power by preventing windows that are shorter than normal activity lengths. Similarly, we enforce a maximum window length to improve robustness in case a local minimum is missed. We use $t_{max} = 3$ s as the upper bound since it is long enough to cover all transitions.

Figure 4.4 shows the segmented data for the complete duration of the illustrative example given in Figure 4.3. The proposed approach clearly segments each step of walk. Moreover, it captures the transitions from walking to sitting and sitting to standing very well. Furthermore, the segments obtained from the algorithm for all the activities are shown in Figure 4.5 using red asterisks. The algorithm clearly marks each step in jump, walk, and stairs up/down activities. For sit, stand, and lie down activities, the algorithm uses a 3-second window whenever the sensor data is static. At the same time, whenever there is a transition, such as from sit to stand in Figure 4.5(c), the segmentation algorithm detects this and marks a new segment for the transition. This segmentation allows us to extract meaningful features from the sensor data, as described in the next section.

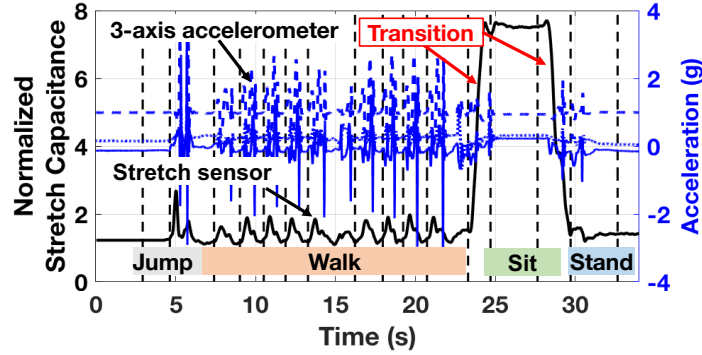


Figure 4.4: Illustration of the sensor data segmentation

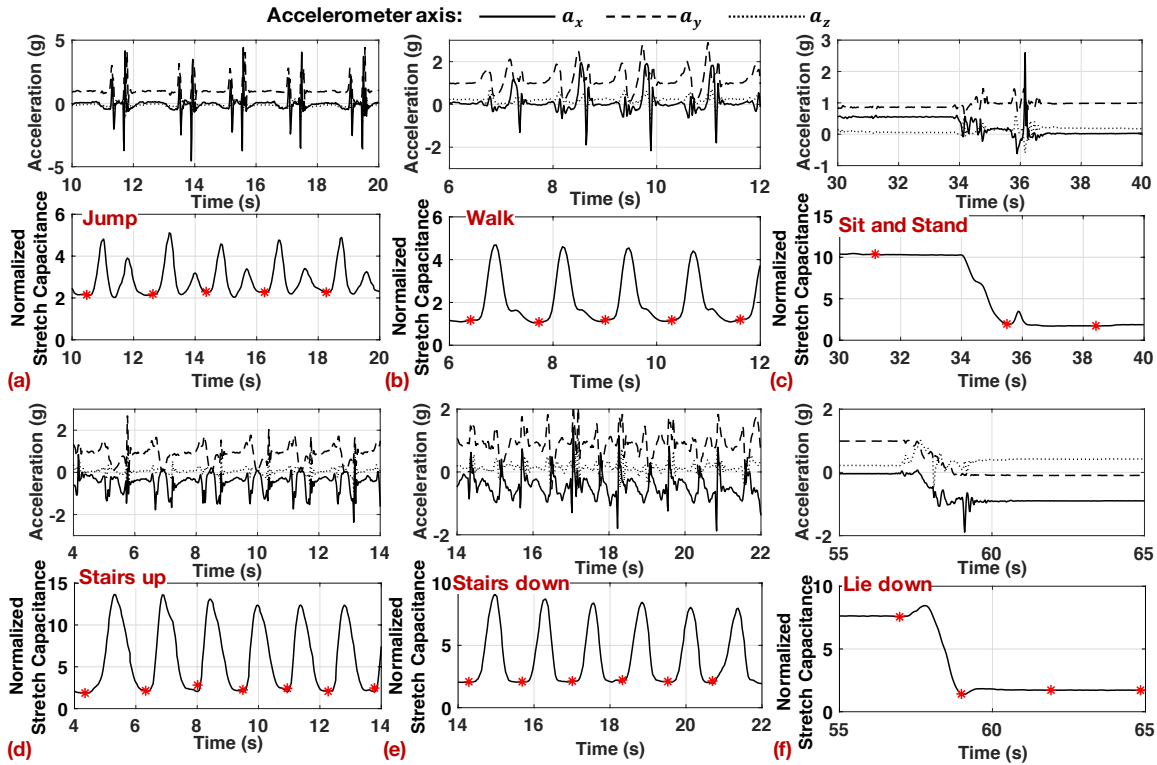


Figure 4.5: Visualization of segmentation for all activities in the HAR framework

4.4.3 Feature Generation

To achieve a high classification accuracy, we need to choose representative features that capture the underlying movements. We note that human movements typically do not exceed 10-Hz. Since statistical features, such as mean and variance, are not necessarily representative, we

focus on FFT and DWT coefficients, which have clear frequency interpretations. Prior studies typically choose the largest transform coefficients [153] to preserve the maximum signal power as in compression algorithms. However, sorting loses the frequency connotation, besides using valuable computational resources. Instead, we focus on the coefficients in the frequency bins of interest by preserving the number of data samples in each segment, as described next.

Stretch sensor features: The stretch sensor shows a periodic pattern for walking, and remains mostly constant during sitting and standing, as shown in Figure 4.4. As the level of activity changes, the segment duration varies in the (1,3] second interval. We can preserve a 10 Hz sampling rate for the longest duration (3 s during low activity) if we maintain $2^5 = 32$ data samples per segment. As the level of activity intensifies, the sampling rate grows to 32 Hz, which is sufficient to capture human movements. We choose a power of 2, since it enables efficient FFT computation in real-time. When the segment has more than 32 samples due to larger sensor sampling rate, we first sub-sample and smooth the input data as follows:

$$s_s[k] = \frac{1}{2S_R} \sum_{i=-S_R}^{S_R} s(tS_R + i), \quad 0 \leq k < 32 \quad (4.2)$$

where $S_R = \lfloor N/32 \rfloor$ is the subsampling rate, and $s_s[k]$ is the sub-sampled and smoothed data point. When there are less than 32 samples, we simply pad the segment with zeros.

After standardizing the size, we take the FFT of the current window and the previous window. We use two windows as it allows us to capture any repetitive patterns in the data. With a 32 Hz sampling rate during high activity regions, we cover $F_s/2 = 16$ Hz activity per Nyquist theorem. We observe that the leading 16 FFT coefficients, which cover the [0-8] Hz frequency range, carry most of the signal power in our experimental data. Therefore, they are used as features in our classifiers. The level of the stretch sensor also gives useful information. For instance, it can reliably differentiate sit from stand. Hence, we also add the minimum and maximum value of the stretch sensor to the feature set.

Accelerometer features: Acceleration data contains faster changes compared to the stretch data, even though the underlying human motion is slow. Therefore, we sub-sample and smoothen the acceleration to $2^6 = 64$ points following the same procedure given in Equation 4.2. Three-axis accelerometers provide acceleration a_x , a_y , and a_z along x -, y -, and z -axes, respectively. In addition, we compute the body acceleration excluding the effect of gravity g as $b_{acc} = \sqrt{a_x^2 + a_y^2 + a_z^2} - g$, since it carries useful information.

Discrete wavelet transform is an effective method to recursively divide the input signal to approximation A_i and detail D_i coefficients. One can decompose the input signal to $\log_2 N$ samples, where N is the number of data points. After one level of decomposition, A_1 coefficients in our data correspond to 0-32 Hz, while D_1 coefficients cover 32-64 Hz band. Since the former is more than sufficient to capture acceleration due to human activity, we only compute and preserve A_1 coefficients with $O(N/2)$ complexity. The number of features could be further reduced by computing the lower level coefficients and preserving the largest ones. As shown in the performance breakdown in Table 4.8, using the features in the neural network computations takes less time than computing the DWT coefficients. Moreover, keeping more coefficients and preserving the order maintains the shape of the underlying data.

Feature Overview: In summary, we use the following features:

Stretch sensor: We use 16 FFT coefficients, the minimum, and maximum values in each segment resulting in 18 features.

Accelerometer: We use 32 DWT coefficients for a_x , a_z , and b_{acc} . In our experiments, we use only the mean value of a_y , since no activity is expected in the lateral direction, and b_{acc} already captures its effect given the other two directions. This results in 97 features.

General features: The length of the segment also carries important information, since the number of data points in each segment is normalized. Similarly, the activity in the previous

window is useful to detect transitions. Therefore, we also add these two features to obtain a total of 117 features.

4.5 Classifier Design

4.5.1 Supervised Learning for State Classification

In the offline phase of our framework, the feature set is assigned a label corresponding to the user activity. Then, a supervised learning technique takes the labeled data to train a classifier that is used at runtime. Since one of our major goals is online training using reinforcement learning, we employ a cost-optimized neural network (NN). We also compare our solution to the most commonly used classifiers by prior work and provide brief explanations.

Support Vector Machine (SVM): SVM [58] finds a hyperplane that can separate the feature vectors of two output classes. If a separating hyperplane does not exist, SVM maps the data into higher dimensions until a separating hyperplane is found. Since SVM is a two-class classifier, multiple classifiers need to be trained for recognizing more than two output classes. Due to this, SVM is not suitable for reinforcement learning with multiple classes [94], which is the case in our HAR framework.

Random Forests and Decision Trees: Random forests [58] use an ensemble of tree-structured classifiers, where each tree independently predicts the output class as a function of the feature vector. Then, the class which is predicted most often is selected as the final output class. C4.5 decision tree [135] is another commonly used classifier for HAR. Instead of multiple trees, C4.5 uses a single tree. Reinforcement learning using random forests has been recently investigated in [127]. As part of the reinforcement learning process, additional trees are constructed and then a subset of trees is chosen to form the new random forest. The extra trees add additional

processing and memory requirements on the system, making it unsuitable for implementation on a wearable system with limited memory.

k-Nearest Neighbors (k-NN): k-Nearest Neighbors [58] is one of the most popular techniques used by many previous HAR studies. k-NN evaluates the output class by first calculating k nearest neighbors in the training dataset. Then, it chooses the class that is most common among the k neighbors and assigns it as the output class. k-NN requires storing all the training data locally. Since storing the training data on a wearable device with limited memory is not feasible, k-NN is not suitable for online training.

4.5.2 Proposed NN Classifier Design

Design space exploration for neural network: Choosing an appropriate structure for the NN is crucial to balance the classification accuracy with the resource requirements of the wearable device. A larger NN achieves a higher accuracy while increasing the memory and processing requirements of the device. Moreover, a larger neural network may also lead to overfitting to the training data, leading to a lower accuracy on new data samples. Therefore, it is crucial to choose the appropriate structure for the NN such that it is robust to new data while keeping the computational complexity low.

We choose the structure of the NN classifier by performing a design space exploration with varying number of hidden layers and neurons in each layer, as shown in Figure 4.6. Specifically, we first set the number of hidden layers and then vary the number of layers in each layer. Then, we train each of these configurations to obtain accuracy values. In order to ensure that the classifier is robust, we also obtain the accuracy of data not seen during training. We repeat this process by changing the number of hidden layers and record the classification accuracy. Finally, we choose the configuration that optimizes the accuracy and resource requirement trade-off.

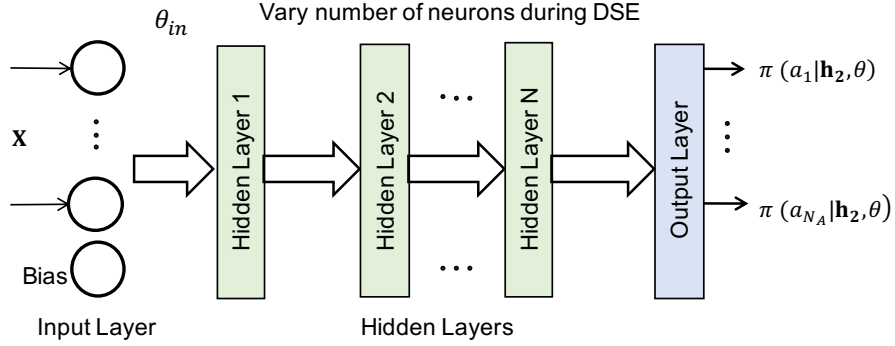


Figure 4.6: Design space exploration for neural network configuration

NN Classifier after DSE: At the end of the design space exploration, we choose the NN shown in Figure 4.7 as our classifier. The NN consists of an input layer, two hidden layers, and an output layer. The input layer processes the features denoted by \mathbf{X} and relays them to the first hidden layer with the ReLU activation. The second hidden layer then processes the output of the first layer and applies the ReLU activation. We denote the number of neurons in the hidden layers with N_{h1}, N_{h2} , respectively. These are chosen after the design space exploration. However, we use the variable form here to keep the description more general. The output layer in the proposed NN for HAR includes a neuron for each activity $a_i \in \mathbf{A} = \{J, L, S, St, W, SU, SD, T\}$, $1 \leq i \leq N_A$, where N_A is the number of activities in set \mathbf{A} , listed in Table 4.1. The output neuron of a given activity a_i computes $O_{a_i}(\mathbf{X}, \theta_{in}, \theta)$ as a function of the input features \mathbf{X} , and the neural network weights $\{\theta_{in}, \theta_{h1}, \theta\}$. To facilitate the policy gradient approach described in Section 4.6.1, we express the output O_{a_i} in terms of the output of the first hidden layer as:

$$O_{a_i}(\theta_{in}, \theta_{h1}, \theta) = O_{a_i}(\mathbf{h}_2, \theta) = \sum_{j=1}^{N_{h2}+1} h_{2,j} \theta_{j,i}, \quad 1 \leq i \leq N_A \quad (4.3)$$

where $h_{2,j}$ is the output of the j^{th} neuron in the first hidden layer, and $\theta_{j,i}$ is the weight from j^{th} neuron in the second hidden layer to output activity a_i . Note that $h_{2,j}$ is a function of \mathbf{X} ,

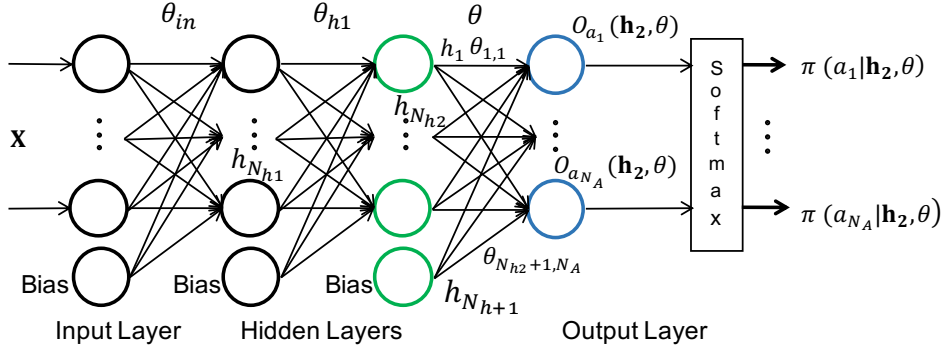


Figure 4.7: The NN used for activity classification and online learning.

θ_{in} , and θ_{h1} . The summation in Equation 4.3 goes to $N_{h2} + 1$, since there are N_{h2} neurons and one bias term in the hidden layer.

Once we calculate the value of each output neuron, we apply the softmax function to compute the probability of each activity as:

$$\pi(a_i|\mathbf{h}_2, \theta) = \frac{e^{O_{a_i}(\mathbf{h}_2, \theta)}}{\sum_{j=1}^{N_A} e^{O_{a_j}(\mathbf{h}_2, \theta)}}, \quad 1 \leq i \leq N_A \quad (4.4)$$

The output probabilities $\pi(a_i|\mathbf{h}_2, \theta)$ are expressed as a function of the second hidden layer outputs \mathbf{h}_2 instead of the input features, since our online learning algorithm will leverage it. Finally, the activity which has the maximum probability is chosen as the output.

Implementation cost: Our optimized classifier requires 264 multiplications for the FFT of stretch data, $121N_h + (N_{h1} + 1)(N_{h2} + 1) + (N_{h2} + 1)N_A$ multiplications for the NN and uses only 2 kB memory.

4.6 Online Learning for Human Activity Recognition

The trained NN classifier is implemented on the IoT device to recognize the human activities in real-time. In a real-world setting, the device will be used by users who are not part of the training process. These new users may have activity patterns that may not match the patterns

seen by the NN during the offline training process. Moreover, the activity patterns of the same user may change temporarily due to an injury. Therefore, there is a strong need to develop approaches that continuously updates the weights of the NN to adapt to changes in the user or user patterns.

We use two online learning algorithms for HAR that can be used depending on the level of feedback available from the users, since online learning depends critically on feedback from users. The user can give feedback upon completion of an activity, such as walking, which contains multiple segments (i.e., non-uniform action windows). When the user can indicate whether the inferred activity is correct or wrong, we use the policy gradient algorithm. In contrast, when the user can provide the actual label for the misclassified activities, we incrementally update the weights of the neural network using supervised learning. When no feedback is provided by the user, the weights of the network remain the same. In the following, we provide details on each of the above approaches.

4.6.1 Online Learning with Policy Gradient

We use the policy gradient approach to update the weights of the network when the user can only provide indirect feedback in terms of correct or incorrect. The policy gradient algorithm is suitable in this scenario since it can efficiently use the feedback to update the policy, i.e., the activity probabilities in Equation 4.4. We use the following definitions for the state, action, policy, and the reward.

State: Stretch sensor and accelerometer readings within a segment are used as the continuous state space. We process them as described in Section 4.4.3 to generate the input feature vector \mathbf{X} (Figure 4.7).

Policy: The NN processes input features as shown in Figure 4.7 to generate the hidden layer

outputs $\mathbf{h}_2 = \{h_j, 1 \leq j \leq N_h + 1\}$ and the activity probabilities $\pi(a_i|\mathbf{h}_2, \theta)$, i.e., the policy given in Equation 4.4.

Action: The activity performed in each sensor data segment is interpreted as the action in our RL framework. It is given by $\operatorname{argmax} \pi(a_i|\mathbf{h}, \theta)$, i.e., the activity with maximum probability.

Reward: Online training requires user feedback, which is defined as the reward function. When no feedback is provided by the user, the weights of the network remain the same. The user can give feedback upon completion of an activity, such as walking, which contains multiple segments (i.e., non-uniform action windows). If the classification in this period is correct, a positive reward (in our implementation +1) is given. Otherwise, the reward is negative (−1). We define the sequence of segments for which a reward is given as an *epoch*. The set of epochs in a given training session is called an *episode* following the RL terminology [156].

Objective: The value function for a state is defined as the total reward that can be earned, starting from that state and following the given policy until the end of an episode. Our objective is to maximize the total reward $J(\theta)$ as a function of the classifier weights.

Proposed Policy Gradient Update: In general, all the weights in the policy network can be updated after an epoch [156]. This is useful when we start with an untrained network with random weights. When a policy network is trained offline, as in our case, its first few layers generate broadly applicable intermediate features [100]. Consequently, we can update only the weights of the output layer to take advantage of offline training and minimize the computation cost. More precisely, we update the weights denoted by θ in Figure 4.7 to tune our optimized NN to individual users.

Since we use the value function as the objective, the gradient of $J(\theta)$ is proportional to the gradient of the policy [156]. Consequently, the update equation for θ is given as:

$$\theta_{t+1} \doteq \theta_t + \alpha r_t \frac{\nabla_{\theta} \pi(a_t|\mathbf{h}_2, \theta_t)}{\pi(a_t|\mathbf{h}_2, \theta_t)}, \quad \alpha : \text{Learning rate} \quad (4.5)$$

where θ_t and θ_{t+1} are the current and updated weight matrices, respectively. Similarly, a_t is the

current action at time t , r_t is the corresponding reward, and \mathbf{h}_2 denotes the hidden layer outputs. Hence, we need to compute the gradient of the policy to update the weights. To facilitate this computation and partial update, we partition the weights into two disjoint sets as \mathcal{S}_t and $\overline{\mathcal{S}}_t$. The weights that connect to the output O_{a_t} corresponding to the current action are in \mathcal{S}_t . The rest of the weights belong to the complementary set $\overline{\mathcal{S}}_t$. With this definition, we summarize the weight update rule in a theorem in order not to disrupt the flow of the chapter with derivations. Interested readers can go through the proof.

Weight Update Theorem: Given the current policy, reward, and the learning rate α , the weights in the output layer of the NN given in Figure 4.7 are updated online as follows:

$$\theta_{t+1,j,i} \doteq \begin{cases} \theta_{t,j,i} + \alpha r_t (1 - \pi(a_t | \mathbf{h}_2, \theta_t)) \cdot h_j & \theta_{t,j,i} \in \mathcal{S}_t \\ \theta_{t,j,i} - \alpha r_t \pi(a_i | \mathbf{h}_2, \theta_t) \cdot h_j & \theta_{t,j,i} \in \overline{\mathcal{S}}_t \end{cases} \quad (4.6)$$

Proof: The partial derivative of the policy $\pi(a_t | \mathbf{h}_2, \theta)$ with respect to the weights $\theta_{j,i}$ can be expressed using the chain rule as:

$$\frac{\partial \pi(a_t | \mathbf{h}_2, \theta)}{\partial \theta_{j,i}} = \frac{\partial \pi(a_t | \mathbf{h}_2, \theta)}{\partial O_{a_i}(\mathbf{h}_2, \theta)} \frac{\partial O_{a_i}(\mathbf{h}_2, \theta)}{\partial \theta_{j,i}} \quad (4.7)$$

where $1 \leq j \leq N_h + 1$ and $1 \leq i \leq N_A$. When $\theta_{t,j,i} \in \mathcal{S}_t$, action a_t corresponds to output $O_{a_t}(\mathbf{h}_2, \theta)$. Hence, we can express the first partial derivative using Equation 4.4 as follows:

$$\begin{aligned} \frac{\partial \pi(a_t | \mathbf{h}_2, \theta)}{\partial O_{a_t}(\mathbf{h}_2, \theta)} &= \frac{e^{O_{a_t}(\mathbf{h}_2, \theta)}}{\sum_{j=1}^{N_a} e^{O_{a_j}(\mathbf{h}_2, \theta)}} - \frac{(e^{O_{a_t}(\mathbf{h}_2, \theta)})^2}{\left(\sum_{j=1}^{N_a} e^{O_{a_j}(\mathbf{h}_2, \theta)}\right)^2} \\ &= \pi(a_t | \mathbf{h}_2, \theta) (1 - \pi(a_t | \mathbf{h}_2, \theta)) \end{aligned} \quad (4.8)$$

Otherwise, i.e., $\theta_{t,j,i} \in \overline{\mathcal{S}}_t$, the derivative is taken with respect to another output. Hence, we can find the partial derivative as:

$$\frac{\partial \pi(a_t | \mathbf{h}_2, \theta)}{\partial O_{a_i}(\mathbf{h}_2, \theta)} = -\frac{e^{O_{a_t}(\mathbf{h}_2, \theta)} e^{O_{a_i}(\mathbf{h}_2, \theta)}}{\left(\sum_{j=1}^{N_A} e^{O_{a_j}(\mathbf{h}_2, \theta)}\right)^2} = -\pi(a_t | \mathbf{h}_2, \theta) \pi(a_i | \mathbf{h}_2, \theta) \quad (4.9)$$

The second partial derivative in Equation 4.7, $\partial O_{a_i}(\mathbf{h}_2, \theta) / \partial \theta_{j,i}$, can be easily computed as h_j using Equation 4.3. The weight update is the product of learning rate α , reward r_t , h_j , and the partial derivative of the policy with respect to the output functions. For the weights $\theta_{t,j,i} \in \mathcal{S}_t$, we use the partial derivative in Equation 4.8. For the remaining weights, we use Equation 4.9. Hence, we obtain the first and second lines in Equation 4.6, respectively. **Q.E.D** \square

4.6.2 Online Updates with Incremental Supervised Learning

Online learning using the policy gradient algorithm is most useful when the activity labels are not available. While it can be used when activity labels are available, the convergence rates of the policy gradient algorithm are lower when compared to supervised learning. Therefore, we use supervised learning for performing incremental weight updates when the user can provide activity labels at runtime, as outlined in Algorithm 2. The algorithm takes the offline trained weights of the NN as its input. Then, the first step in the algorithm is to initialize a buffer B of size M that is used to store the training data for weight updates. With this initialization, we start the online phase of the algorithm. For each activity segment t , we first obtain the feature vector \mathbf{X}_t and use it with the weights to determine the activity probabilities (lines 4–5). Then, we assign the activity with the maximum probability as the output activity. This activity is shown to the user who then provides the actual activity label a_t^* to the algorithm. If the actual activity label a_t^* does not match the activity output of the NN, we store both the feature vector \mathbf{X}_t and the label a_t^* in the buffer B (lines 7–10). Otherwise, we proceed to the next activity. This process continues until the buffer is full. Once the buffer is full, we use the training data in the buffer to update the weights of the NN using the backpropagation algorithm. Similar to the policy gradient approach, we update only the weights in the output layer. Finally, we reset the data in buffer B so that training data from the updated network can be collected.

Algorithm 2: Weight Update via Supervised Learning

```
1 Input: Offline trained NN weights  $\{\theta_{in}, \theta_{h2}, \theta\}$ 
2 Initialize the buffer  $B$  of size  $M$  to store new training examples
3 for each activity segment  $t$  do
4   | Generate feature vector  $\mathbf{X}_t$ 
5   | Evaluate activity probabilities  $\pi(a_t | \mathbf{h}_2, \theta_t)$  using Equation 4.4
6   |  $a_t \leftarrow \operatorname{argmax} \pi(a_i | \mathbf{h}, \theta)$ 
7   | Obtain activity label  $a_t^*$  from user
8   | if  $a_t \neq a_t^*$  then
9     | Append  $\{\mathbf{X}_t, a_t^*\}$  to  $B$ 
10  | end
11  | if  $B$  is full then
12    | Obtain  $\theta_{t+1}$  using backpropagation algorithm on current weights  $\theta_t$  and  $B$ 
13    | Reset the data in  $B$ 
14  | end
15 end
```

In summary, the weights of the output layer are updated online using either of the two algorithms after user feedback. We note that the weights of the hidden layers can be updated similarly by computing the gradient of the hidden layers with respect to their weights. Detailed results for the improvement in accuracy using the online learning approaches and a comparison among them are presented in Section 4.7.4.

4.7 Experimental Evaluation and Discussions

4.7.1 Experimental Setup

We implement the proposed HAR framework on the TI-CC2650 [160] IoT device. We place the TI-CC2650 device on the ankle while the flexible stretch sensor is worn on the knee. The stretch sensor transmits the data to the TI-CC2650 [160] IoT device that processes the data to preforms the activity recognition. The recognized is then transmitted to a host device, such

as a smartphone. We transmit only the activity classification since transmission of the raw data incurs a higher communication overhead.

Training, cross-validation, and test data split: We first divide the users into two sets, *offline training*, and *online training*. The offline training set includes 18 users while the online training set includes the remaining 4 users. The users in the offline training set are used to train the neural network classifier. Within this dataset, we reserve 60% data for training, 20% data for cross-validation, and 20% data for test. The trained classifier is then used to perform activity classification for the online training users and update the weights of the network. Furthermore, in order to ensure robustness of the proposed framework, we create a total of 30 combinations of *offline training* and *online training* sets. In each combination we ensure that number of common users is minimal.

4.7.2 Neural Network Design Space Exploration

We use a neural network to perform online activity recognition and training. The NN has to be implemented on the wearable device with limited memory (in our case 20 kB). Therefore, it should have a small memory footprint, i.e., a lower number of weights, while giving a high recognition accuracy. To choose the neural network structure for HAR, we perform a design space exploration with a single hidden layer and two hidden layer networks. In each of the networks, we vary the number of neurons in the hidden layers to study the effect on accuracy and memory requirements. Figure 4.8(a) shows the change in accuracy as we increase the number of neurons in the hidden layer for a NN with a single hidden layer. We see that the accuracy of the network saturates at around 93% after 4 neurons in the hidden layer. The memory requirement M for this network is given by $M = 121 * N_h + N_h * 8$, where N_h is the number of neurons in the hidden layer. The equation shows that the addition of a single

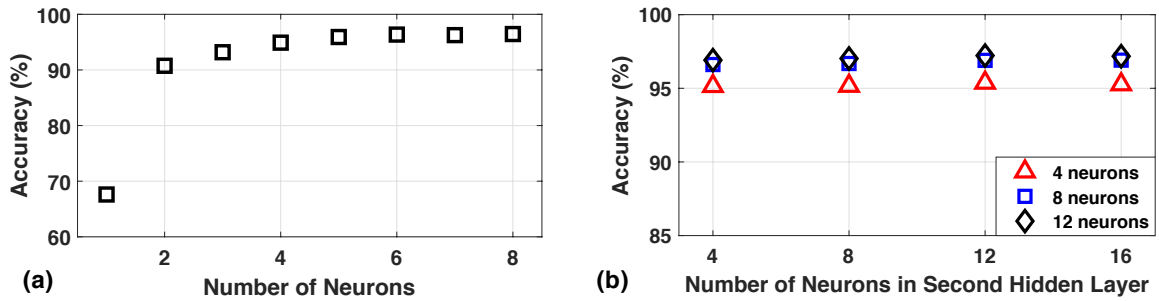


Figure 4.8: Comparison of accuracy with the number of neurons

neuron in the hidden layer leads to 120 additional weights. Therefore, addition of neurons to the NN incurs a high memory cost while providing marginal improvements in accuracy.

Next, we analyze the accuracy obtained by NNs with two hidden layers. We first fix the number of neurons in the first layer and then vary the number of neurons in the second layer. We repeat this for different number of neurons in the first layer. Figure 4.8 shows the accuracy of the NN when we fix the first layer neurons to 4, 8, and 12, respectively. The three types of markers represent the neurons in the first layer, while the x-axis shows the number of neurons in the second layer when the first layer is fixed. All the networks achieve similar accuracy with 4-neuron networks having a slightly lower accuracy. At the same time, the 4-neuron networks come with a significant memory advantage. When we compare the networks with one and two hidden layers, the two-layer networks achieve better accuracy with a slightly higher memory cost. This is because the memory cost additional of neurons in the second layer is much lower than in the first layer. Therefore, in our implementation, we choose a NN with 4 neurons in the first layer and 8 neurons in the second layer. We choose this over a network with just 4 neurons in the second layer as it provides a more robust operation. The chosen network provides a 95.32% accuracy with a 2 kB memory requirement.

4.7.3 Accuracy Analysis of the Neural Network

4.7.3.1 Confusion Matrix

Once we finalize the structure of the NN classifier, we move on to train the NN using the data from 18 users reserved for offline training. Then, we analyze the accuracy of each activity in the training set by obtaining the confusion matrix, as shown in Table 4.5. There is one column and one row corresponding to the activities of interest. The numbers on the diagonal show the recognition accuracy for each activity. For example, the first row in the first column shows that jump is recognized with 94.6% accuracy. We also include the total number of activity windows with the corresponding label at the beginning of each row to provide the absolute numbers. For instance, a total of 392 windows were labeled “Jump” for the 18 users chosen for training.

From the confusion matrix, we see that the NN achieves accuracy greater than 94% for all activities except transition. The accuracy is lower for transitions as each transition window includes features from two distinct activities. The loss in accuracy is acceptable for transitions, as we can indirectly infer by looking at the segments before and after the transition. Finally, we

Table 4.5: Confusion matrix for 18 training users

	Jump	Lie Down	Sit	Stand	Walk	Stairs Up	Stairs Down	Transition
Jump	94.6%	0.00	0.26%	0.00	4.08%	0.00	0.26%	0.77%
Lie Down	0.00	99.8%	0.21%	0.00	0.00	0.00	0.00	0.00
Sit	0.00	0.00	96.4%	2.98%	0.00	0.00	0.00	0.63%
Stand	0.00	0.00	2.47%	95.1%	0.00	0.00	0.00	2.47%
Walk	0.75%	0.00	0.25%	1.23%	94.5%	0.19%	0.69%	2.38%
Stairs Up	0.00	0.00	0.00	0.00	0.00	96.3%	3.66%	0.00
Stairs Down	0.00	0.00	0.00	0.00	3.80%	0.00	96.2%	0.00
Transition	1.23%	0.00	2.47%	2.88%	2.88%	0.00	0.00	90.5%

Table 4.6: Comparison of accuracy for different classifiers

Classifier	Train Acc. (%)	Test Acc. (%)	Overall Acc. (%)
Random Forest	100.00	95.60	99.12
C4.5	98.67	91.70	97.28
k-NN	96.76	94.39	96.29
SVM	99.01	93.53	97.91
Our NN	96.24	91.71	95.32

note that all 30 combinations of training user sets achieve similar confusion matrices. Therefore, we do not report the confusion matrix for each combination.

4.7.3.2 Comparison with Other Classifiers

One to one comparison with existing approaches for HAR is not feasible since they use devices, datasets, and activities that are different from our study. Therefore, we implement commonly used supervised learning classifiers on our dataset and compare the accuracies. Table 4.6 shows the accuracy of commonly used classifiers. We see that the proposed neural network classifier achieves a competitive accuracy when compared to the other classifiers. While the other classifiers achieve a slightly higher accuracy than the NN, they are not amenable to online learning. In contrast, the proposed NN can be efficiently updated at runtime to enable online learning for HAR, which is one of the focus areas of this work.

4.7.3.3 Robustness of the NN Classifier

We need to ensure that the proposed NN classifier is robust to input from different users. Therefore, we perform an accuracy analysis with the 30 user combinations described in 4.7.1. We first train with 60% of the data from the 18 users present in the *offline training* set of

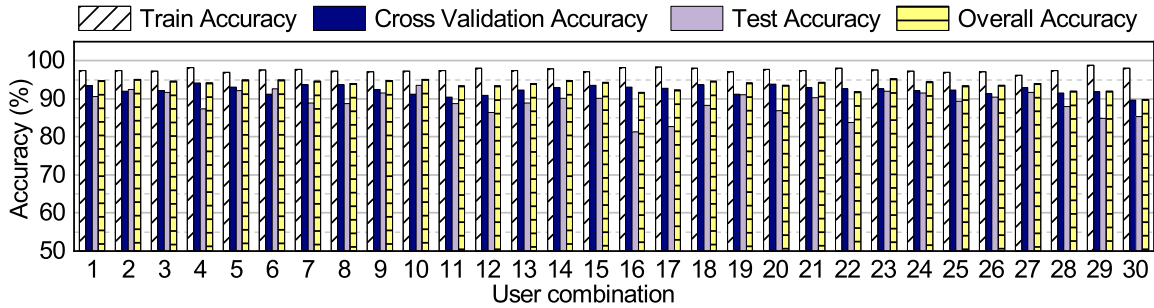


Figure 4.9: Comparison of accuracy with different combinations of users for training

each user combination. The training phases also uses 20% of the data in each set for cross-validation during training. After training a NN with each user combination, we test them with the remaining 20% data and 4 unseen users (*online training set*). The accuracy results for the 30 classifiers obtained in this analysis are summarized in Figure 4.9. We see that all the user combinations achieve training accuracy higher than 95% and cross-validation accuracy greater than 90%. Moreover, most of the combinations achieve test accuracy higher than 90%. A few of the user combinations, such as 16, 17, and 30, have lower test accuracy. This can be attributed to the fact that the activity signatures observed in the training set of 18 users do not capture the activity signatures of the 4 test users. We can overcome this issue by either including the users in the training set or applying online learning.

4.7.4 Online Learning with new users

We use the NN classifiers trained for each of the 30 user combinations to recognize the activities of the users not included in the training set. As we see in the previous section, the test accuracies for some of the user combinations are lower than 90%. Therefore, we use the proposed online learning algorithms to adapt the weights of the classifiers to the new users.

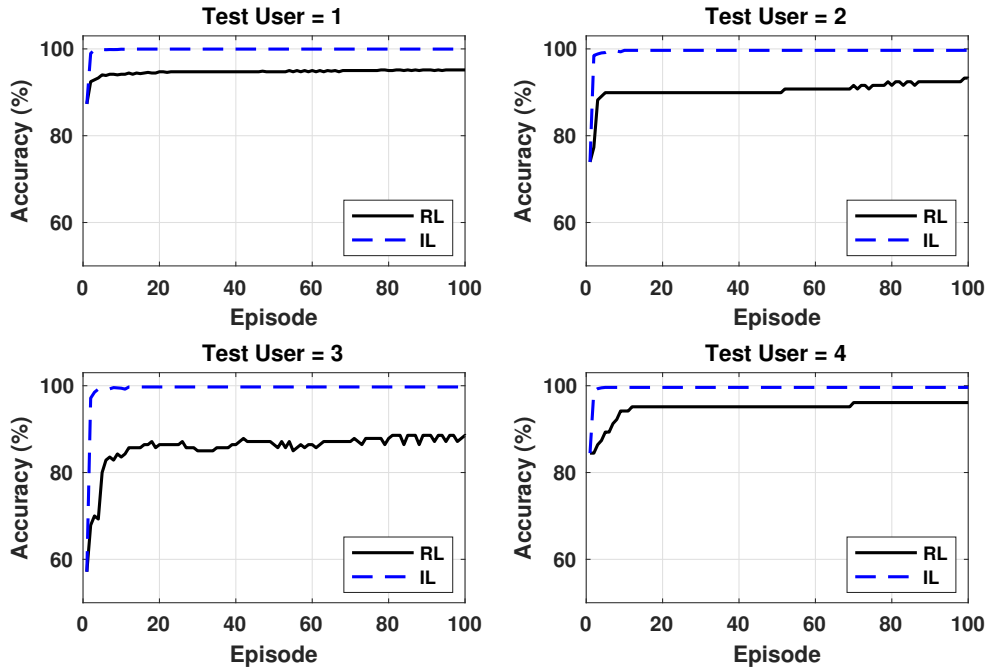


Figure 4.10: Comparison of reinforcement learning and incremental learning

Figure 4.10 shows the improvement in accuracy obtained by the reinforcement learning and incremental supervised learning for four users not seen during training. The initial NN for these users is obtained from user combination 25 that has greater than 95% training accuracy. However, we see that the initial accuracy for all the four users is lower than the accuracy for training users. In particular, the accuracy for user 10 is only about 60%. Starting with this initial accuracy, we apply the online learning algorithms. Each episode in the x-axis corresponds to an iteration of RL using the data set for new users. The weights of the NN are updated after each segment as a function of the user feedback for a total of 100 episodes. We note that the data of a particular user are reused in each episode of online learning. As more data becomes available for each user, we can use different subsets of data in each episode. We see that the online learning algorithms provide a consistent improvement in accuracy for all the new users. For instance, the policy gradient algorithm improves the accuracy of user 19 from about 83% to 95%, while the incremental supervised learning approach improves the accuracy to almost

100%. Similar improvements are observed for other users as well. We also see that the incremental supervised learning approach improves the accuracy much faster and to a higher level when compared to the policy gradient approach. This is expected since it uses the actual activity labels instead of an indirect reward. Therefore, it is more beneficial to use the incremental supervised learning algorithm when the user provides the actual activity labels. We note that the user feedback can be obtained periodically from the user by prompting them to enter the information on a smartphone app. The frequency of the feedback can be reduced as the classifier adapts to the user.

Table 4.7 provides a summary of the improvement obtained through online learning for all 30 user combinations. The table contains a set of three columns for each user in the *online training* set. Within each set of columns, we show the initial accuracy at the beginning of online training, the final accuracy obtained with reinforcement learning, and the final accuracy obtained with incremental learning. We see that the incremental learning approach achieves accuracy greater than 99% for all user combinations. We can attribute this to the fact that the neural network adapts to the patterns of the user, thus providing personalized HAR. Reinforcement learning also achieves accuracy greater than 90% for most of the user combinations and test users. For a few of the users, such as test user 1 in combination 16, the accuracy improvement with reinforcement learning saturates. This limitation can either be resolved using incremental learning or updating all the layers in the network. In summary, the online learning algorithms proposed in this chapter improve the accuracy for users not previously seen by the network. This ensures that the device can adapt to new users very easily and provide personalized HAR.

Table 4.7: Comparison of accuracy improvement with online learning

User comb.	Test User 1 Accuracy (%)			Test User 2 Accuracy (%)			Test User 3 Accuracy (%)			Test User 4 Accuracy (%)		
	Init.	RL	IL	Init.	RL	IL	Init.	RL	IL	Init.	RL	IL
1	93.7	97.5	99.7	93.2	98.9	99.9	88.2	95.1	99.7	75.0	82.1	99.7
2	91.1	92.4	99.6	83.3	86.1	99.2	82.4	94.1	98.2	91.1	95.8	99.8
3	93.7	97.5	99.5	92.2	98.4	99.9	76.1	87.7	99.9	88.4	95.3	99.9
4	91.1	100.0	99.9	86.9	94.6	100.0	58.8	85.3	99.6	80.6	94.2	99.6
5	97.5	97.5	99.7	96.2	97.5	99.9	82.5	96.0	99.8	84.4	91.1	99.1
6	86.4	92.0	99.7	97.2	97.2	99.7	97.1	98.8	100.0	86.3	94.3	99.9
7	94.3	98.9	99.5	88.2	94.1	98.8	91.4	94.5	99.8	60.8	89.2	99.8
8	92.0	93.2	99.8	92.2	95.3	99.8	79.3	95.3	99.9	87.4	96.1	99.7
9	89.6	94.4	99.7	94.4	97.2	99.4	93.3	96.9	99.8	91.3	97.1	99.8
10	93.1	97.9	99.8	88.2	94.1	98.8	98.4	99.2	99.8	96.8	98.7	100.0
11	66.0	93.8	99.9	80.2	93.2	99.8	90.1	95.9	99.9	87.3	96.8	99.7
12	84.7	92.4	99.8	79.7	94.4	100.0	54.9	65.7	99.6	82.2	88.9	99.1
13	69.3	84.3	99.7	88.9	91.7	98.9	80.2	96.1	99.9	93.6	99.4	99.9
14	56.4	82.9	99.7	76.5	94.1	97.6	89.2	94.9	99.9	84.4	91.1	99.3
15	81.4	87.1	99.8	93.2	96.4	99.8	98.4	99.6	99.8	67.6	91.2	99.6
16	48.4	66.8	99.9	55.7	81.4	99.9	92.6	95.1	99.8	88.9	95.2	99.8
17	43.0	67.7	99.9	96.3	98.4	99.9	86.4	94.2	99.7	88.9	91.1	99.1
18	92.4	94.1	99.7	86.1	91.7	98.9	52.9	72.5	99.9	91.3	91.3	99.7
19	89.1	92.4	99.7	87.0	93.8	99.9	85.9	91.4	99.8	96.2	99.4	99.9
20	95.0	95.8	99.7	64.1	80.1	99.9	82.4	88.2	98.2	87.9	94.8	99.8
21	88.6	98.1	99.9	88.2	94.1	98.8	81.6	94.2	99.6	85.7	92.1	99.8
22	94.4	99.3	100.0	42.7	59.6	99.9	89.8	96.6	99.7	96.8	99.4	99.9
23	89.1	97.8	99.8	92.4	95.0	99.7	92.0	94.3	99.7	82.2	91.1	99.3
24	78.3	100.0	100.0	90.1	99.0	99.9	92.0	94.5	99.8	90.4	94.9	99.9
25	87.3	95.2	100.0	73.9	92.4	99.7	57.1	87.1	99.7	84.5	96.1	99.6
26	91.9	95.4	99.9	85.5	94.7	99.9	88.9	97.2	98.9	88.9	97.8	99.6
27	92.0	96.4	100.0	89.1	100.0	100.0	99.2	99.2	99.9	92.9	96.0	99.7
28	83.7	88.5	100.0	86.7	98.8	99.9	87.4	94.1	99.7	95.9	97.9	99.9
29	68.3	85.9	100.0	76.1	100.0	99.8	55.9	73.5	99.7	63.1	93.0	99.9
30	81.1	90.3	100.0	72.2	95.0	99.9	54.3	94.4	99.8	66.9	95.5	99.9

4.7.5 Power, Performance and Energy Evaluation

To fully assess the cost of the proposed HAR framework, we present a detailed breakdown of execution time, power consumption, and energy consumption for each step. The first part of HAR involves data acquisition from the sensors and segmentation. Since the segmentation

Table 4.8: Execution time, power and energy consumption

	Block	Exe. Time (ms)	Average Power (mW)	Energy (μ J)
Sense	Read/Segment	1500.00	1.13	1695.00
Compute	DWT	7.90	9.50	75.05
	FFT	17.20	11.80	202.96
	NN	2.50	12.90	32.25
	Overall	27.60	11.24	310.26
Comm.	BLE	8.60	5.00	43.00

algorithm runs continuously while the data is acquired, its energy consumption is included in the sensing block. Table 4.8 shows the power and energy consumption for a typical segment of 1.5 s. The average power consumption for data acquisition is 1.13 mW, leading to a total energy consumption of 1695 μ J. If the segments are of a longer duration, the energy consumption for data sensing increases linearly. Following the data segmentation, we extract the features and run the classifier. The execution time, power, and energy for these blocks are shown in the "Compute" rows in Table 4.8. As expected, the FFT block has the largest execution time and energy consumption. However, it is still two orders of magnitude lower than the duration of a typical segment. Finally, the energy consumption of the BLE communication block is given in the last row of Table 4.8. Since we transmit the inferred activity, the energy consumed by the BLE communication is only about 43 μ J. With less than 12.5 mW average power consumption, our approach enables close to 60-hour uninterrupted operation using a 200 mAh @ 3.7 V battery [51]. Hence, it can enable self-powered wearable devices [23] that can harvest their own energy [125].

4.8 Summary

Human activity recognition has wide-ranging applications from movement disorders to patient rehabilitation to activity promotion in the general population. Successful research in HAR critically depends on the availability of open-source datasets. To address this need, we presented a w-HAR, an open-source dataset for human activity recognition that, *for the first time*, includes data from both wearable stretch and accelerometer sensors. We provide three versions of the sensor data as part of w-HAR. The first version provides raw sensor data that allows researchers to develop their own algorithms for all steps of HAR. Secondly, we provide segmented data that gives researchers the freedom to just focus on feature generation and classification. Finally, we provide a baseline feature set for researchers who want to focus only on classifier development.

We also presented a HAR framework on a wearable IoT device using stretch and accelerometer sensors. The first step of our solution is a novel technique to segment the sensor data non-uniformly as a function of the user motion. Then, we generate FFT and DWT features using the segmented data. The feature data was used to perform a DSE of neural network classifiers for HAR. After the DSE, we obtained a NN classifier that achieved 95% accuracy for user data available at design time. Then, we introduced two online learning algorithms to continuously improve the classifier weights for new user subjects as a function of user feedback. The online learning algorithms improved the accuracy for new users by as much as 40% with less than 12.5 mW power consumption.

TRANSFER LEARNING FOR HUMAN ACTIVITY RECOGNITION USING REPRESENTATIONAL ANALYSIS OF NEURAL NETWORKS

5.1 Introduction

Human activity recognition (HAR) is a critical component in a range of health and activity monitoring applications [30, 74, 89, 107]. It provides valuable insight into movement disorders by allowing health professionals to monitor their patients in a free-living environment [44, 53]. HAR is also the first step towards understanding gait parameters, such as step length and gait velocity, which are also used in movement disorder analysis and rehabilitation [147, 175]. In addition, HAR is used for obesity management and promoting physical activity among the public. Due to these high-impact applications of HAR, it has received increased research attention in recent years [153, 169].

Most HAR techniques start with collecting sensor data from users available at design time [96]. This data is used to train a classifier for the activities of interest. Then, the trained classifier is used by new users in the field, whose data is not available for training. This approach assumes that the HAR classifiers can be transferred across different user sets. However, this assumption may not hold in general as activity patterns can change with age, gender, and physical condition. For instance, Figure 5.1 shows the stretch sensor data during walking for four users in our experimental dataset. There is a significant variation both in the range of sensor values and the data patterns. Furthermore, the activity patterns may vary over time, even for a given user, due to the progression of symptoms, injury, or other physical changes. These variations can significantly reduce the recognition accuracy for new users as we demonstrate

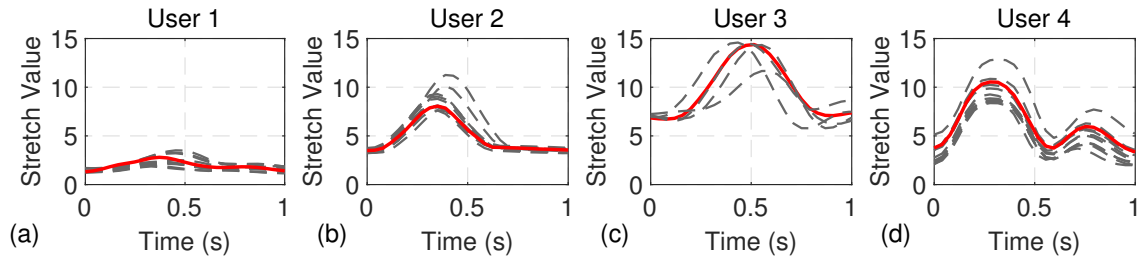


Figure 5.1: Comparison of stretch sensor [117] data of four users for a single step during walk. There is a significant change in both the range of values and data patterns. The grey dashed lines show different instances of the same activity, while the red line shows a *representative* activity window for each user.

in this chapter. Therefore, *classifiers that are designed offline must adapt to changing data patterns of new and existing users to achieve high classification accuracy.*

Training classifiers from scratch for new users is expensive due to high data storage and computational requirements. It is especially challenging for low-power wearable devices and smartphones, which are the most commonly used devices for HAR [96, 153]. Moreover, classifiers trained offline have better generalization capability due to a large amount of data [113]. Although training from scratch for a particular user may increase its performance, it may reduce robustness due to overfitting. In contrast, transfer learning with a common feature set can carry over the generalization capabilities from the offline stage and fine-tune for user-specific features in the field.

This chapter first demonstrates that HAR classifiers designed offline cannot be transferred as a whole to an arbitrary set of users. Then, it presents a systematic study to determine how to transfer the offline knowledge and adapt classifiers to individual users. This study is performed using convolutional neural networks (CNNs) due to their ability to produce broadly applicable features from raw input data. Specifically, we use canonical correlation analysis (CCA) [113] to evaluate the distance between the layers of CNNs trained with different sets of users. This analysis reveals that the initial layers of CNNs provide representations that are general across all the users, while the deeper layers discriminate between the features specific to each set of users.

Based on this insight, the proposed approach achieves high accuracy and significant savings in training time by fine-tuning the deeper layers that differentiate users while transferring the weights of the initial layers.

The proposed approach is validated using an in-house wearable HAR dataset (w-HAR), which is released to the public, and two public datasets [6, 139]. We start by dividing the users in each dataset into multiple clusters so that the effect of transfer learning for unseen user clusters can be evaluated. In a challenging scenario, the clusters have to be as separated as possible such that the classifier can only learn the patterns from the users in the training set. For instance, the four users in Figure 5.1 belong to different clusters. To this end, we generate user clusters both randomly (for average-case) and using k-means clustering [58] (for the most challenging scenario). Next, a CNN classifier is trained for each user cluster. After analyzing the similarity between the CNNs trained with different user clusters, the weights are transferred between user clusters and the dissimilar layers are fine-tuned. Extensive evaluations for the w-HAR dataset show that in the worst case, transferring weights and fine-tuning the last layer achieves accuracy that is the same as the accuracy obtained by training from scratch while reducing the computation during training by 66%.

In summary, the major contributions of this chapter are as follows:

- An empirical demonstration which shows that HAR classifiers designed offline cannot be transferred to an arbitrary set of users,
- A systematic analytical study which reveals that deeper layers of HAR classifiers capture user-specific information, while the initial layers provide general features,
- Extensive experimental evaluations using three datasets that show up to 43% and on average 14% higher accuracy compared to the accuracy without using transfer learning for new users.

5.2 Related Research

Transfer learning: Transfer learning aims to leverage the information learned in one domain to improve the accuracy in a new domain [121]. The information used for transfer includes the weights of a classifier [55, 56, 148], features [138], and instances of data [43]. The transfer can occur between different applications or between different scenarios of a single application [121]. A popular example of transfer learning between applications is medical imaging [144, 151], where CNNs trained for classifying ImageNet [47] are adapted to classify medical images. Similarly, transfer learning for new scenarios includes adaptation to a new device [2], classes [27], or users [68].

One of the fundamental aspects of transfer learning is identifying what information to transfer. To this end, research has focused on parameter transfer [119], feature representation transfer [7, 28], and data instance transfer [43]. We focus our attention on parameter transfer since the proposed approach uses parameter transfer as well. Oquab et al. [119] design a method to reuse layers trained with one dataset to compute mid-level image representation for images in another dataset. Yosinski et al. [176] empirically quantify the generality versus specificity of neurons in each layer of a deep convolutional neural network for the ImageNet [47] dataset. They show that the features in the initial layers are general in that they are applicable to multiple image recognition tasks. Morcos et al. [113] directly analyze the hidden representations of each layer in CNNs by using Canonical Correlation Analysis (CCA). CCA enables a comparison of the learned representations between different neural network layers and architectures.

Transfer learning has been applied successfully in fields such as medical imaging classification [144] and computer vision [134]. Salem et al. [144] presented an approach to transfer a CNN from image classification domain to electrocardiogram (ECG) signal classification domain. Similarly, the authors in [136] explore properties of transfer learning from natural image

classification networks to medical image classification. Quattoni et al. [134] show that prior knowledge from unlabeled data is useful in learning a new visual category from few examples [134]. The authors developed a visual-category learning algorithm called sparse prototype learning that can learn an efficient representation from a set of related tasks while taking advantage of unlabeled data.

Transfer learning for HAR: Research on HAR has increased in recent years due to its potential in applications such as movement disorders, obesity management, and remote patient monitoring [30, 74, 107]. One of the challenges in HAR algorithms is that the data available at design time may not be representative of the activity patterns of new users. As a result, accuracy can degrade for new users [50]. Recent research has used transfer learning to address this issue [40, 50, 101, 141]. The survey by Cook et al. [40] presents how different types of transfer learning have been used for HAR. Here we present the transfer learning approaches that are most relevant to this chapter while referring readers to the survey for a broader analysis. Dine et al. [50] perform an empirical study to analyze the performance of transfer learning methods for HAR and find that the maximum mean discrepancy method is most suitable for HAR. A CNN-based method to transfer learned knowledge to new users and sensor placements is presented in [36]. The authors empirically determine the number of layers to transfer based on the accuracy obtained after transferring. However, this method is not scalable since the training has to be repeated for each configuration of the transfer. Rokni et al. [141] use transfer learning to personalize a CNN classifier to each user by retraining the classification layer with new users. However, the authors do not provide any insight into the number of layers that can be transferred between users and how it benefits the learning for new users. In contrast, we perform representational analysis using CCA to determine layers that need to be fine-tuned for new users.

Our work proposes a complete transfer learning framework for HAR using representational

analysis of CNNs. We start with clustering the users such that they are as separated as possible. This clustering allows us to test the *robustness* of our approach. Then, we analyze the hidden representation of different layers of a CNN by using CCA similarity. Using insights from the CCA similarity analysis, we fine-tune specific layers of the network to adapt to new users. The proposed approach is evaluated on w-HAR and two public datasets [6, 139] with both manually and randomly generated clusters.

5.3 Human Activity Recognition Framework

We use the human activity recognition application described in Chapter 4. A brief summary is provided here to maintain the flow of the chapter. Most HAR approaches start with data from wearable inertial sensors or a smartphone to record the data when the user is performing the activities of interest.⁸ After collecting the sensor data, the next step is to pre-process and segment the sensor data for feature generation. The most common approach in literature for segmentation is to divide the data into one to ten-second windows with a 50% overlap between consecutive windows [6, 93, 168]. Then, the data within each window is processed to generate the features for the classifier. A variety of classifiers, such as neural networks, decision trees, random forest, and k-nearest neighbors, have been used for HAR. This chapter focuses on CNNs since the convolutional layers produce broadly applicable features, while the remaining layers facilitate efficient online weight updates.

⁸Video cameras are also used for HAR, but we focus on HAR using wearable sensors and smartphones.

5.3.1 Experimental Datasets

Wearable HAR dataset (w-HAR): We use the w-HAR dataset described in Chapter 4 as one of the experimental datasets in this chapter. The w-HAR dataset includes empirical data from 22 users while performing seven activities and transitions between them.

UCI HAR dataset [6]: The UCI HAR consists of data from 30 users who performed six activities (*lie down, sit, stand, stairs down, stairs up, and walk*) while wearing a smartphone on the waist. The dataset records readings from the accelerometer and gyroscope sensors in the smartphone. The dataset provides 561 time and frequency domain features for each activity window of 2.56 s. We use the default feature set provided by the dataset in our analysis.

UCI HAPT dataset [139]: This is an updated version of the UCI HAR dataset where the authors added information about postural transitions, such as stand-to-sit and sit-to-stand. Similar to the UCI HAR dataset, the HAPT dataset provides 561 time and frequency domain features for each activity.

5.4 Transfer Learning for Human Activity Recognition

5.4.1 Flow of the Proposed Transfer Learning Approach

Figure 5.2 shows an overview of the proposed transfer learning framework. First, the feature data is split into multiple clusters by using k-means clustering. The clustering ensures that users having similar data patterns, such as walking speed, are grouped together. More importantly, it ensures that users across different clusters are more dissimilar than random partitioning. Next, a CNN classifier is trained with each user cluster (UC) obtained in the previous step. To avoid overfitting, we divide each user cluster into 60% train, 20% cross-

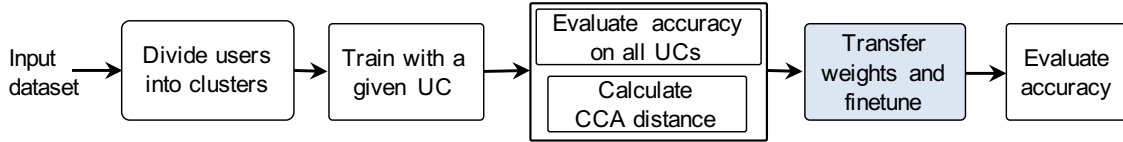


Figure 5.2: Overview of the transfer learning approach for HAR

validation, and 20% test data. The test accuracy obtained is the baseline accuracy when the respective clusters are used for training the CNN. In the next step, a CNN trained with one UC is tested with data of other UCs that are not used in training the CNN. 100% of the other UC data are used for testing since none of it is used during training. For example, a CNN trained with the training data of UC 1 is tested with all the data of UC 2. The accuracy obtained in this step is referred to as the *cross-UC accuracy*. Along with the accuracy evaluation, we also calculate the CCA distance between trained CNNs in this step. The CCA distance helps in analyzing layers that generalize for all the UCs and the layers that differentiate UCs. Based on the CCA distance analysis, the next step is transferring the CNN weights between UCs and fine-tuning the layers that provide distinguishable information for each UC. This step is performed for both k-means clustering and multiple randomly generated clusters to test the robustness of the approach. Finally, we evaluate the accuracy after completing the fine-tuning process.

5.4.2 Clustering Users with Distinct Activity Patterns

The first step in the proposed framework is partitioning the users into clusters. Clustering ensures that users in separate clusters have as distinct activity patterns as possible, as illustrated in Figure 5.1. Hence, we can analyze the benefits and efficiency of transfer learning under more challenging conditions than random partitioning. We employ the following steps to obtain the user clusters.

Representative Window for Each User: Each user has multiple windows for the same activity since the collected data is segmented before feature generation. For example, one-minute long walking data is divided into 60 activity windows, assuming a one-second segment duration. Furthermore, activity data may be coming from different experiments with small differences in sensor locations. As a result, there are variations in the features across different windows, even for a given user and activity. To bypass the variations across windows and facilitate clustering, we identify a *representative window* for each activity of each user. For example, consider our first dataset with 22 users and 8 activities. To obtain the representative windows, we first extract all the activity windows for a given user and activity (e.g., all windows labeled as “walk” for user-1). Then, we compute the mean Euclidean distance from each window to all other windows for this user-activity pair. A large distance means that the corresponding window is more likely to be an outlier. In contrast, the window with the smallest mean distance to the rest of the windows is marked as the *representative window* for the corresponding user-activity pair, as illustrated with red lines in Figure 5.1.

k-means Clustering: The previous process results in one representative window for each activity for each user. That is, each of the 22 users has 8 representative windows (one for each activity) in our dataset with 22 users and 8 activities. Next, we compute the mean correlation distance [165] between the representative windows across different users for each activity. A small distance means that the activity pattern of the user is similar to other users. Conversely, a larger distance implies that the user’s activity pattern is separated from the other users. The distance to other users for each activity is stored as a multidimensional vector whose length is equal to the number of activities in the dataset. Finally, the distance vectors are used with the k-means algorithm [58] to generate user clusters that are as separated as possible. Based on our empirical observations of the data, we choose four clusters each for the three datasets we use in this chapter.

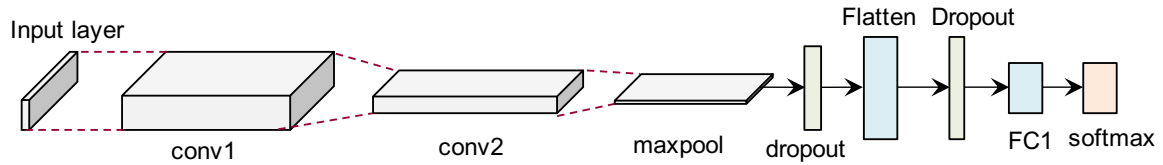


Figure 5.3: The architecture of CNN. The layers annotated at the bottom are used in CCA distance.

5.4.3 Baseline Classifier Training for each User Cluster

CNN for HAR: We design a CNN for each input datasets to recognize the activities in the respective dataset, as shown in Figure 5.3. For three different datasets, the data dimensions are different, while the structure remains the same. The input layer of the CNN takes the feature vectors as the input in the form of a 2-dimensional (2D) image. The input layer is followed by two convolutional layers, maxpooling, and flatten layers. After flattening the data, we feed it to the two fully connected (FC) layers before applying the softmax activation to classify the activity.

We use Tensorflow 2.2.0 [1] with Keras 2.3.4 [37] to train the CNNs. Categorical cross-entropy is employed as the loss during training. The algorithm used for training is the Adadelta [177] optimizer with an initial learning rate of 0.001 and a decay rate of 0.95. We train the CNNs for 100 epochs using a batch size of 128. The training is performed on Nvidia Tesla GPU V100-SXM2 with 32 GB of memory.

CNN Accuracy Evaluation: The CNN shown in Figure 5.3 is trained for each UC obtained in Section 5.4.2. We use 60% data of each UC for training, while 20% of data is used for cross-validation during training. We train 10 networks with each UC with different initialization such that we can analyze both CCA distance and accuracy on an average basis. Next, we analyze the accuracy of the CNN for the UCs not seen during training using all the data of the unseen UCs. Figure 5.4 shows the average accuracy of CNNs trained with each of the four UCs

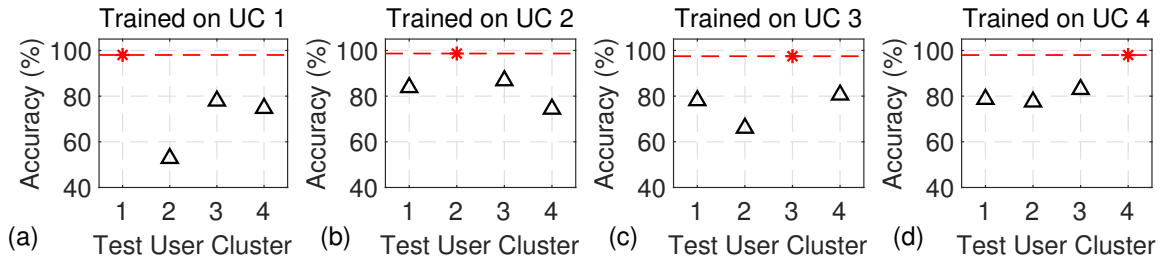


Figure 5.4: The accuracy of the CNNs tested with different UCs for the **w-HAR** dataset. The red star shows the accuracy of the UC used training while the triangles show cross-UC accuracy.

obtained from our dataset and tested on the other three. First, we see that the CNNs achieve a high accuracy on 20% test data of the UC used for training. However, there is a significant reduction in cross-UC accuracy. For example, the accuracy for UC 2 when tested on CNNs trained with UC 1 is only about 52%. Similar behavior is observed for other UCs as well, with the accuracy drop ranging from 10%–40%. This shows that the CNN is only able to learn the data pattern of the current UC, and it cannot generalize other UCs with distinct activity patterns. In the next section, we analyze the distance between networks trained with different UCs to gain better insight into the representations learned by the CNN.

5.4.4 Analysis of Distance Between Trained Networks

Background on CCA Distance: We employ CCA to analyze the distance between different networks and understand the representational similarity between network layers [113]. It analyzes the representational similarity between networks by analyzing the ordered output activations of neurons on a set of inputs, instead of working on the network weights directly. Taking the activation vectors of neurons from two layers (trained with different UCs or with different initializations) as inputs, CCA first finds the linear combinations of the activations such that they are as correlated as possible. Once the correlations are obtained, they are used to compute the

distance between the two activation vectors, i.e., between the two layers [113]. CCA has been successfully used to analyze neural network similarities for medical imaging [136], language models [146], and speech recognition [133]. We use the implementation proposed in [113] to analyze the distance between the CNNs trained for HAR.

Distance Between Networks Trained with the Same UC: To analyze which layers generalize between users, we calculate the mean CCA distance between the networks trained with the same UC. To this end, 10 CNNs with different initializations are trained with the training data of each UC. Next, we find the pairwise distances between the corresponding layers of the 10 CNNs trained with each UC. Finally, pairwise distances are averaged to find the mean CCA distance. Figure 5.5 shows the mean CCA distance among NNs trained with 4 UCs with the *w*-HAR dataset. Each sub-figure in Figure 5.5 shows the distance between networks trained with a single UC when tested on all the four UCs. For example, Figure 5.5(a) shows the mean distance for the 10 networks when they are trained on UC 1 and tested on all four user clusters. The figure shows that for the first four layers of the CNN (two convolution layers, one maxpooling layer, and one dropout layer), the mean CCA distance is low regardless of the input UC. Moreover, the distances are almost equal for all the four UCs. This shows that in

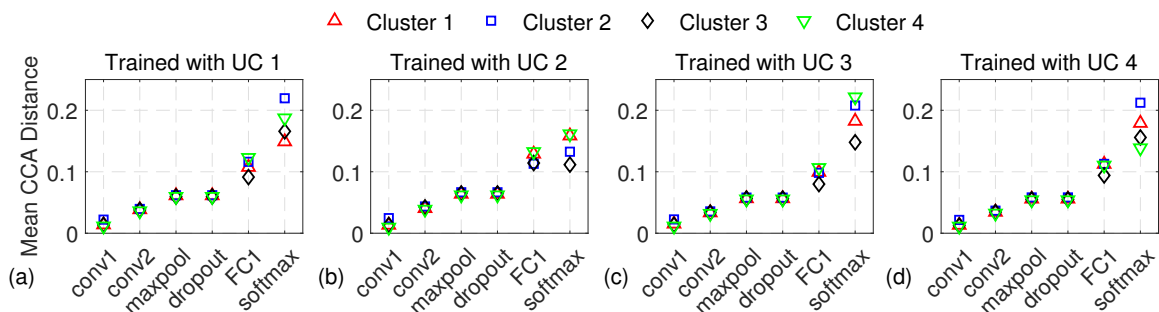


Figure 5.5: The CCA distance between CNNs trained with (a) UC 1, (b) UC 2, (c) UC 3, and (d) UC 4 from the *w*-HAR dataset when tested on all the four UCs.

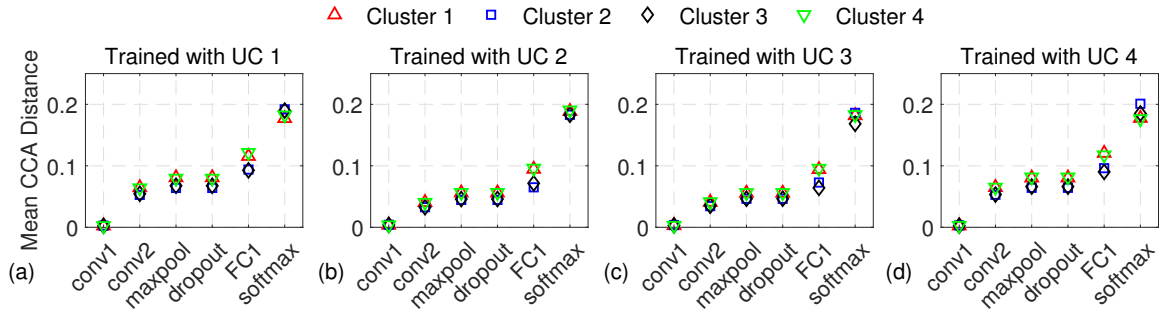


Figure 5.6: The CCA distance between CNNs trained with (a) UC 1, (b) UC 2, (c) UC 3, and (d) UC 4 from the **UCI HAR dataset** when tested on all the four UCs.

the first four layers learn features that are similar for all the users in the dataset. The networks start to diverge for different UCs from the first fully connected layer. The largest divergence is seen at the softmax layer, where the distance is lowest for the UC used for training. This means that the fully connected layers extract information that is specific to each UC and do not generalize to other UCs. We observe a similar trend for the UCI dataset, where the distance increases with layers, as shown in Figure 5.6.

In addition, we also calculate the CCA distance between networks trained with different UCs (*e.g.*, the distance between CNNs trained with UC 1 and UC 2). The analysis of the distance from this perspective helps in understanding the similarity between networks trained with users that have distinct activity patterns. Figure 5.7 shows the CCA distance between CNNs trained with UC 1 and the other three UCs from the w-HAR dataset. We see that the first four layers are similar to each other regardless of the input data. Furthermore, the last layer shows the highest divergence in the CCA distances when tested with the four UCs. Figures 5.8 and 5.9 show the CCA distances for CNNs trained with UCI HAR and UCI HAPT datasets, respectively. A similar pattern is observed for the UCI HAR and UCI HAPT datasets. The absolute distances are lower since there is a higher similarity between activity patterns of different UCs in the UCI datasets compared to the w-HAR dataset. In summary, this shows that

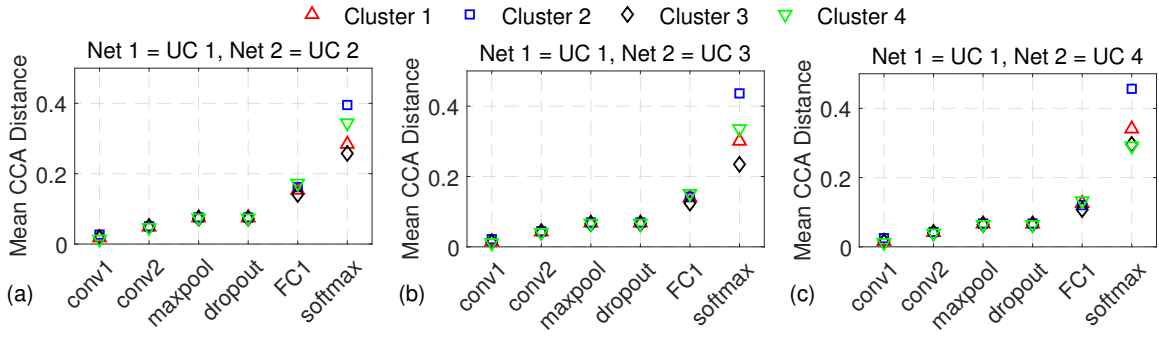


Figure 5.7: The CCA distance between CNNs trained with (a) UC 1 and UC 2, (b) UC 1 and UC 3, (c) UC 1 and UC 3 from the **w-HAR** dataset when tested on all the four UCs.

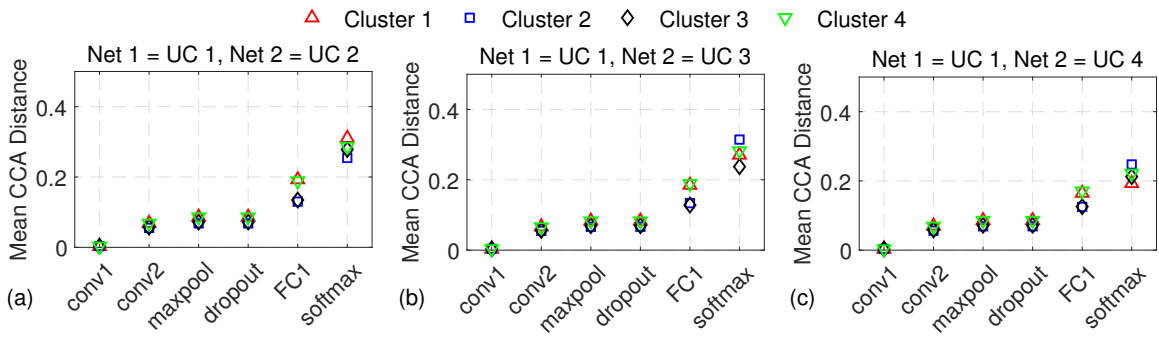


Figure 5.8: The CCA distance between CNNs trained with (a) UC 1 and UC 2, (b) UC 1 and UC 3, (c) UC 1 and UC 3 from the **UCI HAR** dataset when tested on all the four UCs.

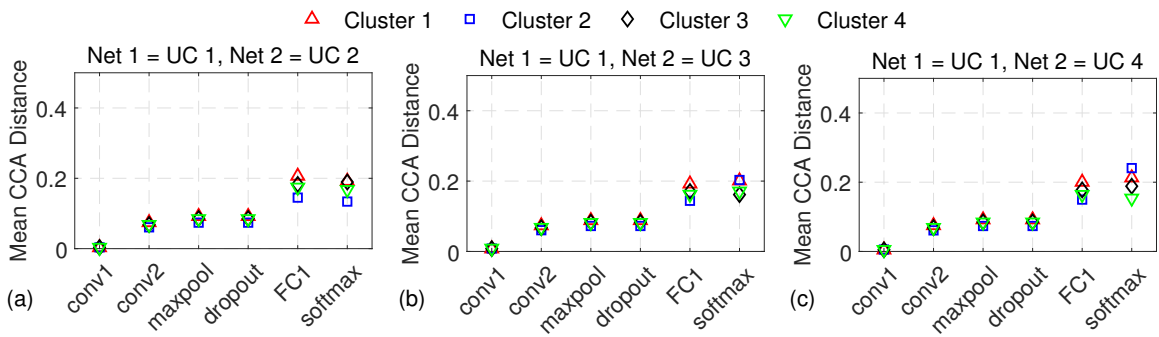


Figure 5.9: The CCA distance between CNNs trained with (a) UC 1 and UC 2, (b) UC 1 and UC 3, (c) UC 1 and UC 3 from the **UCI HAPT** dataset when tested on all the four UCs.

the first four layers learn general features that are applicable to all the UCs even when different UCs are used to train the CNN.

5.4.5 Transferring the NN and Fine-tuning

CCA distance analysis reveals that the convolutional layers provide general features, while the deeper layers provide the most distinguishing information. We use this insight to optimize the transfer learning process and improve the training time for new UCs. Specifically, we transfer the weights of the first four layers from a trained CNN to a network targeting another UC. Then, the deeper layers are fine-tuned with the data of the new UC. The fine-tuning process uses 60% of the new UC's data for training, 20% data for cross-validation, and the remaining 20% for testing. Following this process, we avoid training the convolutional layers, thus saving a significant amount of computations. We evaluate the accuracy after fine-tuning under two scenarios: 1) fine-tune the last FC layer and 2) fine-tune the last two FC layers. Figure 5.10 shows that the accuracy for new UCs improves significantly after fine-tuning either the last FC layer or the last two FC layers for the w-HAR dataset. With fine-tuning of one layer, we obtain 18% accuracy improvement on average. Specifically, the accuracy for UC 2 improves from 52%, 64%, and 75% to 90%, 92%, and 91%, respectively. When we fine-tune the last two layers, the accuracy improves to 95% for all UCs.

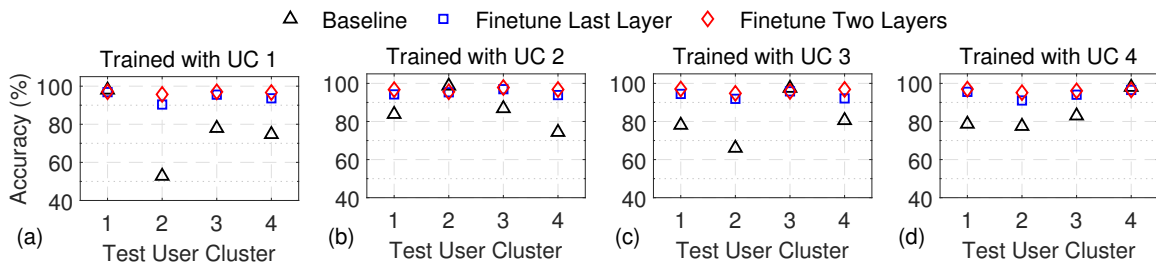


Figure 5.10: Comparison of accuracy between original and fine-tuned CNN for the **w-HAR** dataset.

5.5 Evaluations

5.5.1 Accuracy Analysis

The analysis performed in the previous sections showed results with user clusters that are designed to be as distinct as possible. In this section, we validate the transfer learning approach to randomly generated user clusters. To this end, we first generate 200 random user splits from the w-HAR dataset. Each user split contains four user clusters, in line with previous sections. We then train 5 CNNs for each cluster and test their cross-UC accuracy before applying transfer learning. The first column in Figure 5.11 shows the distribution of the cross-UC accuracy. The minimum cross-UC accuracy is 65% for all the UCs among 200 random user splits. We also show the minimum accuracy for the k-means clustering using a red dot. The minimum accuracy with k-means is 52%, which is lower than the minimum accuracy for all random user splits. This experimentally shows that our k-means clustering approximates the *worst-case scenario* well, where the users are as distinct as possible.

Next, we fine-tune the last one and two layers of the CNNs to capture the information specific to each user cluster, as shown in the second and third columns of Figure 5.11, respectively. The classification accuracy improves significantly after the fine-tuning process. Specifically,

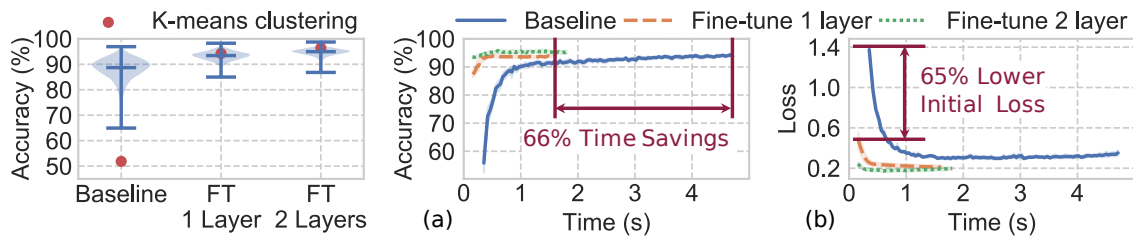


Figure 5.11: Comparison between original and fine-tuned NN for 200 UCs.

Figure 5.12: Transfer learning improvement analysis: (a) Training time, (b) Loss.

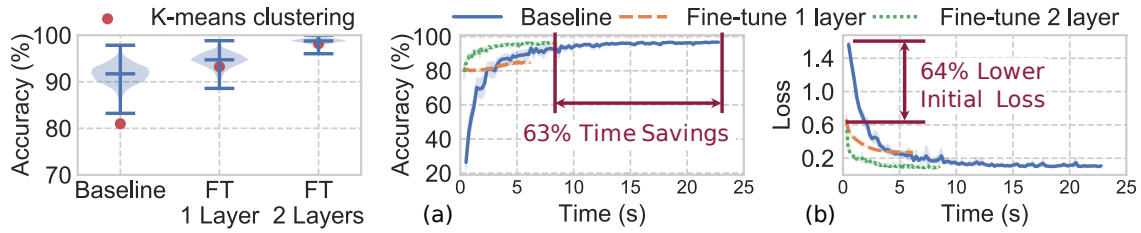


Figure 5.13: Comparison between original and fine-tuned NN for 100 UCs from the **UCI HAR** dataset.

Figure 5.14: Transfer learning improvement analysis: (a) Training time, (b) Loss for the **UCI HAR** dataset.

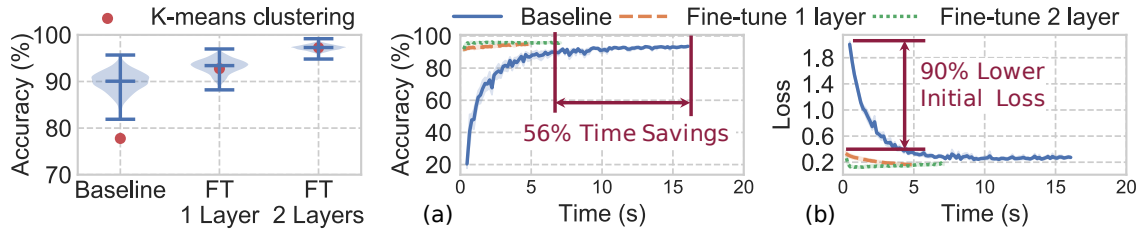


Figure 5.15: Comparison between original and fine-tuned NN for 100 UCs from the **UCI HAPT** dataset.

Figure 5.16: Transfer learning improvement analysis: (a) Training time, (b) Loss for the **UCI HAPT** dataset.

the median accuracy after fine-tuning one layer is about 93%, while it further increases to about 95% by fine-tuning the last two layers. These median accuracies are very close to the accuracy obtained for the k-means clustering. This shows our k-means clustering is representative of a wide range of randomly generated user clusters. We also note that some UCs achieve a higher accuracy after fine-tuning when compared to the k-means clustering because the users in these clusters have similar features. Conversely, some UCs have lower accuracy after fine-tuning because the source UC does not have all the activities present in the target UC.

We also analyze the accuracy of 100 randomly chosen UCs for the UCI HAR and UCI HAPT datasets, respectively. Figures 5.13 and 5.15 show that the median accuracy obtained for the 100 random clusters is similar to the accuracy obtained from the k-means clustering. This is in line with the results from the w-HAR dataset. In summary, the proposed transfer

learning approach of fine-tuning the deeper layers of the network significantly improves the classification accuracy.

5.5.2 Training Time, Loss and Convergence Analysis

Transfer learning provides benefits in training speed and convergence when compared to training from scratch for new user clusters. Figure 5.12(a) shows the comparison of training time between the baseline and the proposed transfer learning approach. The transfer learning approach has both a higher starting accuracy and lower convergence time with respect to the baseline. Specifically, fine-tuning one layer converges to 93% in about 1.6 s, which is 66% lower than the baseline approach of training from scratch. When two layers of the CNN are fine-tuned, the accuracy is higher than the baseline, with a small increase in the training time. The loss also shows similar behavior in Figure 5.12(b), where the starting loss with transfer learning is 65% lower when compared to the baseline. The loss at the end of training is also lower with the transfer learning approach. Next, Figure 5.14(a) shows the comparison of training time between the baseline and proposed transfer learning approach for the UCI HAR dataset. When two layers of the CNN are fine-tuned, the accuracy is the same as the baseline with 63% lower training time. Similar results are observed for the loss in Figure 5.14(b) where the starting loss with transfer learning is 64% lower than the baseline. Finally, Figure 5.16(a) and Figure 5.16(b) show the corresponding results for the UCI HAPT data set. In this case, we observe that the training time is 56% lower, while the starting loss is 90% lower. In summary, this shows that the general features transferred from a trained CNN aid in learning the activities of new users.

5.6 Summary

HAR has wide-ranging applications in movement disorders, rehabilitation, and activity monitoring. This chapter presented a transfer learning framework to adapt HAR classifiers to new users with distinct activity patterns. We started with a representational analysis of CNNs trained with distinct user clusters. This analysis revealed that initial layers of the network generalize to a wide range of users while the deeper layers providing distinguishing information. Based on this insight, we fine-tuned deeper layers of the network while transferring the other layers. Experiments on three HAR datasets showed that our approach achieves up to 43% and on average 14% accuracy improvement when compared to the accuracy without using transfer learning. Fine-tuning the deeper layers also leads to 66% savings in training time and better convergence while maintaining the same accuracy as training from scratch for new users.

REAP : RUNTIME ENERGY-ACCURACY OPTIMIZATION FOR ENERGY HARVESTING IOT DEVICES

6.1 Introduction

Wearable low-power internet-of-things (IoT) devices enable health monitoring, activity tracking, and wide-area sensing applications [13, 29, 96, 103]. These devices must stay on for as long as possible to analyze user activities. At the same time, they have to provide the maximum quality of service. These two objectives compete with each other since higher accuracy comes at the cost of larger energy consumption. Since weight and form-factor constraints prohibit large batteries, the feasibility of these devices depends critically on optimizing the energy-accuracy trade-off optimally at runtime.

Widely used dynamic power management techniques optimize the power-performance trade-off by switching between different power states. High-performance states are used to execute computationally heavy workloads at the expense of power consumption, while low-performance states are used to save power. In analogy, energy-accuracy trade-off in self-powered devices can be optimized by utilizing multiple design points. This is a challenging task, since the analytical characterization of the accuracy is much harder than developing power consumption and performance models. For example, we consider an activity recognition application where a wearable device infers the user activities, such as jogging, by processing motion sensor data. The recognition accuracy is a strong function of the users. Hence, energy-accuracy optimization requires *user studies* and *optimally chosen design points*, in addition to a *runtime optimization algorithm* that utilizes multiple design points.

This chapter presents a Runtime Energy-Accuracy oPtimization (*REAP*) framework for energy-constrained IoT devices. While our framework is general, we focus on health and activity monitoring applications where a wearable device processes sensor inputs to infer user activities. The recognized activities are sent to a gateway, such as a smartphone, for further processing. *REAP* co-optimizes the accuracy and active time under a tight energy budget. This optimization is enabled by the following three contributions.

User studies for accuracy evaluation: We perform experiments with 14 users to recognize the following activities: *sit, stand, walk, jump, lie down*, and *transitions* among them. During these experiments, we collect 3-axis accelerometer and stretch sensor data. We obtain a total of 3553 activity windows from these experiments. After labeling, we utilize this data for evaluating the accuracy of the human activity classifiers used in this chapter. We note that the data used in this chapter is a subset of the w-HAR dataset.

Pareto-optimal design points: A common baseline in activity monitoring applications is to obtain a classifier with the highest recognition accuracy [15]. High accuracy is obtained by using a sophisticated set of sensors, features, and classification algorithms, all of which imply a larger energy consumption, hence, lower active time. Other design points can be obtained by reducing the number of sensors and feature set to save energy. In turn, the energy savings lead to longer active time under a given harvested energy budget, albeit with lower accuracy. *To enable this work, we implemented 24 design points (DPs) with varying energy-accuracy trade-offs* on our hardware prototype. Among them, we choose 5 Pareto-optimal DPs as our primary designs used at runtime. We provide detailed execution time and power consumption breakdown for sensing, feature generation, and processing steps for each of these 5 DPs.

Runtime optimization algorithm: Given an energy budget, two fundamental objectives are to maximize the recognition accuracy and the amount of time the device is on, i.e., the active time. We first formulate this co-optimization problem assuming that there are N design points

with different energy-accuracy trade-offs. We define a general objective function that enables us to tune the importance of active time versus recognition accuracy. Then, we propose an efficient runtime algorithm that determines how much each DP should be used to optimize the accuracy-active time trade-off.

Experimental results using a custom prototype based on TI Sensortag [159] IoT board show that *REAP* outperforms all static design points under a range of energy budget constraints. *REAP* achieves both 46% higher expected accuracy and 66% longer active time compared to the highest performance DP. *REAP* also achieves comparable active time to the lowest energy design points while providing significantly higher expected accuracy. This makes *REAP* suitable for use in a wide range of energy harvesting profiles.

The major contributions of this chapter are as follows:

- A runtime technique to co-optimize the accuracy and active time of energy-harvesting IoT devices,
- Pareto-optimal design points with varying energy-accuracy trade-offs for human activity recognition (HAR),
- Experiments on a custom prototype with 14 user studies that show significant improvements both in expected accuracy and active time compared to static design points.

The rest of this chapter is organized as follows. In the rest, Section 4.2 reviews the related work. Section 6.3 and Section 6.4 present the accuracy-active time optimization problem and DPs used in this chapter, respectively. Finally, Section 6.5 presents the experimental results and Section 6.6 concludes the chapter.

6.2 Related Work

Energy harvesting for IoT devices has received significant attention due to their small form factor and low capacity batteries [81, 155]. These devices can be broadly categorized into two classes. The first class of devices rely solely on harvested energy and turn off when no energy is harvested [150]. The second class of devices uses a small battery as a backup to extend the active time [33, 83, 167]. These approaches manage the power consumption of the device such that the total energy consumed over a finite horizon is equal to the harvested energy [23]. As a result, the device has a long lifetime without battery replacement or manual charging. *REAP* applies to all devices that operate under a fixed energy budget.

Using ambient energy sources necessitates the development of energy allocation and duty cycling algorithms [33, 149, 167]. For example, the work in [83] uses linear programming to determine the duty cycle of the application as a function of the harvested energy. Similarly, the algorithm in [167] uses a linear quadratic controller to assign the duty cycle of the device while maintaining a set battery level. There are also algorithms for dynamic energy management of energy-harvesting devices for long-term energy-neutral operation [23, 33]. However, these approaches choose between on and off power states leading to sub-optimal operation. They also lack the notion of accuracy or any concrete application, unlike the approach in this chapter.

Human activity and health monitoring using wearable devices have significant impact potential to sports, patients with movement disorders, and the elderly [53, 80, 96]. A recent work presents a wearable system for mobile analysis of running using motion sensors [103]. The authors selectively identify the best sampling points to maintain high accuracy while reducing sensing and analysis energy overheads. The work in [80] presents a framework to detect falls by using a wearable device equipped with accelerometers. Authors in [29] design a classifier that detects physical activity using a body-worn accelerometer. It offers an accurate classi-

fier for human activity recognition, but it cannot sustain operation under tight energy budget constraints. Based on this observation, we find Pareto-optimal design points for the HAR application that offer varying levels of accuracy and energy consumption. Then, we use these design points to maximize the expected accuracy of HAR.

In summary, we present a unique combination of (1) a runtime energy-accuracy optimization technique, and (2) experimental evaluation with 5 concrete design points for HAR. We released the experimental data to the public to stimulate research in this area [16].

6.3 Runtime Energy-Accuracy Optimization

6.3.1 Preliminaries

We consider human activity monitoring applications implemented on energy-constrained IoT devices. We denote the period over which the total energy budget is provided as T_P , as summarized in Table 6.1. *REAP* computes the energy allocations at runtime with a period of T_P , which is set to one hour in our experiments. If the energy consumption over this period exceeds the amount of harvested energy and remaining battery level, the device powers down and misses user activity. Hence, our goal is to maximize the active time and the expected accuracy over a given period T_P .

Suppose that the IoT device can operate at N distinct DPs. The recognition accuracy achieved by design point i is denoted by a_i , while the corresponding power consumption is given as P_i for $1 \leq i \leq N$. In addition to these design points, we denote the time that the device remains off as t_{off} . Finally, the power consumption during the off period, which is due to the energy harvesting and the battery charging circuitry, is denoted by P_{off} .

Table 6.1: Summary of symbols used in the optimization problem.

Symbol	Description	Symbol	Description
T_P	Activity period	$J(t)$	Objective function
E_b	Energy budget	t_i	Active time of DP_i
a_i	Recognition accuracy of DP_i	α	Accuracy-active time trade-off parameter
P_{off}	Power consumption in the off state	P_i	Power consumption of DP_i

6.3.2 Optimization Problem Formulation

In a given activity period, the system may operate at different design points, resulting in varying levels of active time and accuracy. Let t_i denote the amount of time DP i is utilized during T_P . The active time of the device is simply given by the sum of the active times of each DP: $\sum_{i=1}^N t_i$. Likewise, the expected accuracy over the activity period can be expressed as $E\{a\} = \frac{1}{T_P} \sum_{i=1}^N a_i t_i$. The expected accuracy is a useful metric that incorporates both active time and accuracy, but it does not allow emphasis of one over the other. Therefore, we define a generalized cost function and solve the following optimization problem:

$$\text{maximize} \quad J(t) = \frac{1}{T_P} \sum_{i=1}^N a_i^\alpha t_i \quad (6.1)$$

$$\text{subject to} \quad t_{off} + \sum_{i=1}^N t_i = T_P \quad (6.2)$$

$$P_{off} t_{off} + \sum_{i=1}^N P_i t_i \leq E_b \quad (6.3)$$

$$t_i \geq 0 \quad 0 \leq i \leq N \quad (6.4)$$

Objection function $J(t)$: The parameter α in Equation 6.1 enables a smooth trade-off between the active time and accuracy. When $\alpha = 1$, the objective function reduces to the expected accuracy. Similarly, when $\alpha = 0$, the objective function reduces to total active time. In general,

the objective function gives a higher weight to the accuracy when $\alpha > 1$, and to active time when $\alpha < 1$.

Constraints: The constraint given in Equation 6.2 states that the sum of the active times and off period is equal to the activity period T_P . Similarly, Equation 6.3 specifies the energy budget constraint. The left-hand side gives the sum of the energy consumed in the off state and active states. The energy budget E_b on the right-hand side is determined by energy allocation techniques using the expected amount of harvested energy and battery capacity [83]. Finally, Equation 6.4 ensures that all active times are non-negative.

6.3.3 Runtime Optimization Algorithm

The solution to the optimization problem formulated in Section 6.3.2 gives the active times of each design point that maximize the objective function in Equation 6.1. We must solve this problem at runtime because the available energy budget is not known at design time. Furthermore, the importance given to accuracy versus active time (i.e., α) may change due to user preferences.

The optimization objective and the constraints in Equations 6.1- 6.3 are linear in the decision variables t_i and t_{off} . Thus, we use a procedure based on the simplex algorithm [41], as outlined in Algorithm 3. The inputs are the energy budget E_b , Pareto-optimal DPs, and the maximum number of iterations. The output is a vector with the values of decision variables t_i $1 \leq i \leq N$ and t_{off} that maximize the objective. We start the optimization process by constructing a tableau with the initial conditions. The first row of the tableau describes the objective function, while the other rows describe the constraints. In each iteration of the procedure, we find the pivot column by finding the column with the largest value in the last row of the tableau. Using the pivot column, we next find the pivot row in the tableau in Line 8 of

Algorithm 3: The *REAP* Procedure

Input : Design points, energy budget E_b , max. iterations
Output : Time allocated to each design point

- 1 Initialize the *tableau* with objective function and constraints
- 2 Add slack variables for inequality constraints
- 3 **while** $iter \leq max. iterations$ **do**
- 4 $PivotCol \leftarrow findPivotCol(tableau)$
- 5 **if** $PivotCol < 0$ **then**
- 6 **return** Optimal Solution
- 7 **end**
- 8 $PivotRow \leftarrow findPivotRow(tableau, PivotCol)$
- 9 Update the *tableau* using the *PivotCol* and *PivotRow*
- 10 **end**

the algorithm. Then, we update the tableau using the pivot column and row. The procedure terminates when all the entries in the last row are non-positive. In this case, the pivot column is set as negative, and the optimal solution is returned. *Our implementation takes 1.5 ms on our prototype.* Since the optimization algorithm runs every hour, it takes a negligible portion of the activity period and energy budget.

6.4 Human Activity Recognition Case Study

REAP is broadly applicable to energy-harvesting IoT devices that operate under a fixed energy budget. To illustrate the optimization results on a real example, we employ human activity recognition, i.e., HAR, as a driver application [15].

6.4.1 Background and Baseline Implementation

There is a steady increase in the use of wearable and mobile devices for the treatment of movement disorders and obesity-related diseases [53]. This technology enables data collection

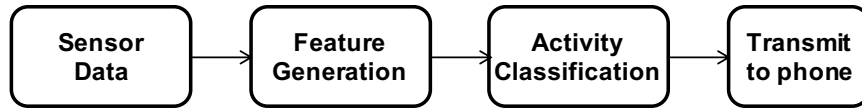


Figure 6.1: Overview of the human activity recognition application.

while the patients perform their daily activities. The first step in this effort is to understand what activity the user is performing at a given time. For example, the gait quality of the patient cannot be checked unless we know the user is walking. Therefore, HAR on mobile devices has recently attracted significant attention [96].

We implement a HAR application on a custom prototype based on the TI-Sensortag IoT board [159] and a passive stretch sensor, as outlined in Figure 6.1. It starts with the sampling of the accelerometer and stretch sensors. The streaming sensor data is processed using the TI-CC2650 MCU to generate the feature vector. This feature vector is then processed by a parameterized neural network to infer the activity of the user. Finally, the inferred activity is transmitted to a host device, such as a phone, using the Bluetooth Low Energy (BLE) protocol.

The energy consumption and recognition accuracy of HAR depends on the types of sensors used, the active time of sensors, the type of features, and the complexity of the classifier. The left side of Figure 6.2 shows the different configurations available for the sensors. For instance, we can use all three axes of the accelerometer or turn off selected axes to lower the energy consumption. In the extreme case, we can turn off the accelerometer to eliminate its energy consumption. Once we choose the configuration of the sensors, we can choose the sensing period, i.e., the time for which sensors are active, for each activity duration. By default, the sensors are turned on during the full activity duration. Turning off the sensors early, such as after 50% of the activity duration, provides energy savings at the cost of missed data points, hence, accuracy. We can also control the complexity of the features to trade off accuracy and energy consumption. Complex features, such as Fast Fourier Transform (FFT) and Discrete Wavelet Transform (DWT), offer higher accuracy at the expense of higher energy consumption.

Sensors		Computation		
Accel. axes	Stretch	Sensing period (%)	Signal features	NN structure
X, Y, Z		100	DWT of accel.	$4 \times 12 \times 7$
X, Y	Yes	75	16-FFT of stretch	$4 \times 8 \times 7$
X or Y	No	50	Statistics of accel.	4×7
None		40	Statistics of stretch	4×7




Figure 6.2: The knobs used to obtain design points with different energy-accuracy trade-offs.

In contrast, statistical features have lower energy consumption, albeit with lower accuracy. Finally, the structure and depth of the NN classifier can be controlled to obtain further energy-accuracy trade-off, as illustrated in Figure 6.2.

We exploit this trade-off between energy and accuracy to design 24 different DPs implemented on the TI-Sensortag based prototype, as described in the following section.

6.4.2 Pareto-Optimal Design Points

We design a total of 24 DPs by exploiting the energy-accuracy trade-off illustrated in Figure 6.2. We start by using all the axes of the accelerometer, generating complex features, and using an NN classifier with 2 hidden layers, which provide the highest recognition accuracy. Then, we progressively reduce the number of axes of the accelerometer and sensing period to reduce the energy consumption of the DPs. We always use the passive stretch sensor in our DPs, since it has low energy consumption. There is a need for detailed accuracy and energy consumption characterization of each DP to obtain the Pareto-optimal design points. To find the accuracy of each design point, we performed experiments with 14 different users. We use a total of 3553 activity windows from the experiments and labeled each window with the corre-

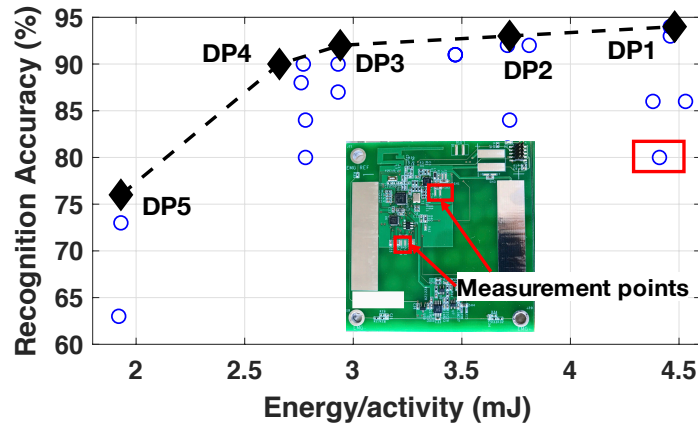


Figure 6.3: The energy-accuracy trade-off of various design points. The dashed line connects the selected 5 design points.

sponding activity. Each DP is designed using 60% of this data for training, 20% for validation, and the remaining 20% for testing.

All 24 design points are implemented on our prototype to profile the execution time and measure the power consumption using the test pads on our prototype. Figure 6.3 shows the recognition accuracy and energy per activity for each design point. As expected, each DP offers a unique energy-accuracy trade-off. For example, DP1 shows the highest accuracy with the highest energy consumption, while DP5 shows the lowest recognition accuracy and energy consumption. However, some design points do not offer any benefit in the energy-accuracy trade-off. For example, the design point marked with a red rectangle is dominated by DP2, DP3, and DP4. Hence, we consider 5 Pareto-optimal design points shown using black diamonds (DP1 to DP5) to validate the REAP framework. Table 6.2 summarizes the details of the configuration, accuracy, execution time, and energy for 5 Pareto-optimal DPs.

Design Point-1 (DP1): DP1 offers the highest accuracy by utilizing all three axes of the accelerometer for the entire activity window of 1.6 s. It uses 16-FFT of the stretch sensor data and statistical features of the accelerometer, such as the mean and standard deviations. DP1 has the

Table 6.2: Accuracy, execution time, power, and energy consumption of different human activity recognition design points.

Design point description		MCU exec. time distribution (ms)					Per activity summary			
DP no.	Features	Acc. (%)	Accel. feat.	Stretch feat.	NN	Total	MCU energy (mJ)	Sensor energy (mJ)	Energy (mJ)	Power (mW)
1	Statistical accel., 16-FFT stretch	94	0.83	3.83	1.05	5.71	2.38	2.10	4.48	2.76
2	Statistical y-axis accel., 16-FFT stretch	93	0.27	3.83	1.00	5.10	2.29	1.43	3.72	2.30
3	Statistical x- and y-axis accel. (0.8 s), 16-FFT stretch	92	0.27	3.83	0.90	5.00	2.10	0.84	2.94	1.82
4	Statistical y-axis accel. (0.6 s), 16-FFT stretch	90	0.14	3.83	1.00	4.97	2.09	0.57	2.66	1.64
5	16-FFT stretch	76	0.00	3.83	0.88	4.71	1.85	0.08	1.93	1.20

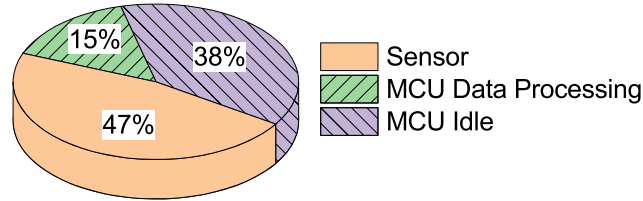


Figure 6.4: Energy consumption distribution of DP1 over one-hour activity period T_P . Total energy consumption is 9.9 J.

highest accuracy of 94% at the cost of the highest energy consumption of 4.48 mJ per activity. The energy breakdown in Figure 6.4 shows that about 47% of the energy consumption is due to the sensors. Thus, reducing the sensor activity is an effective mechanism to save energy.

Design Point-2 (DP2): DP2 reduces the sensory energy by utilizing only the y-axis of the accelerometer along with the stretch sensor. As depicted in Table 6.2, the energy consumption of the sensor reduces from 2.10 mJ to 1.43 mJ. It achieves an accuracy of 93%, which is 1% lower than DP1.

Design Point-3 (DP3): As shown in Figure 6.2, reducing the sensing period leads to lower energy consumption. DP3 exploits this by sampling the x- and y-axes of the accelerometer for

50% of each activity window, i.e., 0.8 s. As a result, the energy consumption of the sensor reduces to 0.84 mJ, and the total energy consumption of DP3 becomes 2.94 mJ per activity, while the recognition accuracy drops to 92%.

Design Point-4 (DP4): DP4 is similar to DP3, except that the sensing period of the accelerometer is further reduced to 40% (0.6 s). This reduces the energy consumption of DP4 to 2.66 mJ per activity, with a recognition accuracy of 90%.

Design Point-5 (DP5): DP5 uses only the stretch sensor for data features to minimize energy consumption. The energy consumption reduces to 1.93 mJ per activity, the lowest energy consumption among all our design points. However, it also has the lowest recognition accuracy of 76%.

Offloading to a host: Finally, we note that the raw sensor data can be directly sent to a host device, such as a smartphone or server, for processing. To assess the viability of this alternative, we implemented and measured its energy consumption. Sending the raw sensor data over BLE consumes 5.5 mJ per activity without any significant increase in the recognition accuracy. In contrast, transmitting just the recognized activity consumes only about 0.38 mJ per activity. Hence, offloading is not an energy-efficient choice.

6.5 Experimental Evaluation

6.5.1 Experimental Setup

IoT device: We use a custom prototype based on the TI-Sensortag IoT platform [159] to implement the proposed design points. The prototype consists of a TI CC2650 MCU, Invensense MPU-9250 motion sensor unit, a stretch sensor, and energy harvesting circuitry. Sensors are

sampled at 100 Hz and the MCU runs at 47 MHz frequency. Power measurements from the prototype and data from 14 user subject studies are used to obtain the 24 design points.

Energy harvesting data: We use the solar radiation data measured by the NREL Solar Radiation Research Laboratory to obtain the energy harvesting profile from January 2015 to October 2018 [4]. We use the profile for each hour within this data to generate the energy budget. These energy budgets are then used to evaluate *REAP* and the static design points in Section 6.5.4.

6.5.2 Expected Accuracy and Active Time Analysis

We first analyze the results of the proposed optimization approach as a function of the allocated energy over a one-hour activity period T_P . In the most energy-constrained scenario, the minimum energy required to run the energy harvesting and monitoring circuitry is 0.18 J. In the opposite extreme, 9.9 J energy is sufficient to run DP1, the most power-hungry design point, throughout T_P . Therefore, we sweep the allocated energy starting with 0.18 J, and find the optimal active time of each DP using the proposed approach.

Figure 6.5(a) shows the expected accuracy ($\alpha = 1$) as a function of the energy budget. The expected accuracy of all the design points approaches to zero when the energy budget is close to 0.18 J, since the device is almost always off. As the energy budget increases (Region 1), the accuracy of all DPs starts growing since they can become active. None of the design points can afford to stay 100% active under the energy budget in Region 1. We observe that the design point with the lowest energy consumption (DP5) achieves significantly higher accuracy because it can stay in the active state much longer. *REAP* matches or exceeds the accuracy of DP5 under the most energy-constrained scenario. When the energy budget goes over 4.3 J, DP5 can remain active throughout the activity period, but its recognition accuracy saturates. The other DPs benefit from more energy in Region 2, while *REAP* outperforms all by utilizing

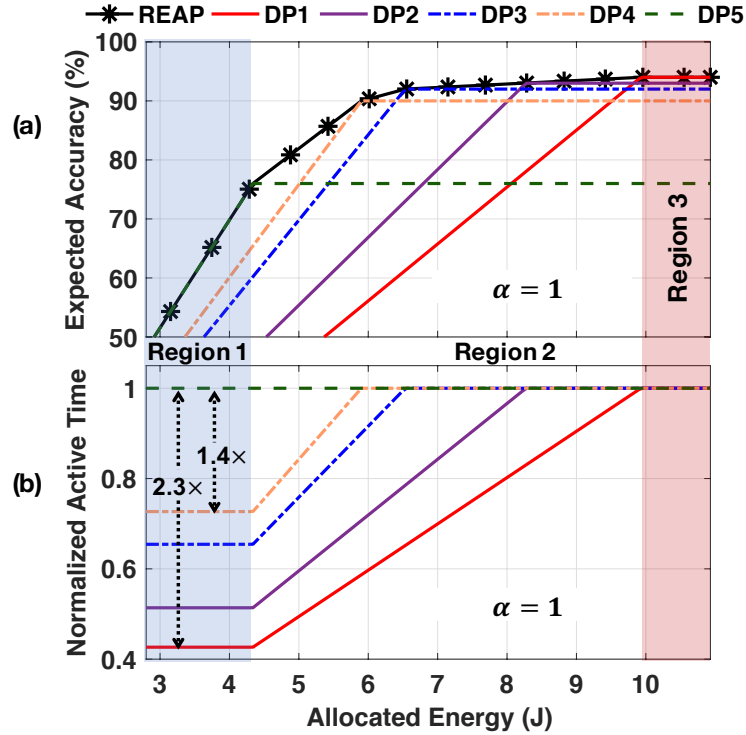


Figure 6.5: (a) Expected accuracy of *REAP* and design points. (b) The active time of each DP normalized to *REAP*.

them optimally. At 5 J energy budget, for example, *REAP* utilizes DP4 42% of the time and DP5 for 58% of the time to optimize the expected accuracy. Finally, all design points can remain active throughout the activity period when the energy budget is larger than 9.9 J. Hence, their accuracy saturates, and *REAP* chooses to DP1 beyond this point. In summary, *REAP* consistently outperforms or matches the accuracy of all individual DPs by utilizing *multiple DPs* optimally.

The active time of each DP *normalized to REAP* is plotted in Figure 6.5(b). DP5 is expected to have the longest active time since it has the least energy consumption. *REAP* successfully matches its active time in all the regions. In Region 1, *REAP* also achieves 2.3× larger active time compared to DP1 while providing significantly better accuracy. *REAP* consistently pro-

vides longer active times compared to DP1, DP2, and DP3 until the energy budget becomes large enough to sustain them throughout the activity period T_P .

6.5.3 Accuracy – Active Time Trade-off Analysis

Next, we analyze how *REAP* can exploit the trade-off between the accuracy and active time using the parameter α in objective function $J(t)$ in Equation 6.1. Since Section 6.5.2 considered the expected accuracy ($\alpha=1$), this section considers $\alpha > 1$, which gives more emphasis for higher accuracy.

As a representative example, Figure 6.6 shows the comparison of objective values of the 5 design points with *REAP* when α is set to 2. *REAP* always achieves higher performance than the lowest energy design DP5, since accuracy is given higher weight. The difference between *REAP* and DP5 increases further as *alpha* grows. When the energy budget is less than 6 J, DP4 outperforms all the other DPs, while *REAP* successfully matches it. In contrast, DP1, DP2, and DP3 have a very low performance since they are mostly in the off state. When the energy bud-

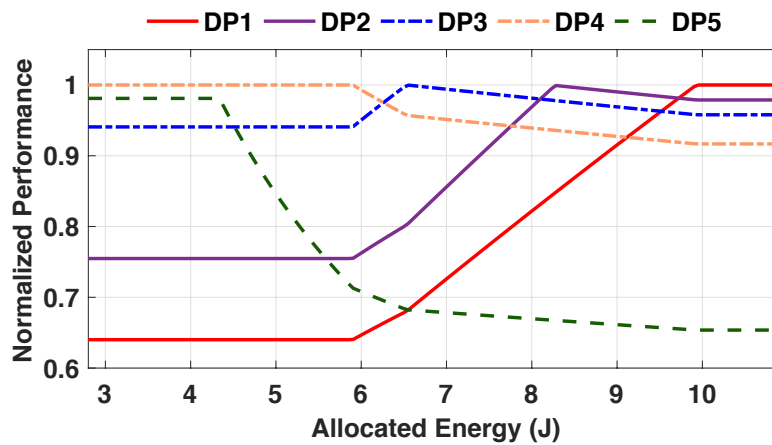


Figure 6.6: The objective value $J(t)$ of static design points (Equation 6.1) normalized to $J(t)$ of *REAP* with $\alpha = 2$.

get exceeds 6 J, there is sufficient energy to provide higher accuracy, but DP4 cannot exploit it. Hence, the higher accuracy design points become affordable and start outperforming DP4 one by one. Notably, *REAP* consistently outperforms or matches the static DPs, as we have also observed in Figure 6.5. For example, DP3 provides the same performance as *REAP* when the energy budget is 6.5 J. As the energy allocation increases beyond 6.5 J, *REAP* starts outperforming DP3 by optimally switching between DP1, DP2, and DP3. This trend continues until the energy allocation reaches 9.9 J, beyond which there is sufficient energy to support DP1 alone. Thus, *REAP* reduces to DP1 in this region. In summary, *REAP* exceeds or matches the performance of any individual DP.

6.5.4 Case Study using Real Solar Energy Data

In this section, we evaluate *REAP* under real solar radiation data measured by NREL Solar Radiation Research Laboratory at Golden, Colorado. This data is used to calculate the amount of energy that can be harvested by a flexible solar cell [57] on our prototype. Using the harvested energy budget, we compare the performance of *REAP* against the static DPs over an entire month. Figure 6.7 shows the performance of *REAP* normalized to DP1, DP3, and DP5 as a function of α . Due to space limitation, we plot the DPs with the highest performance (DP1), lowest energy (DP5), and best trade-off (DP3). Our gains with respect to DP2 and DP4 are larger than those of DP3.

When active time is emphasized in the objection function ($\alpha = 0.5$), *REAP* outperforms DP1 by $1.4\times$ – $2.2\times$ with an average improvement of $1.6\times$ across the month. DP1 suffers the most in this case as it has the largest energy consumption among all the DPs. Since accuracy becomes more important with larger α , the improvement of *REAP* over DP1 reduces. However, we still obtain $1.1\times$ – $1.3\times$ improvement even for $\alpha = 8$. We observe a similar trend in

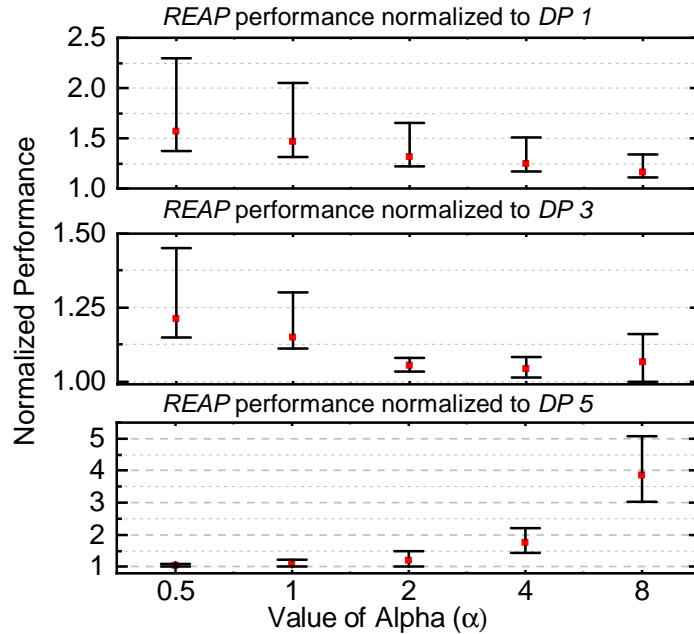


Figure 6.7: Performance (i.e., $J(t)$) achieved by *REAP* normalized to DP1, DP3, and DP5 during the month of September 2015. Error bars represent the range of improvement.

improvements for *REAP* over DP3 as well. The improvement is $1.1\times-1.4\times$ for $\alpha = 0.5$, and it gradually decreases with larger α . The improvements over DP3 are relatively lower since DP3 offers the best trade-off between energy consumption and accuracy among our Pareto-optimal design points.

Finally, we see that improvements over DP5 follow the opposite trend. When $\alpha = 0.5$, DP5 matches the active time of *REAP* due to its lower energy consumption. However, the performance of DP5 diminishes severely with increasing α . In summary, *REAP* can provide higher performance than any individual design point under any optimization objective. If the user needs higher accuracy, *REAP* can successfully adapt to the new requirements and tune the IoT device.

6.6 Summary

This chapter presented a runtime energy-accuracy optimization technique for energy-constrained IoT devices. The proposed approach dynamically chooses design points with different energy-accuracy trade-offs to co-optimize the accuracy and active time under energy budget constraints. To demonstrate the effectiveness in a realistic setting, we implemented a human activity recognition application on a custom IoT prototype. We presented 5 Pareto-optimal design points with different energy-accuracy trade-offs. We achieve 46% higher expected accuracy and 66% longer active time compared to the highest performance design point, and 22% to 29% higher accuracy than low-power design points without sacrificing the active time.

AN ULTRA-LOW ENERGY HUMAN ACTIVITY RECOGNITION ACCELERATOR

7.1 Introduction

The potential of human activity recognition (HAR) has led to both academic work [153] and commercial devices [69]. The nature of target applications requires the user to either carry or wear the device that performs HAR. Thus, a significant number of studies during the last decade used smartphones [153], while wearable solutions based on inertial measurement units (IMU) gained momentum with the advances in wearable electronics [96]. A recent user survey concludes that 27% of users give up using the device since charging is inconvenient, especially for those coping with a health problem [45]. Similarly, a significant number of users do not want to transmit their personal data to a different device due to privacy concerns [120]. Thus, practical solutions must perform HAR locally under an ultra-low energy budget that practically eliminates charging requirements.

Existing solutions on smartphones are inconvenient because the users need to carry a device. Furthermore, smartphones consume in the order of a few Watts [114] and fail to provide real-time guarantees since activity monitoring is not their primary goal. Low-power wearable devices can address these problems and bring the power consumption down to 10–30 mW [15]. However, this power consumption range is still significantly larger than the capacity of ambient energy harvesting sources, such as photovoltaic cells (indoor: 0.1 mW/cm^2) [125, 164] and human motion (0.73 mW/cm^3) [61]. Moreover, this power envelope still leads to merely 40 hours of operation using a wearable form-factor 1 g battery with 130 mAh capacity [51].

This chapter presents the first fully-integrated ultra-low power-hardware accelerator that

provides an end-to-end solution, from reading the sensors all the way to classifying the activity. The proposed HAR engine first reads in the raw sensor readings through a parameterized interface that can adapt to different sensor inputs. In this chapter, we employ a 3-axis accelerometer sensor and a capacitive stretch sensor sampled at 250 Hz and 25 Hz, respectively. Then, it pre-processes the raw sensor data using a moving average filter. After this step, it generates the features used for classification. Our flexible feature generation blocks allow generation of both simple statistical features and complex features, such as 64-point fast Fourier transform (FFT) and discrete wavelet transform (DWT) coefficients. Finally, the features are used by a deep neural network (DNN) to classify the following locomotion activities: $\{jump, lie\ down, sit, stand, stairs\ down, stairs\ up, walk\}$. To assist people with movement disorders, we focus on identifying the locomotion activity in which they are engaged.

The novel contributions of this chapter are threefold. *First*, we present a baseline HAR engine with a single-level classifier to quantify the impact of custom design over programmable solutions. Post-layout evaluations using a 65 nm TSMC technology show that our baseline design has a 1.353 mm² area. It achieves 95% accuracy while consuming 51 μ W active power and 14 μ W idle power. The baseline design consumes 22.4 μ J per activity, which is about two orders of magnitude lower than the best embedded solution reported in the literature [14, 15]. Detailed power consumption breakdown of our baseline design also reveals that the FFT feature generation and 3-layer DNN are the dominant components. Our *second* contribution is a novel activity-aware low-power HAR engine that uses the insights provided by our baseline design and the nature of human activities. This design classifies the activities first as *static* and *dynamic* using a simple support vector machine classifier and simple statistical features. If the output of the first level is *static*, then the actual activity is found using the same features with a decision tree. This design eliminates both complex features and the DNN classifier to reduce the dynamic power consumption to 5.56 μ W. If the output of the first-level classifier

is *dynamic*, then we use a smaller single hidden layer DNN to identify the activity. Our novel activity-aware HAR engine design achieves higher (97%) accuracy with a negligible penalty in the area. More importantly, the total power consumption drops from 51 μW to 20 μW and 45 μW for static and dynamic activities, respectively. *Finally*, the proposed designs are evaluated extensively with both in-house data from 22 user studies and a publicly available dataset [140].

The major contributions of this chapter are as follows:

- The first fully integrated custom HAR engine that integrates all steps from reading raw sensor data to activity classification,
- A novel activity-aware HAR engine that consumes the lowest energy (22.4 μJ per activity) reported in the literature,
- An extensive experimental study with HAR data from 22 users and post-layout evaluations using 65 nm TSMC technology.

The rest of this chapter is organized as follows: Section 7.2 reviews the related work and highlights our unique contributions. Section 7.3 gives an overview of the proposed baseline HAR engine, while Section 7.4 presents our activity-aware 2-level HAR engine. Section 7.5 describes the power consumption optimization techniques used in the chapter. Finally, Section 7.6 presents the experimental evaluation, and Section 7.7 summarizes our major contributions.

7.2 Related Work

Several recent studies have proposed special purpose hardware for human activity recognition [88, 105, 166]. Klinefelter et al. present a system on chip (SoC) that integrates an analog front-end (AFE), a microcontroller, energy harvesting capability, and a wireless modem for

Table 7.1: Comparison of the proposed HAR engine to relevant custom designs reported in the literature

Ref.	[171]	[106]	[166]	[104]	[32]	Proposed
Target App.	Vital signal monitoring	Vital signal monitoring	Signal acquisition	Signal acquisition	Sensor AFE for physical act.	HAR
Technology	130 nm	130 nm	180 nm	180 nm	500 nm	65 nm
Frequency	32 kHz	1-20 MHz	1 MHz	Up to 2 kHz	120 Hz	100 kHz
Voltage	1.0 V	0.9 V	1.2 V	1.1 V	2.7 V-3.3 V	1.0 V
Power	530 μ W	93–322 μ W	120 μ W	88.6 μ W	108–132 μ W	45–51 μ W
Area	16 mm ²	6.25 mm ²	49 mm ²	5.45 mm ²	196 mm ²	1.35 mm ²

biomedical applications [88]. Similarly, the work in [166] proposes a signal-acquisition SoC for personal health applications. The main focus of these studies is to develop an AFE for acquiring sensor data for health applications. The AFE is integrated into an ARM Cortex-M0 processor and multiply-accumulate units to provide digital processing capabilities. While this approach provides a low power consumption for the AFE, it has to use the ARM core for operations such as preprocessing and feature generation, leading to higher power consumption. The work in [105] develops a hardware accelerator for monitoring the change in activity of the user. Specifically, the authors implement a dynamic time warping-based decision-making module to detect movements of interest. Whenever the module detects a movement of interest, a sophisticated processing unit, such as a microcontroller, is activated to extract more information about the movement. While this design can reduce the idle power consumption, it still requires a higher-power microcontroller to classify activities. In contrast, our HAR engine provides a *fully integrated low-power solution* for all aspects of HAR from data preprocessing to classification.

Custom-designed solutions attracted significant attention in health and activity monitoring applications due to their power consumption advantages. For example, Wong et al. [171] present an SoC for vital sign monitoring implemented in the 130 nm technology. The proposed

SoC integrates a full-custom hardware MAC, digital microprocessor core and I/O peripherals, analog to digital converter, wireless transceiver, and custom sensor interfaces. Its power consumption is 530 μW at 1 V supply voltage, as shown in Table 7.1. A more recent vital signal monitoring SoC, also implemented in 130 nm technology, achieves 93–322 μW power consumption at 0.9 V supply voltage [106]. Similarly, a signal-acquisition SoC for personal health applications is proposed in [166]. This design achieves 120 μW power consumption at 1.2 V supply voltage. A more specialized ASIC design for ECG signal acquisition achieves 88.6 μW power consumption by operating at lower than 2 kHz frequency at 1.1 V supply voltage [104]. Finally, a generic sensor front-end architecture for physical activity monitoring systems is presented in [32]. This design provides a flexible way to build a complete sensor interface chip which consumes 120 μW in ON-state.

In this chapter, we propose an ultra-low energy end-to-end *hardware accelerator* for HAR. We use a commercial AFE to read sensor data and feed it to the accelerator. We demonstrate the proposed HAR engine on the 65 nm LP commercial technology using activity data from 22 users. To the best of our knowledge, this is the first hardware implementation that accelerates all steps of digital processing. It leads to three orders of magnitude higher energy-efficiency compared to existing *software implementations* on low-power microcontrollers [15]. Furthermore, it has competitive power consumption and area compared to relevant ASIC implementations reported in the literature, as summarized in Table 7.1.

7.3 The Proposed Baseline HAR Engine

The baseline HAR engine implements the HAR application presented in Chapter 4. We repeat brief explanations of the processing blocks for completeness of this chapter. Readers familiar with Chapter 4 can skip the following description and jump to Section 7.4.

7.3.1 Input Data

The input to the proposed HAR engine is the raw sensor data. The choice of sensors affects both classification accuracy and power consumption. In this work, we employ a 3-axis accelerometer, one of the most commonly used sensors, and a stretch sensor. In our user studies, we also collected 3-axis gyroscope data. However, experiments show that adding them does not increase the accuracy significantly, even though it incurs a 1-10 mW power consumption overhead.

3-Axis Accelerometer: The proposed HAR accelerator receives the streams from a 3-axis accelerometer. The sampling rate of this device is set to 250 samples/s and each output sample to the accelerator consists of three 16-bit words for x, y, z axes of the accelerometer, respectively.

Stretch Sensor: The textile-based stretch sensor [117] measures the degree of bending at the joints of our body. In our design, the stretch sensor is attached to one knee of the user. The output of the stretch sensor is a 16-bit capacitive value, which is normalized to have a range similar to the accelerometer. We use a 25 Hz sampling rate and stream the data from the sensor to the proposed HAR engine.

7.3.2 Preprocessing the Raw Sensor Data

Raw sensor data is commonly preprocessed to filter out the noise and prepare for feature generation. The proposed design employs a moving average filter with a width of 8 samples to smooth the input data. All four streams, i.e., 3-axis accelerometer and stretch data, flow through the filter. Since the body acceleration provides useful information about the user movement, we also compute it using the filtered outputs as follows:

$$b_{acc} = \sqrt{a_x^2 + a_y^2 + a_z^2} \quad (7.1)$$

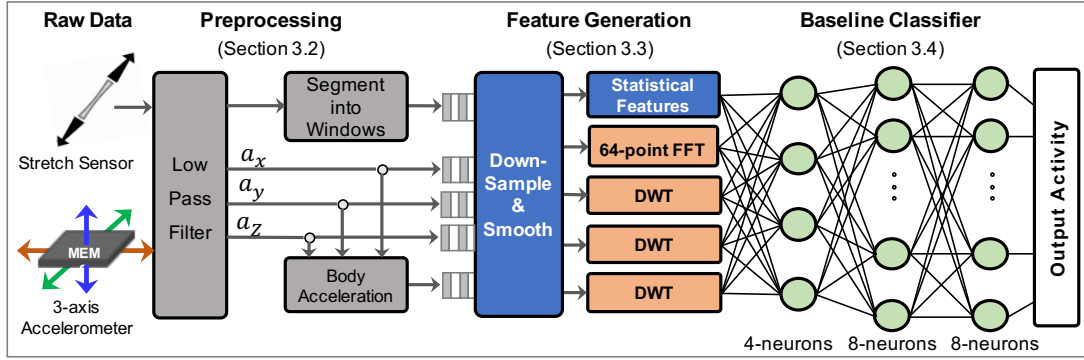


Figure 7.1: Architecture of the baseline HAR Engine

The final preprocessing step is segmenting the data into activity windows. This is necessary because more than one-second of data is required to identify the underlying activity. Each new sample is first stored in a FIFO buffer to segment the data efficiently in real-time. Meanwhile, the segmentation block computes the five-point derivative of the stretch data to identify trends in the activity, such as flat, increasing, and decreasing regions. It marks the boundary of a window when a new trend is detected or the maximum window duration (3 seconds in our design) is reached, as shown in Figure 7.1.

The segmentation block also plays a key role in minimizing the energy consumption of the proposed HAR engine, which is one of our major goals. We note that the feature generation and activity classifier can stay mostly in a low-power mode, i.e., clock or power gated, until the data of a whole window is populated. We utilize the output of the segmentation block to enable the feature generation and activity classifier blocks. After completing the classification of the current activity in those blocks, they immediately go back to the low-power mode until the enable signal from the segmentation block is detected, as described in Section 7.5.

7.3.3 Feature Generation

Downsampling and Smoothing: Once the segmentation block marks the completion of an activity window, the feature generation block in Figure 7.1 starts reading the data from the FIFO. The DNN classifier requires a fixed-length of features at its inputs, while the duration of the activity window in the segmentation block is variable. For example, the duration of dynamic activities, such as walk and run, is smaller than those of static activities. Moreover, the activity duration may show large variations across different dynamic activities and users, even for the same activity.

The feature generation step starts with a block to produce a fixed-length feature data set regardless of the activity duration. For this purpose, the segmentation block generates an 8-bit output that specifies the length of the activity window. This length is then used to determine the downsampling rate required to reduce the input data size to the feature data size. In this work, the accelerometer is downsampled to 64-sample windows, and stretch data is downsampled to 32-sample windows since larger values do not increase the accuracy. At the same time, our parameterized design allows choosing smaller values to reduce the area and power consumption. Once the rate is determined, the downsampling block (DS) reads input sample data from the FIFO and selectively stores it into its output buffer.

Fast Fourier Transform Features: The stretch sensor data generally shows a periodic pattern for dynamic activities, such as walking, stairs down, and stairs up. This means that FFT coefficients can be utilized to capture the periodicity. To this end, we append two consecutive activity windows, the minimum length required to compose a periodic signal. Then, we take the 64-point FFT of this signal to characterize the frequency spectrum. Among the 64 FFT coefficients, we include the first 16 coefficients in the feature vector since this is sufficient

to capture frequencies up to 8.25 Hz, which is higher than the frequency observed in human motion.

Discrete Wavelet Transform Features: The accelerometer data is typically noisy. Therefore, we use the approximation coefficients of the DWT to obtain robust features from the accelerometer data. Analysis on a high-level reference implementation in Python with Keras APIs and Tensorflow-backend shows that approximation coefficients of a single-level DWT are sufficient to capture the acceleration of human activity at 0 to 32 Hz. The proposed HAR engine produces 32 DWT coefficients for a_x , a_z , and b_{acc} . Thus, the subsequent DNN classifier uses a total of 96 DWT features, each represented as a 16-bit number.

Statistical Features: Statistical features of the sensor data also provide useful information for static activities, such as sitting and standing, without requiring complex processing. We utilize the minimum and maximum values observed in an activity window for all four streams, i.e., 3-axis accelerometer and stretch data. In addition, we compute the mean of a_z and the variances of a_x , a_y , a_z , and b_{acc} using the accelerometer sensor data. These specific features are selected based on our extensive analysis on the Python reference implementation. The details of this step are omitted since feature selection is not the focus of this chapter.

7.3.4 Single-Level Baseline DNN Classifier

The last step of the proposed HAR engine is the DNN classifier, as depicted in Figure 7.1. To determine the size of the neural network, we performed a design space exploration with one and two hidden layers. Within each of these structures, we varied the number of neurons in the hidden layers. Then, we trained each neural network and evaluated the accuracy. We observe that using two hidden layers with the ReLU activation function provides the best accuracy for the baseline classifier. The first hidden layer has 4 neurons, while the second hidden layer has

8 neurons. The output layer of the DNN classifier has 8 neurons, one for each activity. We use a linear activation in the output layer and then choose the activity with the maximum value as the output activity. The choice of max function in the output layers allows us to avoid the use of exponents in the softmax function, which is more commonly employed to obtain the classification output. Next, we describe the operation of the DNN and proposed optimizations performed to improve the implementation.

Architecture and Operation: Our DNN architecture consists of two finite state machines. The first state machine governs the state of the overall network (*State-1*), which also corresponds to the state of the first layer (L1). It has four states: *Init*, *WeightLoad*, *Idle*, and *Busy*. The second state machine (*State-2*) governs the remaining, i.e., second and output layers, as well as the max function at the output. The DNN operates as follows using these states:

- *Init*: On power-up, *State-1* enters the *init* state, where all registers are reset to their default values.
- *Init* → *WeightLoad*: *State-1* moves to the *WeightLoad* state when it receives an enable signal that indicates new DNN weights are available for loading. Then, the input weights are loaded from an off-chip ROM to corresponding memory for each neuron. In our implementation, there are a total of 596 weights, each represented with 16 bits.
- *WeightLoad* → *Idle*: After the weights are loaded, *State-1* enters the *Idle* state. In this state, the DNN classifier waits for an input valid signal that indicates the presence of new input features.
- *Idle* → *Busy*: When a new set of input features is available at the inputs of the DNN classifier, *State-1* moves to the *Busy* state. In this state, the classifier first registers the 120 input features in parallel. Then, the neurons in the first layer process the registered inputs and activate the second layer. Subsequently, the second and output layer are activated one by

one to produce the classifier output. Finally, the output flag is raised at completion of the max block.

Optimizations: The DNN classifier in the proposed HAR engine is parameterized to facilitate configurability and scalability. The basic building block is a parameterized neuron module. It takes the number of inputs to the neuron, weights, and features as input parameters. Therefore, this module can be instantiated in all three layers of the DNN with the appropriate parameters. We have further parameterized the multiply-accumulate (MAC) block, the weight memories, ReLU, and Max functions to facilitate design space exploration. Since the parameterized neurons are used to construct the hidden layers and the output layer, the architecture of the DNN can be changed easily. For example, only an hour was required to do the necessary changes and verification in going from the two hidden layer architecture used in the baseline HAR engine (Figure 7.1) to the single hidden layer architecture used in the activity-aware HAR engine presented in Section 7.4.

7.4 Activity-Aware 2-Level HAR Engine

This section presents a novel hierarchical HAR engine using the insights gained from the implementation of the baseline design. Analyzing the layout of the baseline classifier reveals that the DNN classifier and FFT blocks are the two major contributors to the power consumption and area, as detailed in Section 4.7. These blocks can be avoided for static activities whose complexity is significantly lower than that of dynamic activities. Moreover, it is relatively easy to distinguish static and dynamic activities using a simple two-class classifier. Therefore, we first employ a support vector machine (SVM) classifier to determine if the activity is static (*lie down, sit, stand*) or dynamic (*jump, stair down, stair up, walk*), as shown in Figure 7.2. If the outcome is static, then we invoke a relatively simpler decision tree to further classify the

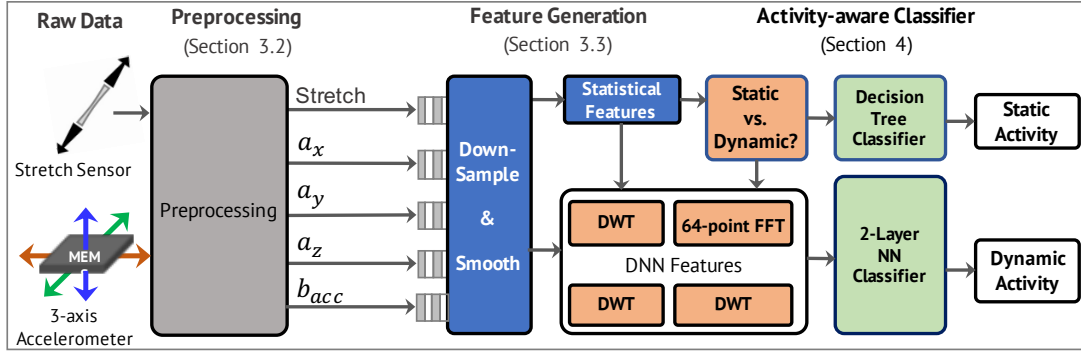


Figure 7.2: Architecture of the activity-aware HAR engine

activity. Otherwise, we still employ a DNN classifier, albeit a smaller one compared to the baseline design, to maintain high accuracy and facilitate future online learning. We note that its energy consumption overhead will be small since it will be powered down when it is not active. Our modular and parameterized baseline design enables us to reuse the preprocessing and most of the feature generation blocks, as shown in Figure 7.2. Therefore, we focus on the new blocks and the overall operation in this section.

7.4.1 Two-Class SVM Classifier

The first step of the 2-level activity-aware HAR engine is to differentiate between static and dynamic activities. We use an SVM to classify between the two types of activities since it is easier to implement using fewer resources than a DNN classifier. Moreover, the SVM classifier uses exclusively statistical features to avoid FFT and DWT computations. These features include the minimum, maximum, mean, and variance of the stretch sensor, 3-axis acceleration (a_x, a_y, a_z), as well as the body acceleration (b_{acc}). In addition, we also include the length of the window as the final feature. Using these features, we train the SVM with our user data. Once we obtain the weights of the SVM, the activity type is evaluated at runtime

using the following equation:

$$y = b + \sum_{i=1}^N \beta_i x_i \quad (7.2)$$

where b is the value of the bias, x_i are the features, and β_i is the weight for the i^{th} feature. The activity is classified static if $y < 0$, and dynamic otherwise. Depending on the output of the SVM classifier, we use a decision tree or a DNN to further classify the activity, as described in the following sections.

7.4.2 Decision Tree (DT) Classifier for Static Activities

If the level-1 SVM classifier marks an activity as static, we identify the activity using a decision tree, which can be easily implemented in hardware using comparators. Besides its simplicity, the DT classifier provides greater than 95% accuracy for static activities, as shown in the experimental results.

The DT classifier uses the same features as the SVM classifier, i.e., the statistical features for the accelerometer and stretch sensor data, as shown in Figure 7.2. Reusing the features allows us to compute them only once and optimize the power consumption. Furthermore, the neural network, FFT, and DWT blocks remain in low-power states leading to additional power savings. The decision tree is implemented as a series of comparators, where each node of the tree consists of a comparator. Depending on the output of the comparator, we choose the next branch of the tree. This process continues until we reach a leaf node and the activity is identified.

7.4.3 DNN Classifier for Dynamic Activities

A DNN classifier is used to identify the activity when the output of the level-1 SVM classifier indicates a dynamic activity. The DNN classifier in the activity-aware 2-level HAR engine has to identify only four activities and the transitions between them. Therefore, the DNN used herein is smaller compared to the one used in the baseline HAR engine. Specifically, the DNN classifier has one hidden layer with 4 neurons and an output layer with 5 neurons. Similar to the baseline design, we use ReLU activation in the hidden layer and softmax classification in the output layer. The feature input to the DNN classifier is the same as the input to the baseline DNN classifier (i.e., the same 120 features). To generate these features, we enable the DWT and FFT blocks whenever the SVM classifier outputs a dynamic activity. Once the features are generated, we evaluate the DNN to obtain the final activity classification.

7.5 Low-Power Optimizations

One of the most important goals of the proposed HAR engines is to operate within the harvested energy budget of wearable devices [23]. This section presents the low-power optimization techniques used in our design that will help in enabling operation under harvested energy budget.

7.5.1 Clock and Data Gating

Human activities typically occur in the order of a few Hertz [99]. We leverage this information to deactivate the part of blocks which are not used all the time. For example, the feature generation starts after segmentation and downsampling are completed. Similarly, there

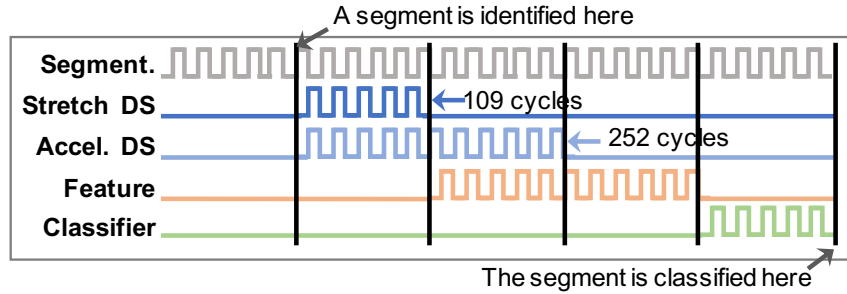


Figure 7.3: A representative timing diagram for the activation of the blocks in the proposed HAR engine. The active times of Feature and Classifier clocks are detailed in Table 7.9.

is a data dependency from feature generation to classification, as shown in Figure 7.3. In total, there are three major dependencies:

- Stretch and accelerometer downsampling (i.e., Stretch DS and Accel. DS in Figure 7.3) depend on the completion of segmentation
- Feature generation dependency is twofold. FFT computation and stretch statistics depend on stretch sensor downsampling. DWT computation and accelerometer statistics depend on accelerometer downsampling.
- Classifier depends on feature generation

We leverage these dependencies to design a custom clock gating solution for HAR applications. The clocks of the stretch and accelerometer downsampling blocks become active only after a valid segment is identified. Since there are fewer stretch sensor data samples than the accelerometer, stretch downsampling finishes earlier and enables the FFT feature generation block (orange line). After downsampling the accelerometer samples, the rest of the feature generation blocks (e.g., DWT) are enabled. Once all features are generated, the clock of the classifier blocks is enabled (green line).

Figure 7.4 shows the clock gating logic for the segmentation and downsampling blocks. The control signals, called (“output_valid”), are asserted to notify the downstream block when the preceding operation is complete. This signal is connected to the “set” input of an SR-

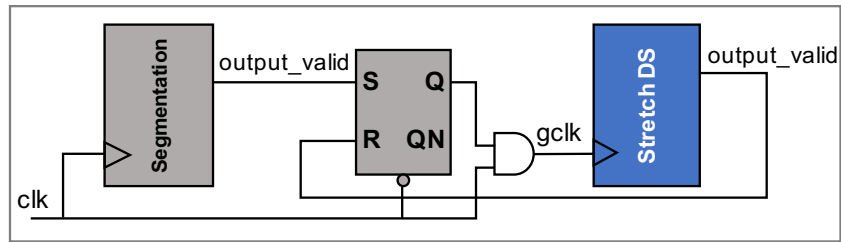


Figure 7.4: Clock gating control logic

Latch. When it is asserted, the output Q of the latch becomes high. Consequently, the gated clock “gclk” enables the operation of the downsampling block. The “output_valid” of the downsampling block is connected to the “reset” input of the same latch. In this way, the gated clock stops when “output_valid” becomes high, i.e., when downsampling is completed. Note that we use a negative latch to prevent any glitches and additional delay in the gated clock.

Clock Gating Implementation Overhead: The clock gating logic uses only an SR-Latch and an AND gate, which introduces negligible overhead. The latch introduces a half-cycle delay before the operation of the next logic begins. To compensate for this and to not lose any data, the outputs of the left-hand logic are registered before they are passed on to the next logic.

The proposed clock gating logic is implemented for all dependencies in the design presented earlier. As a result, we end up with one global clock and four gated clocks for both the baseline and activity-aware HAR engines. Figure 7.5 illustrates which blocks run on different clocks with different colors. Note that these clocks are gated versions of the same global clock, so their frequencies are the same.

7.5.2 Power Gating

Clock gating is useful for reducing the dynamic power consumption, while power gating can reduce both leakage and dynamic power consumption. Since the feature generation and classification blocks stay mostly idle, power gating is a promising power optimization technique. To this end, we divide the proposed HAR engine into two power domains, as illustrated in Figure 7.5. The first power domain (PD1) contains the preprocessing and segmentation blocks, which are always on. The second power domain (PD2) integrates the feature generation and classification blocks, which become active intermittently. In this way, these blocks can turn on only after a segment is identified and turned off once the activity is classified.

The power gating functionality is implemented using a power control unit (PCU). Since the PCU must be always on, it operates on PD1 and the global clock. The PCU works on the same basic principles as the clock gating logic shown in Figure 7.4. By default, the second power domain is turned off during preprocessing and segmentation. The “output_valid” signal of the segmentation block is the first input to the PCU. It is asserted when a valid segment is ready. When this input is received, the SR-latch inside the PCU turns on PD2 by driving the sleep transistor, as shown in Figure 7.5. Then, PD2 stays active until the classifier block output becomes valid.

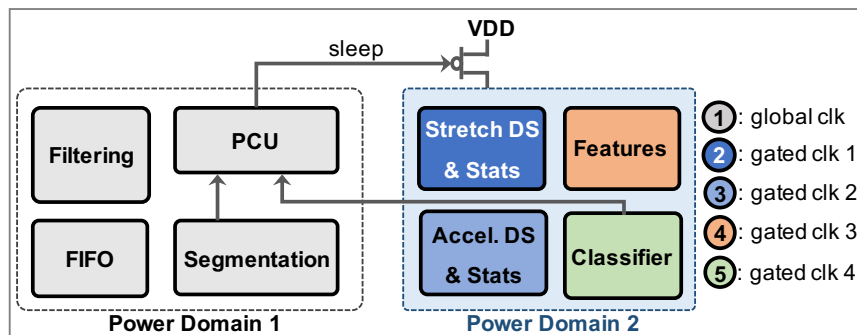


Figure 7.5: Power and clock domains in the proposed design

Power Gating Implementation Overhead: Multiple power domains introduce three distinct overheads. First, the sleep transistor introduces additional leakage current, which can offset the potential power savings, if it is significant. To minimize the leakage of the sleep transistor, we employ high-threshold-voltage low-leakage PMOS header transistors. The high threshold voltage transistors typically have leakage power in the order of nanowatts⁹. Since the leakage power of the proposed HAR accelerator is in the order of microwatts, power gating can achieve significant power savings by lowering the leakage current. The second overhead is the wake-up time, which is typically less than one microsecond. The proposed HAR accelerator receives new sensor samples every few milliseconds when operating at 100–250 Hz sampling frequencies. Therefore, the wake-up time is negligible for the proposed HAR engine. Finally, the PCU and the sleep transistor increase the total area of the design. In our implementation, the logic required for controlling the sleep transistor is a single SR-Latch, which represents a negligible area overhead. Furthermore, the sleep transistor also incurs less than 5% area overhead, which is small when compared to the potential savings in the leakage power. In summary, using two power domains enables significant power savings with a small overhead.

7.5.3 Use of Low-Power Classifiers

In the course of daily life, static activities occur more frequently than dynamic activities. To estimate the average distribution of activities, we performed an analysis of human activities using the American Time Use Survey (ATUS) [163], which is conducted by the United States Census Bureau to record how people spend their time during a day. According to this survey, people spend almost 84% of the time in static activities, such as lie down, sit, and stand. Dy-

⁹https://blogs.synopsys.com/magicbluesmoke/files/2007/10/sleep_transistor_sizing.pdf

dynamic activities, such as walking, running, and stairs up/down, constitute only about 16% of the total time.

Our activity-aware 2-level classifier exploits the distribution of activities to enable additional power and energy savings. First, the DNN, DWT, and FFT blocks are activated only for dynamic activities. For example, if the SVM block classifies an activity as static, these blocks remain clock gated. Consequently, they are activated less frequently compared to the baseline classifier. Furthermore, the DNN block uses fewer resources as detailed in Section 7.6.2 because it does not need to classify static activities. The benefits of using the activity-aware classifier in terms of power and energy consumption are evaluated in Section 4.7.

7.6 Experimental Evaluation

7.6.1 Experimental Setup

Design Tools and Hardware Technology: We first design the proposed HAR engine using Verilog Hardware Description Language (HDL). Then, we synthesize it with TSMC 65 nm low power (LP) technology using Synopsys Design Compiler (DC). To obtain the layout and the area of the design, we perform automatic placement and routing using Cadence Innovus. Finally, we use Synopsys PrimeTime to obtain the timing and power consumption of each block and the entire design. In addition to the low-power optimizations presented in Section 7.5, we also utilize the power optimization options available in the Synopsys Design Compiler to optimize the power consumption of the proposed HAR engine. The same optimization options are used for all designs for a fair comparison.

User Studies: We use the w-HAR dataset described in Chapter 4 to evaluate the accuracy of the designed HAR engines. The labeled data from w-HAR is used to extract the features and train

the DNN classifiers. After training the classifiers, the raw sensor data from the w-HAR dataset are fed to the HAR engine to segment the windows and perform the activity classification.

Training, Cross-validation, and Test data: Data samples from 4 randomly selected users are reserved exclusively for testing. Then, samples from the remaining 18 users are divided into 60% training, 20% cross-validation, and 20% additional test data. Overall, 37% of the test data comes from 4 unknown users.

7.6.2 Design Area

7.6.2.1 Baseline HAR Engine

Figure 7.6 shows the layout of the baseline HAR engine, which has a total area of 1.353 mm². While optimizing the floorplan, the order of the blocks is matched to the logic flow described in Figure 7.1. The area breakdown in Figure 7.8 shows that the FFT block has the largest area. This is expected as the FFT block is the most compute-intensive block in the design. Among other blocks, most of the area is occupied by blocks that have registers to store their input or output data. For example, the FIFO block occupies 21% of the total area. This is also expected since it has to store the filtered data until a new activity window is detected by the segmentation block. Similarly, the DNN block incurs the overhead of storing the weights and performing MAC operations. In contrast, the low pass filter and segmentation blocks take up a smaller area since they do not have to store a significant amount of data for their computations.

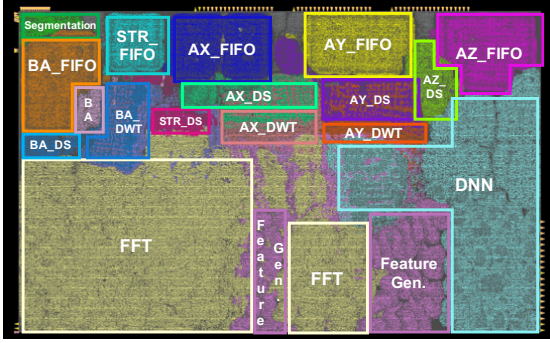


Figure 7.6: Floorplan of the baseline HAR engine

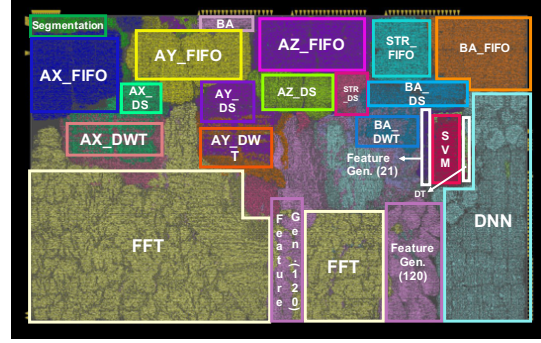


Figure 7.7: Floorplan of the activity-aware 2-level HAR engine

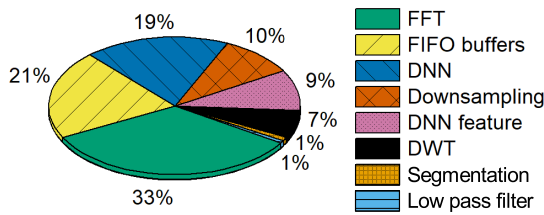


Figure 7.8: Area of the major blocks used in the baseline HAR engine

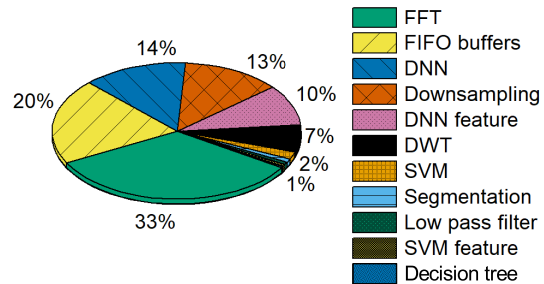


Figure 7.9: Area of the major blocks used in the hierarchical 2-level HAR engine

7.6.2.2 Activity-Aware 2-Level HAR Engine

Figure 7.7 shows the layout of the 2-level HAR engine, which shows a total cell area of 1.357 mm^2 . We observe that the layout closely resembles the baseline classifier except for the feature generation and neural network blocks. This is because both classifiers share the blocks for filtering, segmentation, and downsampling. The activity-aware 2-level design has additional blocks for the support vector machine classifier and decision tree for static activities. However, the total area of the 2-level classifier is only 0.3% more than the baseline classifier because it uses a smaller DNN with fewer neurons for dynamic activities. The 2-level classifier with a slightly larger area allows us to significantly improve the accuracy and lower the power consumption, as we show in the following sections.

Figure 7.9 shows the area breakdown of the activity-aware 2-level HAR engine. Similar to the baseline design, the FFT block consumes the largest area while the neural network block has a smaller area in the 2-level classifier due to its smaller size. Furthermore, the SVM and DT blocks consume a negligible portion of the total area despite their significant power and accuracy benefits.

7.6.3 Accuracy Evaluation

7.6.3.1 Baseline HAR Engine

We start the accuracy analysis of the baseline HAR engine by generating the feature vectors for each activity window in our dataset. To maintain compatibility with the hardware implementation, the features are stored in a 16-bit integer format. This feature data is then supplied to a neural network training algorithm implemented in Python with Keras APIs [37] and Tensorflow backend [1] using the parameters summarized in Table 7.2. These parameters are determined by sweeping them and evaluating the accuracy. For example, the batch size is incremented from 25 to 100 in steps of 25. Of these, a batch size of 50 gives the best accuracy for our dataset. Similarly, we sweep the number of training epochs from 100 to 500. 200 training epochs are used in the experiments since this is sufficient to achieve good accuracy while avoiding overfitting.

At the end of the training, we obtain the weights of the network in the floating-point format. Since our hardware implementation uses 16-bit integer precision, we uniformly quantize the weights and activations to 16 bits using the approach in [90]. For each layer of the neural network, we first find the weight with the maximum magnitude W_{\max} . Then, the quantization

Table 7.2: Parameters used for deep neural network training

Optimizer	Adam [86]
Loss function	Categorical cross-entropy
Initial learning rate	0.001
Epochs	200
Batch size	50

factor Δ_q is obtained as:

$$\Delta_q = \frac{2W_{\max}}{2^{16}} \quad (7.3)$$

The uniform quantization ensures that zero in floating-point precision is mapped to zero in integer precision as well. We use the quantization factor of each layer to obtain the quantized weights of the neural network. After quantizing the weights, we use the 16-bit integer features to quantize the activations in the neural network. Finally, we evaluate the accuracy of the quantized neural network for all the activity windows in our dataset.

Table 7.3 shows the confusion matrix for the baseline HAR engine. It contains one row and a column for each activity classified in this chapter. The rows represent the true activity, whereas the columns represent the activity classified by the proposed baseline HAR engine. The diagonal entries of the confusion matrix show the number of instances that are classified correctly. The baseline DNN classifier is able to recognize all the activities with few or no misclassifications. For example, the walk activity is classified correctly 1913 times out of 2007 activity windows, leading to 95% classification accuracy. We also observe that the walk activity is misclassified as either jump, stairs down, or transition. This happens since the jump and stairs down pattern of some users are similar to that of walk. All the instances of the lie down activity are classified correctly, leading to a 100% recognition accuracy. The lowest accuracy of classification is observed for transitions between the various activities. This is expected since the transitions typically contain features of two different activities, such as in stand to

Table 7.3: Confusion matrix for the baseline classifier

	Jump	Lie Down	Sit	Stand	Walk	Stairs Up	Stairs Down	Transition
Jump	442	0	0	0	5	0	5	6
Lie Down	0	474	0	0	0	0	0	0
Sit	0	0	665	26	0	0	0	5
Stand	0	0	16	576	1	0	0	27
Walk	31	0	1	10	1913	0	10	42
Stairs Up	0	0	0	0	1	101	6	1
Stairs Down	0	0	0	0	1	1	97	1
Transition	7	2	7	14	14	4	0	229

sit transition, which makes it harder for the classifier to identify them with high accuracy. In summary, the baseline DNN classifier achieves an accuracy greater than or equal to 93% for all the activities used in this chapter. Note that additional activities could degrade the accuracy, especially if they are close to the existing activities. More complex classifiers and features may be needed with increasing number and complexity of target activities.

Evaluation with the Opportunity Dataset [140]: In addition to our in-house data, the accuracy of the proposed HAR engine is also evaluated using the publicly available Opportunity dataset [140], which includes data from 4 users for a range of daily activities. Since our work focuses on ambulatory activities, we extract the corresponding data from the Opportunity dataset. Furthermore, we use only the accelerometers mounted on the right knee and the shoe since these setups are closest to the sensors used in our experiments. The data is divided into windows of approximately two seconds to compute the DWT and statistical features for each window. Finally, the DNN is trained with the same structure as our baseline neural network classifier using the extracted features. The proposed DNN classifier achieves an overall accuracy of 95% for the locomotion activities in the Opportunity dataset. This result shows that the proposed features and DNN classifier can classify activities from multiple datasets accurately.

7.6.3.2 Activity-Aware 2-Level HAR Engine

First-Level SVM Classifier: The activity-aware 2-level HAR engine recognizes the activities in two phases, as described in Section 7.4. The SVM classifier employs only the statistical features to determine if an activity is static or dynamic to minimize power consumption. The SVM parameters are determined by using the Statistical and Machine Learning Toolbox in MATLAB. We perform 10-fold cross-validation during training to ensure that the classifier is robust. After finding the weights of the SVM classifier, they are converted to 16-bit integer precision using Equation 7.3. Then, we obtain the confusion matrix for the SVM classifier using the integer weights, as shown in Table 7.4.

Our SVM implementation classifies only 38 windows out of 4740 activity windows incorrectly. More specifically, it misclassifies only 9 static and 29 dynamic windows, as shown in Table 7.4. Hence, it achieves more than 98% overall accuracy.

Table 7.4: Confusion matrix for the level 1 SVM classifier. The static and dynamic activities are classified with over 99% accuracy.

	Static	Dynamic
Static	1781	9
Dynamic	29	2921

Second-Level Decision Tree for Static Activities: Static activities are classified using a decision tree, as described in Section 7.4.2. We train the DT classifier in Weka [70] using 16-bit integer features for all the activity windows in the static class. Table 7.5 shows the confusion matrix for the three static activities in our data set. The DT classifier achieves high accuracy for all three activities. Overall, it classifies only 13 out of 1790 activities incorrectly. Furthermore, it has better accuracy than the DNN classifier used in the baseline HAR engine. For

Table 7.5: Confusion matrix for the activity-aware HAR engine for static activities

	Lie Down	Sit	Stand
Lie Down	473	0	1
Sit	0	691	4
Stand	8	0	612

instance, the number of misclassifications for the stand activity reduces from 44 to 8, which translates to an accuracy improvement from 93% to 99%. As a result, the two-level design has both accuracy and power consumption benefits.

Second-Level Neural Network for Dynamic Activities: The activity-aware HAR engine uses a neural network with a single hidden layer to classify individual dynamic activities, as explained in Section 7.4.3. The neural network uses the same features and training settings as the baseline DNN classifier. Similar to the baseline DNN, we quantize the weights and the activations using Equation 7.3.

The confusion matrix for the dynamic activities is shown in Table 7.6. We observe that the number of misclassifications for all the activities is similar to or fewer than the baseline DNN classifier. The largest improvement is seen for transitions where the number of mistakes decreases from 48 to 25, which is a drop of about 48%. The overall accuracy of the proposed second-level neural network classifier is 96%. In contrast, a decision tree classifier for dynamic

Table 7.6: Confusion matrix for the activity-aware HAR engine for dynamic activities

	Jump	Walk	Stairs Up	Stairs Down	Transition
Jump	445	8	0	1	4
Walk	19	1937	2	9	40
Stairs Up	1	1	105	1	1
Stairs Down	0	0	0	99	0
Transition	5	14	1	5	252

activities could achieve only 90% accuracy. In summary, the activity-aware 2-level HAR engine improves the classification accuracy while also reducing the average power consumption.

7.6.4 Power Consumption Evaluation

Power and energy consumption are among the most important evaluation metrics for the HAR engine. Therefore, we perform a detailed analysis of the power consumption of each major component in the proposed design. Our analysis assumes 100 kHz operating frequency, which provides sufficient performance (4.19 ms operation per activity in the baseline HAR engine), as described in Section 7.6.5. To analyze the power consumption, we first generate the switching activity data of the design using sensor data from our user studies. The switching activity file is then used to calculate the power consumption of the major blocks in the design. The power consumption values reported in this section include savings achieved by the clock gating described in Section 7.5. We estimate up to 30% additional savings by power gating the feature generation and DNN blocks using two power domains until a new activity window is detected.

7.6.4.1 Baseline HAR Engine

Table 7.7 summarizes the average power consumption of the major blocks of the baseline HAR engine. Power consumption is divided into two distinct parts based on the operating mode of each block. The first row shows the power consumption of the always-on preprocessing and FIFO blocks. The majority of the power in the preprocessing block is attributed to active power. Since this is mainly due to the FIFO memories in the preprocessing block, this power consumption can be further reduced by optimizing the memory design. The remaining rows

Table 7.7: Power consumption summary for the baseline HAR engine at $f = 100\text{kHz}$, $V = 1.0\text{ V}$

Operation Mode	Block	Dynamic Power (μW)	Leakage Power (μW)	Total Power (μW)
Data Collection & Preprocessing (always ON)	Filtering	0.49	0.05	0.54
	Segmentation	0.43	0.05	0.48
	FIFO	12.10	1.16	13.26
Classification (once per activity)	Downsample	4.39	0.75	5.14
	DWT	2.49	0.42	2.91
	FFT	11.90	2.61	14.51
	DNN Feature Merge	1.77	0.75	2.52
	DNN	10.14	1.16	11.30
Total	Data Collection	13.02	1.26	14.28
	Classification	30.69	5.69	36.38
	Overall	43.71	6.95	50.66

of the table show the power consumption of blocks that are executed only once per activity. Of these blocks, the FFT block has the highest power consumption as expected. This aligns with our earlier observation that the FFT block has the highest area among all the blocks. The DNN block has a higher active power consumption due to the multiplications involved in each neuron. The total power consumption for classifying the activity after the preprocessing blocks complete is about $36.4\ \mu\text{W}$. This represents about $325\times$ reduction in processing power when compared to implementations on a programmable microcontroller.

7.6.4.2 Activity-Aware 2-Level HAR Engine

This section analyzes the power consumption of the activity-aware 2-level HAR engine. The power values for the always-on blocks remain unchanged since they are reused. The power consumption breakdown of the remaining blocks is summarized in Table 7.8.

As described in Section 7.4, the power consumption of this design depends on the activity,

Table 7.8: Power consumption summary for the activity-aware HAR engine at $f = 100$ kHz and $V = 1.0$ V

Operation Mode	Block	Dynamic Power (μ W)	Leakage Power (μ W)	Total Power (μ W)		
Classification (once per activity)	Common	Downsample	3.33	0.84	4.17	
		SVM Feature Input	0.47	0.04	0.51	
		SVM	0.47	0.15	0.62	
	Static	Decision Tree	0.24	0.02	0.26	
		Dynamic	DWT	1.96	0.43	2.39
			FFT	9.64	2.66	12.3
DNN Feature Input			1.42	0.75	2.17	
DNN	7.60		0.85	8.45		
Total	Static (Data Collection + Classification)	17.24 (12.73+4.51)	2.29 (1.24+1.05)	19.53 (13.97+5.56)		
	Dynamic (Data Collection + Classification)	37.62 (12.73+24.89)	6.96 (1.24+5.72)	44.58 (13.97+30.61)		

i.e., whether the activity is static or dynamic. The first part of the table shows the power consumption of the blocks that are common to both static and dynamic activities. These blocks include Downsample and the SVM classifier. We observe that the SVM classifier has a negligible power consumption of 0.62μ W, while the Downsample block has similar power consumption as the baseline HAR engine. The “Static” part of the table shows the blocks that are used when a static activity is detected by the SVM classifier. Since the decision tree for static activities employs the same features as the SVM classifier, it is the only component contributing to the power consumption, i.e., there is no need to generate other features. As we can see in Table 7.8, the DT consumes only 0.26μ W of power. Finally, the “Dynamic” part of the table shows the power consumption of the engine when a dynamic activity is identified by the SVM classifier. The dynamic part includes computation of the detailed features required for the neural network and the neural network execution itself. As expected, the total power for dynamic activities is higher than the power consumption for static activities.

Several blocks, such as Downsample, have lower power consumption when used in the

activity-aware 2-level HAR engine compared to the baseline design. This difference stems from two reasons. First, these two designs are synthesized separately as a whole, as opposed to integrating blocks as black boxes. Hence, the synthesized block, their output ports, and lengths of the buses they drive have differences. For example, the output of the Downsample block connects to the SVM block in the 2-level HAR engine. In contrast, it connects to all the feature generation blocks, including FFT and DWT, in the baseline design. Second, the power consumption is found using the VCD files in Primetime tool. Since most of the blocks in the 2-level activity-aware design are activated less frequently, they end up having a smaller switching factor, hence, lower power consumption.

In summary, the activity-aware classifier consumes 5.56 μW for static activities and 30.61 μW for dynamic activities once a segment is identified. Including the power consumption of the sensor and data communication, this leads to 1.3 mW, which is 10 times lower than the embedded system solutions.

Voltage Scaling: Once an activity window is detected, the proposed HAR engine takes 421 cycles to generate the features and classify the activity. Hence, operating at as low as 42 kHz provides around 10 ms processing window, which is comparable to the sampling time of the sensors. This slack enables us to lower the supply voltage without a noticeable difference in

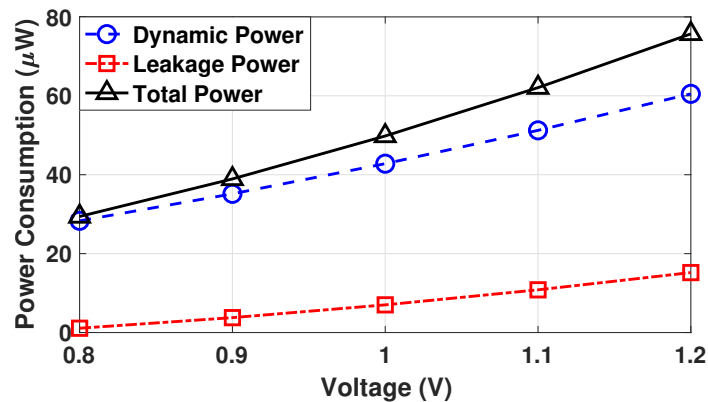


Figure 7.10: Power consumption as a function of voltage

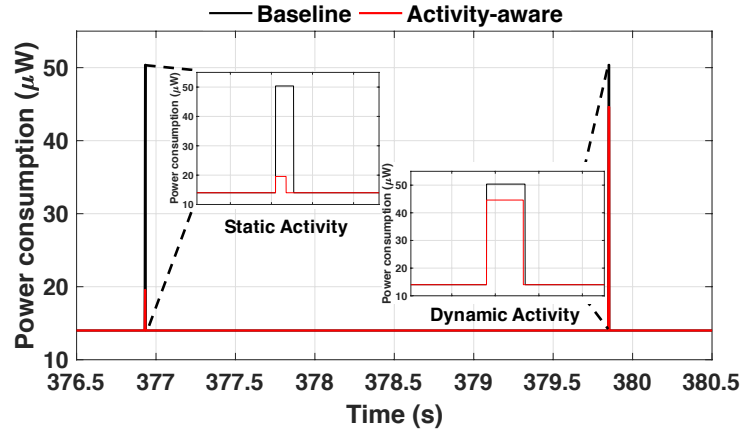


Figure 7.11: Comparison of power consumption of the baseline and activity-aware HAR Engines

performance. We exploited this observation by sweeping the supply voltage and measuring the resulting power consumption. The power consumption at 1.2 V is 75 μW , as shown in Figure 7.10. Operating at 1.0 V and 0.8 V decrease the power consumption to 45 μW and 30 μW , respectively.

Peak Power Consumption Evaluation: Our goal is to operate the HAR engine using ambient energy harvesting. Therefore, it is also important to evaluate the peak power consumption of the design to ensure that the power consumption is within the capabilities of energy-harvesting technologies. Figure 7.11 compares the total power consumption of the two classifiers for the two types of activity windows, static and dynamic. Specifically, the figure shows a static activity at $t = 376.9$ s and a dynamic activity at $t = 379.8$ s. The black line shows the power consumption of the single-level baseline HAR engine. It has the same instantaneous power consumption for both activity types. Furthermore, it has a peak power consumption of almost 51 μW . This can be limiting for energy-harvesting devices since they have to support a peak power of 51 μW for each activity window. In contrast to this, the activity-aware classifier, shown with a red line, has a peak power consumption of only 19.5 μW for static activities,

as seen in the zoomed-in portion inside Figure 7.11. Therefore, the peak power that energy-harvesting devices need to provide is reduced significantly, especially for static activities.

7.6.5 Performance Evaluations

This section presents the execution time of each block in the proposed HAR engines. The time scales of human activities and sampling data rates are much smaller than the maximum operating frequency we can achieve with the commercial TSMC 65 nm technology. In contrast, power consumption is of extreme importance. Both the baseline and activity-aware 2-level HAR engines can operate comfortably at 1 MHz, which is much higher than required. Therefore, we summarize the number of cycles taken by each major block in the rest of this section.

The always-on preprocessing and segmentation blocks take one cycle for processing each incoming sample. Once the segmentation block marks a segment, the corresponding blocks for each classifier are activated. The latencies for these blocks are summarized in Table 7.9. We divide the table into three parts. The first part, denoted by “Common” includes the blocks that are common to both baseline and activity-aware engines. The second part, denoted by “Baseline Classifier” includes the blocks used only in the baseline classifier. Finally, the third part of the table lists the latencies for blocks that are exclusive to the 2-level classifier.

The Downsample block for the accelerometer data has the highest latency while the DWT block has the lowest latency. The Downsample block incurs a higher latency because it has to go through all the samples in a segment serially before generating its output. Moreover, the calculation of statistical features is integrated into the Downsample block, which adds a few cycles of additional execution. We also observe that the Downsample block for the stretch sensor takes fewer cycles to execute. This can be attributed to the fact that the sampling rate of the stretch sensor is lower than that of the accelerometer. Once the Downsample blocks

Table 7.9: Summary of latency of the blocks in the baseline HAR engine and the activity-aware HAR engine

Usage	Block	Latency (cycles)
Common	Acc. Downsample	252
	Stretch Downsample	109
	FFT	21
	DWT	4
Baseline HAR engine	3 Layer DNN	163
Activity-aware HAR engine	SVM	3
	Decision Tree	2
	2 Layer DNN	145

complete their execution, the DWT and FFT blocks execute in parallel leading to their low execution latency of 4 and 21 cycles, respectively. The FFT and DWT blocks are executed for the baseline HAR engine independent of the class of activity, whereas for the activity-aware engine, they are executed for dynamic activities only.

The last step in the baseline HAR engine is to evaluate the activity classification using the three-layer DNN, which takes 163 cycles. For the activity-aware engine, we first evaluate the SVM classifier to distinguish between static and dynamic activities. If the activity is static, we execute the decision tree with a latency of 2 cycles. Otherwise, we execute the FFT, DWT, and the 2-layer DNN, which take 166 cycles in total.

The total latency is not simply summing every block up since there are some overlaps. In summary, including data collection blocks, the total latency for the baseline HAR engine is 419 cycles, which results in 4.19 ms for the classifying the activity at 100 kHz operating frequency. Total latencies for dynamic activity and static activity in the activity-aware HAR engine are 421 cycles (4.21 ms @100 kHz) and 257 cycles (2.57 ms @100 kHz), respectively. Consequently, the time taken for activity recognition is negligible compared to the time required for data collection (in the order of a few seconds).

7.7 Summary

This chapter presented the first hardware accelerator that integrates all aspects of human activity recognition. We first designed a baseline HAR engine following the most commonly used single classifier for all the activities. Then, we exploited the nature of human activities to design an activity-aware 2-level HAR engine that identifies activities in two steps. This approach improves the classification accuracy while reducing the power consumption of the HAR engine. We implemented both HAR engines using a commercial 65 nm process technology. Our extensive post-layout simulations show that the proposed HAR engines are able to reduce the power consumption by about two orders of magnitude compared with conventional programmable solutions.

Further optimization in power consumption and area can be obtained by using an 8-bit quantization for the DNN. Our preliminary results show that it can achieve up to 7% savings in power consumption and a 6% reduction in area. However, 8-bit quantization also reduces the robustness of classification accuracy. Therefore, our future work will consider quantization-aware training approaches to achieve lower power and smaller area.

CONCLUSION AND FUTURE DIRECTIONS

Wearable devices offer great potential to change the landscape of health and activity monitoring. However, their widespread adoption has been hindered by various adaptation and technical challenges. This dissertation presented potential solutions to these challenges.

First, we presented an open-source hardware/software platform for health monitoring. As part of this, we designed a wearable device using flexible hybrid electronics. We released the hardware files, software libraries, and applications as part of the open-source release. The proposed platform will help in creating a common community of researchers in health monitoring.

Our second contribution presented an algorithm for near-optimal energy allocation in energy harvesting wearable devices. The proposed algorithm used dynamic programming to enable recharge-free operations. Experiments with real solar energy harvesting data on our wearable device showed that the proposed algorithm achieves results that are within 3% of the optimal result obtained offline.

Third, we presented a comprehensive framework for HAR. We started with the w-HAR dataset that includes data of seven activities and transitions for 22 users. w-HAR is the first dataset that includes data from wearable stretch sensors and accelerometers. Then, we proposed an online learning approach for HAR. Most of the approaches for HAR use offline learning on smartphones or wearable devices. However, offline learning does not scale well when the models are used on new users. Our approach addressed the limitation by using feedback from users to update the classifier weights. Experimental evaluations with 22 users showed that the proposed approach improves the accuracy for new users by as much as 40% while consuming about 12.5 mW power. One of the critical aspects of online learning is determining the number

of layers to reuse for new users. To this end, we proposed a transfer learning framework to determine the number of neural network layers to transfer such that learning for new users is optimized. We used representational analysis to show that the initial layers of a CNN provide general features while deeper layers provide user-specific information. Using this insight, we transferred the initial layers of CNNs to new users and fine-tuned *only* the deeper layers of the network. Evaluations using three datasets showed that transfer learning achieves up to 43% accuracy improvement when compared to accuracy without using transfer learning.

Our next contribution integrated the energy allocation and HAR applications in a runtime energy-accuracy optimization framework for energy harvesting IoT devices. This is important since a single operating point of an application may not be suitable for every energy budget in energy harvesting IoT devices. Therefore, we designed a total of 22 design points for HAR, which is the driver application for the algorithm. Among these, we chose five Pareto-optimal design points with varying energy-accuracy trade-off. The runtime algorithm dynamically chooses among the Pareto-optimal points design points to co-optimize the accuracy and active time under energy budget constraints. Experiments using our wearable device showed that the proposed algorithm achieves 46% higher expected accuracy and 66% longer active time compared to a static design point with the highest accuracy. This algorithm will aid in achieving recharge-free operation without sacrificing the quality of service to users.

Finally, user surveys have shown that the charging requirement of wearable devices is one of the leading reasons for abandoning them. Hence, practical solutions must offer ultra-low power capabilities that enable operation on harvested energy. To address this need for HAR, we presented *the first fully integrated custom hardware accelerator* (HAR engine) that consumes 22.4 μJ per operation using a commercial 65 nm technology. We presented a complete solution that integrates *all steps of HAR*, i.e., reading the raw sensor data, generating features, and activity classification using a deep neural network (DNN). It achieves 95% accuracy in

recognizing seven activities and transitions while providing three orders of magnitude higher energy efficiency compared to embedded solutions.

In summary, this dissertation made the following contributions towards the wider adoption of wearable devices for health monitoring.

- Wearable IoT devices using flexible hybrid electronics [16, 21, 26],
- Energy-neutral operation through optimal energy harvesting and management [23],
- Online learning framework and open-source dataset for human activity recognition [15],
- A transfer learning framework for human activity recognition,
- Runtime energy-accuracy co-optimization for energy harvesting IoT devices [14],
- An ultra-low-energy hardware accelerator for human activity recognition [25]

8.1 Future Directions

This section provides some future directions for research presented in this dissertation.

Healthcare Applications for Movement Disorders: We presented an extensive HAR framework, which is one of the first steps in the treatment of movement disorders. Next, we need to extend the framework to movement disorder applications such as evaluating the effectiveness of drug therapies, freezing of gait prediction, and tremor analysis in movement disorder patients. The OpenHealth framework can enable these applications by adding sensors and processors needed for movement disorders.

System-on-a-Chip for Wearable Applications: The HAR engine presented in this framework is one of the components of a complete System-on-a-Chip (SoC) for wearable applications. The SoC will integrate general-purpose cores and other commonly used input/output interfaces. It will also integrate special-purpose engines for blocks such as pre-processing of sensor data, feature generation, reconfigurable neural network, and other classifiers.

Improved Energy-Allocation Algorithm Using Approximate Dynamic Programming: The results of the algorithm proposed in [23] degrade as the peak-to-peak variation in the harvested energy and deviation from the expected values increase. Furthermore, the algorithm in [23] does not take into account the expected energy harvest in the next few control intervals when performing the energy allocation. Instead, it uses an aggregate over a finite horizon to make the decisions. Due to this, it experiences large deviations from the optimal allocation when the expected energy harvest is far in the future, as seen in Figure 3.6. Our future work plans to address this limitation by performing a finite look-ahead while making the energy allocation in a given interval. We will use rollout, Monte Carlo search, and approximate dynamic programming to improve the energy allocation. Moreover, we plan to develop stochastic models for the expected energy harvest to ensure that short-term variations in the ambient sources of energy are taken into account. The proposed enhancements will allow us to achieve energy-neutral operation of wearable devices with minimal impact on the quality of service to the users.

Skin Temperature Management for Wearable Devices: Skin temperature issues will become more prominent in wearable devices as we move towards in-body and implantable devices. It is critical to effectively manage the skin temperature of these devices since they are in direct contact with the user. Therefore, one direction of future work can leverage state-of-the-art thermal management in mobile systems to develop novel algorithms for skin temperature management in in-body and implantable devices.

REFERENCES

- [1] Abadi, M. *et al.*, “TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems”, [Online] <http://tensorflow.org/>, accessed 31 July 2019 (2015).
- [2] Akbari, A. and R. Jafari, “Transferring Activity Recognition Models for New Wearable Sensors With Deep Generative Domain Adaptation”, in “Proceedings of the 18th International Conference on Information Processing in Sensor Networks”, pp. 85–96 (2019).
- [3] Alippi, C. and C. Galperti, “An Adaptive System for Optimal Solar Energy Harvesting in Wireless Sensor Network Nodes”, *IEEE Trans. Circuits Syst. I, Reg. Papers* **55**, 6, 1742–1750 (2008).
- [4] Andreas, A. and T. Stoffel, “NREL Solar Radiation Research Laboratory (SRRL): Baseline Measurement System (BMS); Golden, Colorado (Data); NREL Report No. DA-5500-56488”, <http://dx.doi.org/10.5439/1052221>, accessed 5 August 2017 (1981).
- [5] Anguita, D., A. Ghio, L. Oneto, F. X. Llanas Parra and J. L. Reyes Ortiz, “Energy Efficient Smartphone-Based Activity Recognition Using Fixed-Point Arithmetic”, *J. of Universal Comput. Sci.* **19**, 9, 1295–1314 (2013).
- [6] Anguita, D., A. Ghio, L. Oneto, X. Parra and J. L. Reyes-Ortiz, “A Public Domain Dataset for Human Activity Recognition using Smartphones”, in “ESANN”, (2013).
- [7] Argyriou, A., T. Evgeniou and M. Pontil, “Multi-Task Feature Learning”, in “Advances in neural information processing systems”, pp. 41–48 (2007).
- [8] Arif, M., M. Bilal, A. Kattan and S. I. Ahamed, “Better Physical Activity Classification Using Smartphone Acceleration Sensor”, *J. of Med. Syst.* **38**, 9, 95 (2014).
- [9] Banaee, H., M. U. Ahmed and A. Loutfi, “Data Mining for Wearable Sensors in Health Monitoring Systems: A Review of Recent Trends and Challenges”, *Sensors* **13**, 12, 17472–17500 (2013).
- [10] Bao, L. and S. S. Intille, “Activity Recognition From User-Annotated Acceleration Data”, in “Int. Conf. on Pervasive Comput.”, pp. 1–17 (2004).
- [11] Bertsekas, D., A. Nedic and A. Ozdaglar, *Convex Analysis and Optimization* (Athena Scientific Belmont, MA, 2003).
- [12] Bertsekas, D. P., *Dynamic Programming and Optimal Control*, vol. 1 (Athena Scientific Belmont, MA, 1995).
- [13] Bhardwaj, K., N. Suda and R. Marculescu, “EdgeAI: A Vision for Deep Learning in IoT Era”, *IEEE Design & Test* (2019).

- [14] Bhat, G., K. Bagewadi, H. G. Lee and U. Y. Ogras, “REAP: Runtime Energy-Accuracy Optimization for Energy Harvesting IoT Devices”, in “Proc. of Annual Design Autom. Conf.”, pp. 171:1–171:6 (2019), DOI:<https://doi.org/10.1145/3316781.3317892>.
- [15] Bhat, G., R. Deb, V. V. Chaurasia, H. Shill and U. Y. Ogras, “Online Human Activity Recognition using Low-Power Wearable Devices”, in “Proc. of Int. Conf. on Comput. Aided Design”, pp. 72:1–72:8 (2018), DOI:<https://doi.org/10.1145/3240765.3240833>.
- [16] Bhat, G., R. Deb and U. Y. Ogras, “OpenHealth: Open Source Platform for Wearable Health Monitoring”, IEEE Design & Test **36**, 5, 27–34, © 2019 IEEE. Reprinted with permission from authors. (2019).
- [17] Bhat, G., H. Gao, S. K. Mandal, U. Y. Ogras and S. Ozev, “Determining Mechanical Stress Testing Parameters for FHE Designs with Low Computational Overhead”, IEEE Design & Test (2020).
- [18] Bhat, G., S. Gumussoy and U. Y. Ogras, “Power-Temperature Stability and Safety Analysis for Multiprocessor Systems”, ACM Trans. Embed. Comput. Syst. **16**, 5s, 145:1–145:19 (2017).
- [19] Bhat, G., S. Gumussoy and U. Y. Ogras, “Power and Thermal Analysis of Commercial Mobile Platforms: Experiments and Case Studies”, in “2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)”, pp. 144–149 (2019).
- [20] Bhat, G., S. Gumussoy and U. Y. Ogras, “Analysis and Control of Power-Temperature Dynamics in Heterogeneous Multiprocessors”, IEEE Transactions on Control Systems Technology (2020).
- [21] Bhat, G., U. Gupta, N. Tran, J. Park, S. Ozev and U. Y. Ogras, “Multi-Objective Design Optimization for Flexible Hybrid Electronics”, in “Proc. of Int. Conf. on Comput.-Aided Design”, pp. 1–6 (2016).
- [22] Bhat, G., S. K. Mandal, U. Gupta and U. Y. Ogras, “Online Learning for Adaptive Optimization of Heterogeneous SoCs”, in “Proceedings of the International Conference on Computer-Aided Design”, pp. 1–6 (2018).
- [23] Bhat, G., J. Park and U. Y. Ogras, “Near-Optimal Energy Allocation for Self-Powered Wearable Systems”, in “Proc. Int. Conf. on Comput.-Aided Design”, pp. 368–375 (2017), © 2017 IEEE. Reprinted with permission from authors.
- [24] Bhat, G., G. Singla, A. K. Unver and U. Y. Ogras, “Algorithmic Optimization of Thermal and Power Management for Heterogeneous Mobile Platforms”, IEEE Trans. Very Large Scale Integr. (VLSI) Syst. **26**, 3, 544–557 (2018).

- [25] Bhat, G., Y. Tuncel, S. An, H. G. Lee and U. Y. Ogras, “An Ultra-Low Energy Human Activity Recognition Accelerator for Wearable Health Applications”, *ACM Transactions on Embedded Computing Systems (TECS)* **18**, 5s, 1–22, DOI:<https://doi.org/10.1145/3358175> (2019).
- [26] Bhat, G., Y. Tuncel, S. An and U. Y. Ogras, “Wearable IoT Devices for Health Monitoring”, in “TechConnect Briefs 2019”, pp. 357–360 (2019).
- [27] Blanke, U. and B. Schiele, “Remember and Transfer What You Have Learned-Recognizing Composite Activities Based on Activity Spotting”, in “International Symposium on Wearable Computers (ISWC) 2010”, pp. 1–8 (2010).
- [28] Blitzer, J., R. McDonald and F. Pereira, “Domain Adaptation With Structural Correspondence Learning”, in “Proceedings of the 2006 conference on empirical methods in natural language processing”, pp. 120–128 (2006).
- [29] Bonomi, A. G., A. H. Goris, B. Yin and K. R. Westerterp, “Detection of Type, Duration, and Intensity of Physical Activity Using an Accelerometer”, *Medicine & Science in Sports & Exercise* **41**, 9, 1770–1777 (2009).
- [30] Bort-Roig, J., N. D. Gilson, A. Puig-Ribera, R. S. Contreras and S. G. Trost, “Measuring and Influencing Physical Activity With Smartphone Technology: A Systematic Review”, *Sports Medicine* **44**, 5, 671–686 (2014).
- [31] Boyd, S. and L. Vandenberghe, *Convex Optimization* (Cambridge Univ. Press, 2004).
- [32] Bracke, W., P. Merken, R. Puers and C. Van Hoof, “A 1 cm³ Modular Autonomous Sensor Node For Physical Activity Monitoring”, in “2006 Ph.D. Research in Microelectronics and Electronics”, pp. 429–432 (2006).
- [33] Buchli, B., F. Sutton, J. Beutel and L. Thiele, “Dynamic Power Management for Long-Term Energy Neutral Operation of Solar Energy Harvesting Systems”, in “Proc. Conf. on Embedd. Network Sensor Syst.”, pp. 31–45 (2014).
- [34] Case, M. A., H. A. Burwick, K. G. Volpp and M. S. Patel, “Accuracy of Smartphone Applications and Wearable Devices for Tracking Physical Activity Data”, *Jama* **313**, 6, 625–626 (2015).
- [35] Chen, Y. and C. Shen, “Performance Analysis of Smartphone-Sensor Behavior for Human Activity Recognition”, *IEEE Access* **5**, 3095–3110 (2017).
- [36] Chikhaoui, B., F. Gouineau and M. Sotir, “A CNN Based Transfer Learning Model for Automatic Activity Recognition From Accelerometer Sensors”, in “International Conference on Machine Learning and Data Mining in Pattern Recognition”, pp. 302–315 (2018).

- [37] Chollet, F. *et al.*, “Keras”, [Online] <https://keras.io>, accessed 31 July 2019 (2015).
- [38] Chomiak, T., A. Watts, N. Meyer, F. V. Pereira and B. Hu, “A Training Approach to Improve Stepping Automaticity While Dual-Tasking in Parkinson’s Disease: A Prospective Pilot Study”, *Medicine* **96**, 5 (2017).
- [39] Contreras, R., M. Huerta, G. Sagbay, C. LLumiguano, M. Bravo, A. Bermeo, R. Clotet and A. Soto, “Tremors Quantification in Parkinson Patients Using Smartwatches”, in “Proc. IEEE Ecuador Technical Chapters Meeting (ETCM)”, pp. 1–6 (2016).
- [40] Cook, D., K. D. Feuz and N. C. Krishnan, “Transfer Learning for Activity Recognition: A Survey”, *Knowledge and Information Systems* **36**, 3, 537–556 (2013).
- [41] Cormen, T. H., C. E. Leiserson, R. L. Rivest and C. Stein, *Introduction to Algorithms* (MIT press, 2009).
- [42] Custodio, V., F. J. Herrera, G. López and J. I. Moreno, “A Review on Architectures and Communications Technologies for Wearable Health-Monitoring Systems”, *Sensors* **12**, 10, 13907–13946 (2012).
- [43] Dai, W., Q. Yang, G.-R. Xue and Y. Yu, “Boosting for Transfer Learning”, in “Proceedings of the 24th International Conference on Machine Learning”, ICML ’07, pp. 193–200 (2007).
- [44] Daneault, J.-F., “Could Wearable and Mobile Technology Improve the Management of Essential Tremor?”, *Frontiers in neurology* **9**, 257:1–257:8 (2018).
- [45] de Lima, A. L. S. *et al.*, “Feasibility of Large-Scale Deployment of Multiple Wearable Sensors in Parkinson’s Disease”, *PLOS One* **12**, 12, e0189161 (2017).
- [46] Deb, R., *How Does Technology Development Influence the Assessment of Parkinson’s Disease? A Systematic Review*, Master’s thesis, Arizona State University (2019).
- [47] Deng, J., W. Dong, R. Socher, L.-J. Li, K. Li and L. Fei-Fei, “Imagenet: A Large-Scale Hierarchical Image Database”, in “2009 IEEE conference on computer vision and pattern recognition”, pp. 248–255 (2009).
- [48] Dimitrov, D. V., “Medical Internet of Things and Big Data in Healthcare”, *Healthcare Informatics Research* **22**, 3, 156–163 (2016).
- [49] Dinesh, K., M. Xiong, J. Adams, R. Dorsey and G. Sharma, “Signal Analysis for Detecting Motor Symptoms in Parkinson’s and Huntington’s Disease Using Multiple Body-Affixed Sensors: A Pilot Study”, in “Image and Signal Process. Workshop”, pp. 1–5 (2016).

- [50] Ding, R., X. Li, L. Nie, J. Li, X. Si, D. Chu, G. Liu and D. Zhan, “Empirical Study and Improvement on Deep Transfer Learning for Human Activity Recognition”, *Sensors* **19**, 1, 57 (2019).
- [51] DMI International Distribution Ltd, “Curved Lithium Thin Cells”, [Online] <http://www.dmi-international.com/data%20sheets/Curved%20Li%20Polymer.pdf> Accessed 04/18/2018 (2018).
- [52] Dorsey, E. R., F. P. Vlaanderen, L. J. Engelen, K. Kieburz, W. Zhu, K. M. Biglan, M. J. Faber and B. R. Bloem, “Moving Parkinson Care to the Home”, *Movement Disorders* **31**, 9, 1258–1262 (2016).
- [53] Espay, A. J. *et al.*, “Technology in Parkinson’s Disease: Challenges and Opportunities”, *Movement Disorders* **31**, 9, 1272–1282 (2016).
- [54] Estrin, D. and I. Sim, “Open mHealth Architecture: An Engine for Health Care Innovation”, *Science* **330**, 6005, 759–760 (2010).
- [55] Evgeniou, T. and M. Pontil, “Regularized Multi-Task Learning”, in “Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining”, pp. 109–117 (2004).
- [56] Feng, S. and M. F. Duarte, “Few-Shot Learning-Based Human Activity Recognition”, arXiv preprint arXiv:1903.10416 (2019).
- [57] FlexSolarCells, “SP3-37 Datasheet”, http://www.flexsolarcells.com/index_files/OEM_Components/Flex_Cells/specification_sheets/01_FlexSolarCells.com_PowerFilm_Solar_SP3-37_Specification_Sheet.pdf, accessed 5 August 2017 (2013).
- [58] Friedman, J., T. Hastie and R. Tibshirani, *The Elements of Statistical Learning*, vol. 1 (Springer, 2001).
- [59] Gao, H., G. Bhat, U. Y. Ogras and S. Ozev, “Optimized Stress Testing for Flexible Hybrid Electronics Designs”, in “2019 IEEE 37th VLSI Test Symposium (VTS)”, pp. 1–6 (2019).
- [60] Gaudette, B., V. Hanumaiah, S. Vrudhula and M. Krunz, “Optimal Range Assignment in Solar Powered Active Wireless Sensor Networks”, in “Proc. IEEE Infocom”, pp. 2354–2362 (2012).
- [61] Geisler, M., S. Boisseau, M. PEREZ, P. Gasnier, J. Willemin, I. Ait-Ali and S. Perraud, “Human-Motion Energy Harvester for Autonomous Body Area Sensors”, *Smart Materials and Structures* **557**, 1, 012024 (2017).
- [62] GMB, “031009 datasheet”, <http://www.gmbattery.com/Datasheet/LIPO/LIPO-031009-12mAh.pdf>, accessed 5 August 2017 (2009).

- [63] Grant, M. and S. Boyd, “Graph Implementations for Nonsmooth Convex Programs”, in “Recent Advances in Learning and Control”, Lecture Notes in Control and Information Sciences, pp. 95–110 (Springer, London, 2008).
- [64] Grant, M. and S. Boyd, “CVX: Matlab Software for Disciplined Convex Programming, Version 2.1”, <http://cvxr.com/cvx>, accessed 5 August 2017 (2014).
- [65] Gupta, U., J. Park, H. Joshi and U. Y. Ogras, “Flexibility-Aware System-on-Polymer (SoP): Concept to Prototype”, *IEEE Trans. Multi-Scale Comput. Syst.* **3**, 1, 36–49 (2017).
- [66] Gupta, U., C. A. Patil, G. Bhat, P. Mishra and U. Y. Ogras, “Dypo: Dynamic Pareto-Optimal Configuration Selection for Heterogeneous MpSoCs”, *ACM Transactions on Embedded Computing Systems (TECS)* **16**, 5s, 1–20 (2017).
- [67] Györfi, N., Á. Fábrián and G. Hományi, “An Activity Recognition System for Mobile Phones”, *Mobile Networks and Appl.* **14**, 1, 82–91 (2009).
- [68] Hachiya, H., M. Sugiyama and N. Ueda, “Importance-Weighted Least-Squares Probabilistic Classifier for Covariate Shift Adaptation With Application to Human Activity Recognition”, *Neurocomputing* **80**, 93–101 (2012).
- [69] Haghi, M., K. Thurow and R. Stoll, “Wearable Devices in Medical Internet of Things: Scientific Research and Commercially Available Devices”, *Healthcare Informatics Research* **23**, 1, 4–15 (2017).
- [70] Hall, M., E. Frank, G. Holmes, B. Pfahringer, P. Reutemann and I. H. Witten, “The WEKA Data Mining Software: An Update”, *ACM SIGKDD Explorations Newsletter* **11**, 1, 10–18 (2009).
- [71] Hanson, M. A., H. C. Powell Jr, A. T. Barth, K. Ringgenberg, B. H. Calhoun, J. H. Aylor and J. Lach, “Body Area Sensor Networks: Challenges And Opportunities”, *Computer* **42**, 1, 58 (2009).
- [72] Hardkernel, “ODROID-XU3”, https://wiki.odroid.com/old_product/odroid-xu3/odroid-xu3 Accessed 24 Nov. 2018 (2014).
- [73] He, Y. and Y. Li, “Physical Activity Recognition Utilizing the Built-in Kinematic Sensors of a Smartphone”, *Int. J. of Distrib. Sensor Networks* **9**, 4, 481–580 (2013).
- [74] Heldman, D. A., D. A. Harris, T. Felong, K. L. Andrzejewski, E. R. Dorsey, J. P. Giuffrida, B. Goldberg and M. A. Burack, “Telehealth Management of Parkinson’s Disease Using Wearable Sensors: an Exploratory Study”, *Digital biomarkers* **1**, 1, 43–51 (2017).
- [75] Hiremath, S., G. Yang and K. Mankodiya, “Wearable Internet of Things: Concept, Architectural Components and Promises for Person-Centered Healthcare”, in “Proc. MOBIHEALTH”, pp. 304–307 (2014).

- [76] Hoang, D. C., Y. K. Tan, H. B. Chng and S. K. Panda, “Thermal Energy Harvesting From Human Warmth for Wireless Body Area Network in Medical Healthcare System”, in “Int. Conf. on Power Electron. and Drive Syst.”, pp. 1277–1282 (2009).
- [77] Hwang, G.-T. *et al.*, “Self-Powered Cardiac Pacemaker Enabled by Flexible Single Crystalline PMN-PT Piezoelectric Energy Harvester”, *Advanced materials* **26**, 28, 4880–4887 (2014).
- [78] Ineichen, P. and R. Perez, “A New Airmass Independent Formulation for the Linke Turbidity Coefficient”, *Solar Energy* **73**, 3, 151–157 (2002).
- [79] InvenSense, “Motion Processing Unit”, <https://www.invensense.com/products/motion-tracking/9-axis/mpu-9250>, accessed 5 August 2017 (2016).
- [80] Jafari, R., W. Li, R. Bajcsy, S. Glaser and S. Sastry, “Physical Activity Monitoring for Assisted Living at Home”, in “Int. Workshop on Wearable and Implantable Body Sensor Network”, pp. 213–219 (2007).
- [81] Jayakumar, H., K. Lee, W. S. Lee, A. Raha, Y. Kim and V. Raghunathan, “Powering the Internet of Things”, in “Proc. of ISLPED”, pp. 375–380 (2014).
- [82] Johansson, D., K. Malmgren and M. Murphy, “Wearable Sensors for Clinical Applications in Epilepsy, Parkinson’s Disease, and Stroke: A Mixed-Methods Systematic Review”, *Jrnl. of neurology* pp. 1–13 (2018).
- [83] Kansal, A., J. Hsu, S. Zahedi and M. B. Srivastava, “Power Management in Energy Harvesting Sensor Networks”, *ACM Trans. Embedd. Comput. Syst.* **6**, 4, 32 (2007).
- [84] Khan, Y. *et al.*, “Flexible Hybrid Electronics: Direct Interfacing of Soft and Hard Electronics for Wearable Health Monitoring”, *Advanced Functional Materials* **26**, 47, 8764–8775 (2016).
- [85] Kim, S. J., J. H. We and B. J. Cho, “A Wearable Thermoelectric Generator Fabricated on a Glass Fabric”, *Energy & Environmental Science* **7**, 6, 1959–1965 (2014).
- [86] Kingma, D. P. and J. Ba, “Adam: A Method for Stochastic Optimization”, in “Proc. Int. Conf. on Learning Representations”, pp. 1–13 (2014).
- [87] Kirwan, M., M. J. Duncan, C. Vandelanotte and W. K. Mummery, “Using Smartphone Technology to Monitor Physical Activity in the 10,000 Steps Program: A Matched Case–Control Trial”, *J. of Med. Internet Research* **14**, 2 (2012).
- [88] Klinefelter, A. *et al.*, “21.3 A 6.45 μ W Self-Powered IoT SoC with Integrated Energy-Harvesting Power Management and ULP Asymmetric Radios”, in “Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. of Techn. Papers”, pp. 384–385 (2015).

- [89] Krishnakumar, A., G. Bhat, J. Park, H. G. Lee and U. Y. Ogras, “Sensor-classifier co-optimization for wearable human activity recognition applications”, in “2019 IEEE International Conference on Embedded Software and Systems (ICCESS)”, pp. 1–4 (2019).
- [90] Krishnamoorthi, R., “Quantizing Deep Convolutional Networks for Efficient Inference: A Whitepaper”, arXiv preprint arXiv:1806.08342 (2018).
- [91] Kuhn, H. W. and A. W. Tucker, “Nonlinear Programming”, in “Proc. of the Second Berkeley Symp. on Mathematical Statistics and Probability”, pp. 481–492 (University of California Press, 1951).
- [92] Kumar, R., J. Shin, L. Yin, J.-M. You, Y. S. Meng and J. Wang, “All-Printed, Stretchable Zn-Ag₂O Rechargeable Battery via Hyperelastic Binder for Self-Powering Wearable Electronics”, *Advanced Energy Materials* (2016).
- [93] Kwapisz, J. R., G. M. Weiss and S. A. Moore, “Activity Recognition Using Cell Phone Accelerometers”, *ACM SigKDD Explorations Newsletter* **12**, 2, 74–82 (2011).
- [94] Lagoudakis, M. G. and R. Parr, “Reinforcement Learning as Classification: Leveraging Modern Classifiers”, in “Proc. Int. Conf. Mach. Learn.”, pp. 424–431 (2003).
- [95] Lao-atiman, W., T. Julaphatachote, P. Boonmongkolras and S. Kheawhom, “Printed Transparent Thin Film Zn-MnO₂ Battery”, *J. of the Electrochemical Soc.* **164**, 4, A859–A863 (2017).
- [96] Lara, O. D. and M. A. Labrador, “A Survey on Human Activity Recognition using Wearable Sensors”, *IEEE Commun. Surveys & Tut.* **15**, 3, 1192–1209 (2013).
- [97] Latré, B., B. Braem, I. Moerman, C. Blondia and P. Demeester, “A Survey On Wireless Body Area Networks”, *Wireless Networks* **17**, 1, 1–18 (2011).
- [98] Lee, S. I., M. Y. Ozsecen, L. Della Toffola, J.-F. Daneault, A. Puiatti, S. Patel and P. Bonato, “Activity Detection in Uncontrolled Free-Living Conditions Using a Single Accelerometer”, in “2015 IEEE 12th International Conference on Wearable and Implantable Body Sensor Networks (BSN)”, pp. 1–6 (2015).
- [99] Li, B. and H. Holstein, “Perception of Human Periodic Motion in Moving Light Displays – A Motion-Based Frequency Domain Approach”, *Interdisciplinary J. of Artificial Intell. and the Simulation of Behaviour (AISBJ)* **1**, 5, 403–416 (2004).
- [100] Liang, N.-Y., G.-B. Huang, P. Saratchandran and N. Sundararajan, “A Fast and Accurate Online Sequential Learning Algorithm for Feedforward Networks”, *IEEE Trans. Neural Netw.* **17**, 6, 1411–1423 (2006).
- [101] Lin, C.-Y. and R. Marculescu, “Model Personalization for Human Activity Recognition”, in “The Fourth International Workshop on Smart Edge Computing and Networking (SmartEdge’20)”, pp. 1–4 (2020).

- [102] Lin, X., Y. Wang, N. Chang and M. Pedram, “Concurrent Task Scheduling and Dynamic Voltage and Frequency Scaling in a Real-Time Embedded System With Energy Harvesting”, *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.* **35**, 11, 1890–1902 (2016).
- [103] Liu, Q., J. Williamson, K. Li, W. Mohrman, Q. Lv, R. P. Dick and L. Shang, “Gazelle: Energy-Efficient Wearable Analysis for Running”, *IEEE Trans. Mobile Comput.* **16**, 9, 2531–2544 (2017).
- [104] Liu, X., Y. Zheng, M. W. Phyu, F. Endru, V. Navaneethan and B. Zhao, “An Ultra-Low Power ECG Acquisition And Monitoring ASIC System For WBAN Applications”, *IEEE J. on Emerg. and Sel. Topics in Circuits Syst.* **2**, 1, 60–70 (2012).
- [105] Lotfian, R. and R. Jafari, “An Ultra-Low Power Hardware Accelerator Architecture for Wearable Computers using Dynamic Time Warping”, in “Proc. Conf. on Design, Autom. and Test in Europe”, pp. 913–916 (2013).
- [106] Luo, Y., K.-H. Teng, Y. Li, W. Mao, C.-H. Heng and Y. Lian, “A $93\mu\text{W}$ 11Mbps Wireless Vital Signs Monitoring Soc With 3-Lead ECG, Bio-Impedance, And Body Temperature”, in “Proc. IEEE Asian Solid-State Circuits Conf.”, pp. 29–32 (2017).
- [107] Maetzler, W., J. Klucken and M. Horne, “A Clinical View on the Development of Technology-Based Tools in Managing Parkinson’s Disease”, *Movement Disorders* **31**, 9, 1263–1271 (2016).
- [108] Mandal, S. K., G. Bhat, J. R. Doppa, P. P. Pande and U. Y. Ogras, “An Energy-Aware Online Learning Framework for Resource Management in Heterogeneous Platforms”, *ACM Transactions on Design Automation of Electronic Systems (TODAES)* **25**, 3, 1–26 (2020).
- [109] Mandal, S. K., G. Bhat, C. A. Patil, J. R. Doppa, P. P. Pande and U. Y. Ogras, “Dynamic Resource Management of Heterogeneous Mobile Platforms via Imitation Learning”, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* **27**, 12, 2842–2854 (2019).
- [110] Matias, R., V. Paixão, R. Bouça and J. J. Ferreira, “A Perspective on Wearable Sensor Measurements and Data Science for Parkinson’s Disease”, *Frontiers in neurology* **8**, 677 (2017).
- [111] Michael J. Fox Foundation, “Parkinsons Disease Digital Biomarker DREAM Challenge”, [Online] <https://www.synapse.org/#!/Synapse:syn8717496/wiki/>. Accessed 04/15/2018 (2018).
- [112] Micucci, D., M. Mobilio and P. Napolitano, “UniMiB SHAR: A Dataset for Human Activity Recognition Using Acceleration Data from Smartphones”, *Applied Sciences* **7**, 10, 1101 (2017).

- [113] Morcos, A., M. Raghu and S. Bengio, “Insights on Representational Similarity in Neural Networks With Canonical Correlation”, in “Advances in Neural Information Processing Systems”, pp. 5732–5741 (2018).
- [114] Morillo, L., L. Gonzalez-Abril, J. Ramirez and M. de La Concepcion, “Low Energy Physical Activity Recognition System on Smartphones”, *Sensors* **15**, 3, 5163–5196 (2015).
- [115] Mosenia, A., S. Sur-Kolay, A. Raghunathan and N. K. Jha, “Wearable Medical Sensor-Based System Design: A Survey”, *IEEE Trans. Multi-Scale Comput. Syst.* **3**, 2, 124–138 (2017).
- [116] Niezen, G., P. Eslambolchilar and H. Thimbleby, “Open-Source Hardware for Medical Devices”, *BMJ Innovations* **2**, 2, 78–83 (2016).
- [117] O’Brien, B., T. Gisby and I. A. Anderson, “Stretch Sensors for Human Body Motion”, in “Proc. Electroactive Polymer Actuators and Devices”, vol. 9056, p. 905618 (2014).
- [118] Ogras, U., U. Gupta, J. Park and G. Bhat, “Designing Wearable Systems-on-Polymer using Flexible Hybrid Electronics”, in “Printed Electronics: Technologies, Applications and Challenges”, pp. 127–154 (Nova Science Publishers, Inc., 2017).
- [119] Oquab, M., L. Bottou, I. Laptev and J. Sivic, “Learning and Transferring Mid-Level Image Representations Using Convolutional Neural Networks”, in “Proceedings of the IEEE conference on computer vision and pattern recognition”, pp. 1717–1724 (2014).
- [120] Ozanne, A., D. Johansson, U. Hällgren Graneheim, K. Malmgren, F. Bergquist and M. Alt Murphy, “Wearables in Epilepsy and Parkinson’s disease—A Focus Group Study”, *Acta Neurologica Scandinavica* **137**, 2, 188–194 (2018).
- [121] Pan, S. J. and Q. Yang, “A Survey on Transfer Learning”, *IEEE Trans. Knowl. Data Eng.* **22**, 10, 1345–1359 (2010).
- [122] Park, J., G. Bhat, C. S. Geyik, H. G. Lee and U. Y. Ogras, “Optimizing Operations per Joule for Energy Harvesting IoT Devices”, Technical Report pp. 1–7 (2018).
- [123] Park, J., G. Bhat, C. S. Geyik, U. Y. Ogras and H. G. Lee, “Energy-Optimal Gesture Recognition Using Self-Powered Wearable Devices”, in “2018 IEEE Biomedical Circuits and Systems Conference (BioCAS)”, pp. 1–4 (2018).
- [124] Park, J., G. Bhat, A. Krishnakumar, C. S. Geyik, U. Y. Ogras and H. G. Lee, “Energy per Operation Optimization for Energy-Harvesting Wearable IoT Devices”, *Sensors* **20**, 3, 764 (2020).
- [125] Park, J., H. Joshi, H. G. Lee, S. Kiaei and U. Y. Ogras, “Flexible PV-cell Modeling for Energy Harvesting in Wearable IoT Applications”, *ACM Trans. Embed. Comput. Syst.* **16**, 5s, 156 (2017).

- [126] Patel, M. and J. Wang, “Applications, Challenges, and Prospective in Emerging Body Area Networking Technologies”, *IEEE Wireless Commun.* **17**, 1 (2010).
- [127] Paul, A. and D. P. Mukherjee, “Reinforced Random Forest”, in “Proc. Indian Conf. on Comput. Vision, Graphics and Image Processing”, pp. 1:1–1:8 (2016).
- [128] Pérez-López *et al.*, “Dopaminergic-Induced Dyskinesia Assessment Based on a Single Belt-Worn Accelerometer”, *Artificial Intell. in Medicine* **67**, 47–56 (2016).
- [129] Pirttikangas, S., K. Fujinami and T. Nakajima, “Feature Selection and Activity Recognition From Wearable Sensors”, in “Int. Symp. on Ubiquitous Comput. Systems”, pp. 516–527 (2006).
- [130] PowerStream, “PGEB021235 40 mAH - Rechargeable Lithium Polymer Cells”, <http://www.powerstream.com/thin-lithium-ion.htm>, accessed 16 July 2016. (2016).
- [131] Preece, S. J., J. Y. Goulermas, L. P. Kenney, D. Howard, K. Meijer and R. Crompton, “Activity Identification Using Body-Mounted Sensors—A Review of Classification Techniques”, *Physiological Measurement* **30**, 4, R1 (2009).
- [132] Pulliam, C., S. Eichenseer, C. Goetz, O. Waln, C. Hunter, J. Jankovic, D. Vaillancourt, J. Giuffrida and D. Heldman, “Continuous In-Home Monitoring of Essential Tremor”, *Parkinsonism & Related Disorders* **20**, 1, 37–40 (2014).
- [133] Qin, C.-X. and D. Qu, “Towards Understanding Attention-Based Speech Recognition Models”, *IEEE Access* **8**, 24358–24369 (2020).
- [134] Quattoni, A., M. Collins and T. Darrell, “Transfer Learning for Image Classification With Sparse Prototype Representations”, in “2008 IEEE Conference on Computer Vision and Pattern Recognition”, pp. 1–8 (2008).
- [135] Quinlan, J. R., *C4. 5: Programs for Machine Learning* (Elsevier, 2014).
- [136] Raghu, M., C. Zhang, J. Kleinberg and S. Bengio, “Transfusion: Understanding Transfer Learning for Medical Imaging”, in “Advances in Neural Information Processing Systems”, pp. 3342–3352 (2019).
- [137] Raghunathan, V., A. Kansal, J. Hsu, J. Friedman and M. Srivastava, “Design Considerations for Solar Energy Harvesting Wireless Embedded Systems”, in “Proc. of Int. Symp. on Information Processing in Sensor Networks”, p. 64 (2005).
- [138] Raina, R., A. Battle, H. Lee, B. Packer and A. Y. Ng, “Self-Taught Learning: Transfer Learning From Unlabeled Data”, in “Proceedings of the 24th international conference on Machine learning”, pp. 759–766 (2007).
- [139] Reyes-Ortiz, J.-L., L. Oneto, A. Samà, X. Parra and D. Anguita, “Transition-Aware Human Activity Recognition Using Smartphones”, *Neurocomputing* **171**, 754–767 (2016).

- [140] Roggen, D. *et al.*, “Collecting Complex Activity Datasets in Highly Rich Networked Sensor Environments”, in “Proc. Int. Conf. on Networked Sensing Syst. (INSS)”, pp. 233–240 (2010).
- [141] Rokni, S. A., M. Nourollahi and H. Ghasemzadeh, “Personalized Human Activity Recognition Using Convolutional Neural Networks”, in “Thirty-Second AAAI Conference on Artificial Intelligence”, (2018).
- [142] Saadon, S. and O. Sidek, “Micro-Electro-Mechanical System (MEMS)-Based Piezoelectric Energy Harvester for Ambient Vibrations”, *Procedia-Social and Behavioral Sci.* **195**, 2353–2362 (2015).
- [143] Saha, S. S., S. Rahman, M. J. Rasna, A. M. Islam and M. A. R. Ahad, “DU-MD: An Open-Source Human Action Dataset for Ubiquitous Wearable Sensors”, in “2018 Joint 7th International Conference on Informatics, Electronics & Vision (ICIEV) and 2018 2nd International Conference on Imaging, Vision & Pattern Recognition (icIVPR)”, pp. 567–572 (2018).
- [144] Salem, M., S. Taheri and J.-S. Yuan, “ECG Arrhythmia Classification Using Transfer Learning from 2-Dimensional Deep CNN Features”, in “2018 IEEE Biomedical Circuits and Systems Conference (BioCAS)”, pp. 1–4 (2018).
- [145] Sandia National Laboratories, “Sandia’s Ephemeris Model”, <https://pvpmc.sandia.gov/modeling-steps/1-weather-design-inputs/sun-position/sandias-code/>, accessed 5 August 2017 (2017).
- [146] Saphra, N. and A. Lopez, “Understanding Learning Dynamics Of Language Models with SVCCA”, in “Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)”, pp. 3257–3267 (2019).
- [147] Schlachetzki, J. C. *et al.*, “Wearable Sensors Objectively Measure Gait Parameters in Parkinson’s Disease”, *PloS one* **12**, 10 (2017).
- [148] Schwaighofer, A., V. Tresp and K. Yu, “Learning Gaussian Process Kernels via Hierarchical Bayes”, in “Advances in neural information processing systems”, pp. 1209–1216 (2005).
- [149] Shaikh, F. K. and S. Zeadally, “Energy Harvesting in Wireless Sensor Networks: A Comprehensive Review”, *Renew. Sust. Energ. Rev.* **55**, 1041–1054 (2016).
- [150] Shenck, N. S. and J. A. Paradiso, “Energy Scavenging With Shoe-Mounted Piezoelectrics”, *IEEE Micro* **21**, 3, 30–42 (2001).

- [151] Shin, H.-C., H. R. Roth, M. Gao, L. Lu, Z. Xu, I. Nogues, J. Yao, D. Mollura and R. M. Summers, “Deep Convolutional Neural Networks for Computer-Aided Detection: CNN Architectures, Dataset Characteristics and Transfer Learning”, *IEEE transactions on medical imaging* **35**, 5, 1285–1298 (2016).
- [152] Shoaib, M., S. Bosch, O. D. Incel, H. Scholten and P. J. Havinga, “Fusion of Smartphone Motion Sensors for Physical Activity Recognition”, *Sensors* **14**, 6, 10146–10176 (2014).
- [153] Shoaib, M., S. Bosch, O. D. Incel, H. Scholten and P. J. Havinga, “A Survey of Online Activity Recognition Using Mobile Phones”, *Sensors* **15**, 1, 2059–2085 (2015).
- [154] Sridhar, S., P. Misra, G. S. Gill and J. Warrior, “Cheepsync: A Time Synchronization Service for Resource Constrained Bluetooth LE Advertisers”, *IEEE Commun. Mag.* **54**, 1, 136–143 (2016).
- [155] Sudevalayam, S. and P. Kulkarni, “Energy Harvesting Sensor Nodes: Survey and Implications”, *IEEE Commun. Surveys & Tutorials* **13**, 3, 443–461 (2011).
- [156] Sutton, R. S. and A. G. Barto, *Introduction to Reinforcement Learning* (MIT Press, 2018), 2nd edn.
- [157] Tan, Y. K. and S. K. Panda, “Energy Harvesting From Hybrid Indoor Ambient Light and Thermal Energy Sources for Enhanced Performance of Wireless Sensor Nodes”, *IEEE Trans. on Ind. Electron.* **58**, 9, 4424–4435 (2011).
- [158] Texas Instruments, “BQ25504”, <http://www.ti.com/lit/ds/symlink/bq25504.pdf>, accessed 5 August 2017 (2015).
- [159] Texas Instruments, “TI SensorTag”, <https://store.ti.com/cc2650stk.aspx> accessed 24 Nov. 2018 (2016).
- [160] Texas Instruments Inc., “CC-2650 Microcontroller”, [Online] <http://www.ti.com/product/CC2650> Accessed 04/18/2018 (2016).
- [161] Tuncel, Y., G. Bhat and U. Y. Ogras, “Special Session: Physically Flexible Devices for Health and Activity Monitoring: Challenges from Design to Test”, in “2020 IEEE 38th VLSI Test Symposium (VTS)”, pp. 1–5 (2020).
- [162] Ugulino, W., D. Cardador, K. Vega, E. Velloso, R. Milidiú and H. Fuks, “Wearable Computing: Accelerometers’ Data Classification of Body Postures and Movements”, in “Brazilian Symposium on Artificial Intelligence”, pp. 52–61 (2012).
- [163] US Department of Labor, “American Time Use Survey”, [Online] <https://www.bls.gov/tus/>, accessed 13 July 2019 (2017).

- [164] Valenzuela, A., “Energy Harvesting for No-Power Embedded Systems”, [Online] http://focus.ti.com/graphics/mcu/ulp/energy_harvesting_embedded_systems_using_msp430.pdf, accessed 31 July 2019 (2008).
- [165] Van Dongen, S. and A. J. Enright, “Metric Distances Derived From Cosine Similarity and Pearson and Spearman Correlations”, arXiv preprint arXiv:1208.3145 (2012).
- [166] Van Helleputte, N. *et al.*, “18.3 A Multi-Parameter Signal-Acquisition Soc For Connected Personal Health Applications”, in “Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. of Techn. Papers”, pp. 314–315 (2014).
- [167] Vigorito, C. M., D. Ganesan and A. G. Barto, “Adaptive Control of Duty Cycling in Energy-Harvesting Wireless Sensor Networks”, in “Proc. of IEEE Comm. Society Conf. on Sensor, Mesh and Ad Hoc Comm. and Networks”, pp. 21–30 (2007).
- [168] Wang, A., G. Chen, J. Yang, S. Zhao and C.-Y. Chang, “A Comparative Study on Human Activity Recognition Using Inertial Sensors in a Smartphone”, IEEE Sensors J. **16**, 11, 4566–4578 (2016).
- [169] Wang, J., Y. Chen, S. Hao, X. Peng and L. Hu, “Deep Learning for Sensor-Based Activity Recognition: A Survey”, Pattern Recognition Letters **119**, 3–11 (2019).
- [170] Wendler, M., G. Hübner and I. M. Krebs, “Development of Printed Thin and Flexible Batteries”, Int. Circ. Graphic Ed. Res. **4**, 32–41 (2011).
- [171] Wong, A. C., D. McDonagh, G. Kathiresan, O. C. Omeni, O. El-Jamaly, T. C. Chan, P. Paddan and A. J. Burdett, “A 1 V, Micropower System-On-Chip For Vital-Sign Monitoring In Wireless Body Sensor Networks”, in “Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. of Techn. Papers”, pp. 138–602 (2008).
- [172] Woods, A. M., M. Nowostawski, E. A. Franz and M. Purvis, “Parkinson’s Disease and Essential Tremor Classification on Mobile Device”, Pervasive and Mobile Computing **13**, 1–12 (2014).
- [173] World Health Organization, “World Report on Disability”, [Online] http://www.who.int/disabilities/world_report/2011/report/en/. (2011).
- [174] World Health Organization, “Obesity and Overweight. Fact Sheets, 2013”, [Online] <http://www.who.int/mediacentre/factsheets/fs311/en/>. Accessed 03/22/2018 (2013).
- [175] Yang, G., W. Tan, H. Jin, T. Zhao and L. Tu, “Review Wearable Sensing System for Gait Recognition”, Cluster Computing **22**, 2, 3021–3029 (2019).
- [176] Yosinski, J., J. Clune, Y. Bengio and H. Lipson, “How Transferable Are Features in Deep Neural Networks?”, in “Advances in neural information processing systems”, pp. 3320–3328 (2014).

- [177] Zeiler, M. D., “Adadelta: An Adaptive Learning Rate Method”, arXiv preprint arXiv:1212.5701 (2012).
- [178] Zhang, B., F. Huang, J. Liu and D. Zhang, “A Novel Posture for Better Differentiation Between Parkinson’s Tremor and Essential Tremor”, *Frontiers in Neuroscience* **12** (2018).
- [179] Zhang, M. and A. A. Sawchuk, “USC-HAD: A Daily Activity Dataset for Ubiquitous Activity Recognition using Wearable Sensors”, in “Proc. of the Conf. on Ubiquitous Comput.”, pp. 1036–1043 (2012).