Predicting Sensory Data and Extending Battery Life for Wearable Devices

Songchun Fan
Duke University

Qiuyun Llull
Duke University

Benjamin C. Lee
Duke University

Telepath is a framework that supports communication-free offloading for wearable devices. With offline training, activity recognition tasks can be offloaded from the wearable to the user's phone, without transferring raw sensing data. The key observation is that when the user is carrying both devices, the sensing streams on the two devices are highly correlated. By exploiting the correlation, the phone can estimate the wearable's sensing data and emulate the watch. Our evaluations shows that with Telepath, the phone performs accurately on activity recognition tasks that are designed for smart watches, achieving on average 87% of the watch's accuracy while extending the watch's battery life by 2.1x.

1. INTRODUCTION

New generations of wearable devices enable comprehensive sensing and computing, with capabilities from simple step detection to complex gesture recognition. However, limitations in the devices' battery capacities constrain these applications' future. We find that continuously executing motion recognition on a smart watch (*e.g.*, Samsung Gear Live) drains the battery in less than six hours. Consequently, smart watch users are reluctant to use continuous sensing apps, for fear that the watch runs out of battery during the day. These apps include long-running activity recognition services such as fitness tracking, sports training, and health monitoring.

An opportunity lies in typical mobile device usage patterns — a user often carries both a wearable device and a phone simultaneously (*e.g.*, in commute). For instance, a wearable activity tracking app continuously records users' daily activities (*e.g.,* walking, cycling and running) and calculates the number of calories burned. For this app, the phone can take over the app execution and preserving the wearable's energy. Our key observation is that when the user is carrying both devices, their sensory data are highly correlated, which permits *sensory offloading, the idea of offloading not only computation but also sensing*.

Traditionally, offloading architectures shift computation from less capable devices (e.g., mobile) to more capable ones (e.g., server) that clone code and receive input data [4]. Unfortunately, continuous sensing apps cannot offload in this manner, as they process high volumes of sensory data that would require prohibitively high transmission costs. To reduce data movement, wearable devices could pre-process or compress data but doing so would consume power and offset gains from offloading.

We present Telepath, a framework for sensory offloading that transfers app processing from a wearable device to a phone without communicating raw data. Telepath allows the phone to take over sensing and computation by *predicting the wearable's sensory data*. The phone deploys an app clone that reads predicted sensory data, computes activity recognition results, and sends them to the wearable. The wearable uses them as if they had been sensed and computed locally. Telepath decides whether to offload dynamically based on real-time correlation between device data. From the user's perspective, an app employs both wearable and phone's resources transparently.

Telepath estimates data with transfer function models that statistically capture the relationship between two devices' motions. With accurate models, the wearable can offload sensing and computing to the phone with little performance loss but significant energy savings. Yet Telepath encounters challenges. One model cannot capture all the dynamics between wearable devices and phones. Raw sensory data may vary, even for the same activity, due to device placement. We address these challenges with a modeling pipeline that segments sensing data and tailors models for different user activities and device placements.

We prototype Telepath in Android Wear and evaluate it on a Samsung Gear Live watch, with popular activities and a broad spectrum of recognition algorithms. The prototype achieves real-time performance, with 87% of the watch's accuracy on average while extending the watch's battery life by 2.1x.

2. MOTIVATION

2.1 Activity Recognition on Wearables

An activity recognition app sees human motions through sensors like accelerometers, gyroscopes and magnetic field sensors. Before phones were equipped with such sensors, researchers strapped sensors and development boards to specific body parts (e.g., arms) to track motions online and recognize activities offline [8]. Those "wearable" sensors were uncomfortable, which prohibited broader, practical adoption [12].

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

HotMobile '17, February 21-22, 2017, Sonoma, CA, USA
© 2017 ACM. ISBN 978-1-4503-4907-9/17/02. . . $15.00
DOI: http://dx.doi.org/10.1145/3032970.3032971

The recent proliferation of smart phones led researchers to activity recognition algorithms that used phones' sensors [11]. A typical approach extracted features from accelerometer or gyroscope signals and trained classifiers, such as support vector machines, for a set of activities [1]. With continuous sensing and computing, phones classified activities and supplied feedback on sporting progress or health status.

Today, a new generation of technology brings activity recognition apps back to wearable devices such as sports trackers, watches, and armbands [3]. Unlike phones that might be placed in varied and obscure locations (e.g., pockets or purses), wearables are typically placed in a fixed location on the user (e.g., left wrist). This unique property makes wearables better candidates for activity recognition, especially those involving hands [6]. Moreover, recent smart watches appeal to app developers by providing rich sets of sensors, capable systems-on-chip, and most importantly, APIs that are compatible with those on phones.

2.2 Energy Consumption

Today's wearable devices are equipped with computational capabilities like those in phones — a 2016 watch may have more processor cores than a 2014 phone. However, due to its small size, an average wearable device has only 11%[1] of a phone's battery capacity. This short battery life hinders the usage of long-running activity recognition apps.

An activity recognition app consumes energy in three actions: (1) reading data from sensors, (2) processing raw sensory data and computing features that combine multiple sensor measurements, and (3) recognizing activities from extracted features using classifiers. For such an app, traditional approaches that offload computation are unhelpful because wearables dissipate most of their power in sensing not computation (i.e., action (1), not (2) and (3)).

Continuous sensing is expensive because the raw sensing data must be transported from the sensor to the system-on-chip, requiring the CPU to be awake. We find that a Samsung Gear Live watch dissipates 200.5mW when using the accelerometer. At this rate, a fully charged watch battery is estimated to last 5.7 hours. To reduce the energy cost of activity recognition apps, Telepath exploits correlated sensory data across multiple devices to "offload" both sensing and computation, while eliminating communication cost between devices.

2.3 Natural Questions

Do I still need a watch if everything is performed on the phone? Telepath is not a solution that replaces the watch. Rather, it encourages wearable/watch usage by improving the watch's battery life. It aggressively exploits the strong correlation between two devices and uses the phone as a range extender for apps such as walking, biking, running activity recognitions. Saved energy can enable more watch-specific apps, such as hand gesture recognition.

Why not directly run the phone's version of the app?

[1]Based on our survey of 34 mobile and wearable devices from past three years.

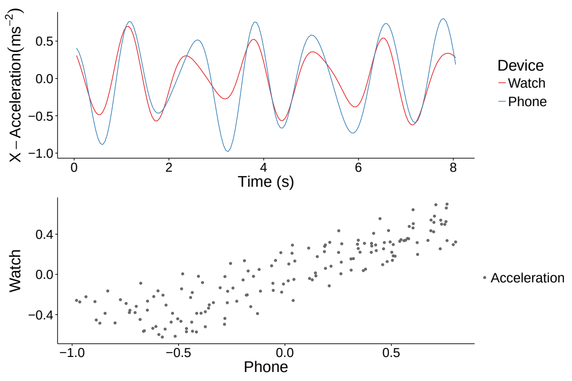

Figure 1: Sensory correlation between devices of a walking user.

Activity recognition apps rely on carefully calibrated models. Our framework allows watch developers to easily port an existing model onto Telepath, without spending time and effort to train additional models for various types of phones. Moreover, a Telepath sensing app dynamically reads data from local and remote sensors, and users need not manually switch between the phone and watch app at runtime.

Does Telepath decrease the phone's battery life? Indeed, the phone sacrifices some battery life as it helps the watch sense and compute. However, phones have large battery capacities and are more convenient to charge during the day. Essentially, Telepath reduces the huge energy gap between the phone and the watch. Our evaluation shows that Telepath increases the watch's battery life from 4.8 hours to 10.0 hours, which is still much shorter than the phone's 17-hour battery life when running Telepath.

3. TELEPATH OVERVIEW

The key observation behind Telepath is the correlation between the wearable's and the phone's sensing streams. Figure 1 presents the de-noised sensory data from a walking user who wears a Samsung Gear Live watch on her left wrist and places a Samsung Galaxy Nexus phone in her right pocket. When walking, her feet step while her hands swing. Thus, leg and arm motions are coordinated. In the top figure, a valley in the watch's x-axis acceleration always matches a valley in the phone's, albeit with slight, consistent time shifts. The scatterplot indicates the two streams are highly correlated. Telepath builds on top of this correlation.

Workflow. Telepath is an application-layer shim that lies between the app and OS. A Telepath instance has two modules – TelepathWear and TelepathPhone – running on the respective devices. Figure 2 shows the workflow. In local mode, TelepathWear interfaces to the wearable's OS, reading sensory data from local drivers and classifying features for activity recognition. In remote mode, TelepathWear neither senses nor computes. Instead, TelepathPhone reads sensory data from the phone, predicts sensory data of the watch, recognizes activity using cloned computation, and sends results back to TelepathWear for UI. Periodically, Telepath verifies device correlation and halts remote execution when correlation is weak.

On the wearable's side, the app interfaces to Telepath Re-

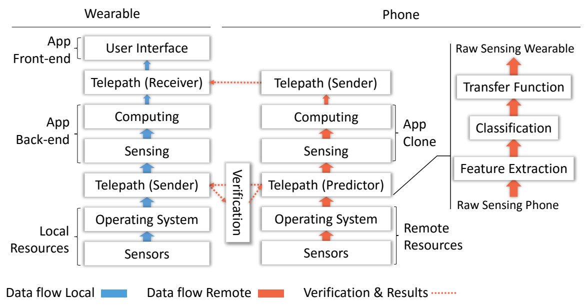

Figure 2: Telepath Workflow. The blue arrows show data flow in local execution. The red arrows show data flow in remote execution.

ceiver, which receives results computed either locally on the wearable or remotely on the phone. The local app back-end includes computing and sensing. It receives data from Telepath Sender, which resides atop OS and requests offloading from the phone. When the offloading is granted, the Sender deactivates the local sensors. Otherwise, it receives sensing data from the local sensors and passes it to the upper layer.

On the phone's side, a Telepath Sender sends computed results to the wearable's Telepath Receiver. Results are produced by a clone of wearable app's back-end. A Telepath Predictor receives sensing data from the phone's local sensors, predicts corresponding sensor readings on the watch, and sends them to the cloned app for computation. The Predictor has three sub-modules — Feature-Extraction, Classification, and Transfer-Function — which we detail in Section 4.

4. PREDICTOR DESIGN

The key component in Telepath is the predictor. Offline, raw data from both devices are collected, processed, and clustered to develop a phone-to-wearable predictor. Online, the phone's sensing data is fetched, processed, and classified to invoke the predictor that predicts the wearable's sensing data.

4.1 Offline Training

Periodically, the phone and the wearable read and upload their sensing data to the cloud for offline training. The key training technique, transfer function modeling, is an autoregressive model that captures the relationship between two time series [7].

4.1.1 Transfer Function Models (TFM)

A classic example of TFM is the relationship between advertising costs and sales. Let A_t and S_t be random variables that denote advertising and sales at time t. The general form of a transfer function is:

$$S_t = v_0 A_t + v_1 A_{t-1} + v_2 A_{t-2} + \cdots + N_t.$$

A_{t-1} is advertising cost at time $t-1$ and v_1 is the advertisement's effect after one time period. N_t is the sum of effects from all factors other than advertising and should be independent of A_t. The function is often rewritten as

$$S_t = (v_0 + v_1 B + v_2 B^2 + \cdots)A_t + N_t = v(B)A_t + N_t,$$

in which B is a backshift operator defined as

$$BA_t = A_{t-1} \quad \text{or} \quad B^m A_t = A_{t-m},$$

and $v(B)$ is a transfer function. The output time series S_t can be predicted from the current observation of A_t as well as the history of the input time series A_t. Given the nature of human motion, we can assume that our input and output time series are bounded, leading to stable transfer functions.

Offline training identifies polynomial $v(B)$ and noise N_t. First, based on training data A_t and S_t, the impulse response weights of the transfer function, $v_0, v_1, \cdots$, are initialized from the correlation coefficients of the two time series. Then, N_t is checked to determine whether it is white noise.

If N_t is not noise and exhibits some relationship with the time series, there is still information left to be captured by the transfer function. In this case, the coefficients of $v(B)$ are revised with maximum likelihood estimation. Training proceeds iteratively until the transfer function captures much of the variance in the time series and N_t resembles white noise. Online prediction estimates S_t from A_t given $v(B)$ and the forecast of N_t.

To train valid transfer function models, the time series must be stationary, which means their statistical properties (*e.g.*, mean, variance, autocorrelation) do not change over time.[2] Unfortunately, we observe that when users perform different motions, the sensing data presents distinct statistical properties. However, each independent activity consists of regular and repetitive motions that produce stationary statistical properties. This finding implies that we must segment our data such that stationarity can be guaranteed within each segment of the time series. We detail the procedure below.

4.1.2 Training Pipeline

Telepath's offline training contains three major steps to extract activity features and improve prediction accuracy. These steps can be parallelized to reduce the training time.

Segmentation - Feature Extraction. We extract features that commonly represent motion in time and frequency domains (see Table 1). Features are extracted within sampling windows. Each window has a width of 128 data points (i.e., a duration of 2.56 seconds with the sampling interpolated to 50Hz), overlapping with each other by 50%. The configuration matches the speed of human movement [2].

Segmentation - Clustering. To segment sensing streams collected from two devices, we classify data points into clusters. Each cluster presents a set of motions, during which the two devices share a unique correlation that differs from those in other clusters. However, the clusters need not have semantic meaning. We use unsupervised clustering, *k-means*, to cluster values for extracted features into $k = 10$ clusters. The number of clusters represents not only different motions, but the same motion with different device placements. Each cluster produces a stationary time series.

Transfer Function Modeling. Finally, we identify one transfer function for each cluster using the algorithm explained

[2]Strictly speaking, this is the definition of weak stationarity but it suffices for our purposes.

ID	Description
1	Standard deviation of x-axis acceleration
2	Standard deviation of y-axis acceleration
3	Standard deviation of z-axis acceleration
4-6	Median of absolute values of i-axis, $i \in \{x, y, z\}$
7-9	Interquartile Range of i-axis
10-12	Coefficients of auto-regression of i-axis
13-15	Largest frequency component in the spectrum of i
16-18	Weighted frequency average of i
19-21	Spectrum skewness of i

Table 1: Selected features for clustering.

earlier. The transfer function is re-trained if clusters change. The training dataset should be large, diverse, and adaptive to capture the most recent relationships between sensors on two devices. When their batteries permit (e.g., charged to more than 80% of capacity), wearables and the phone record paired time series that contain raw sensory data. When charging, the phone sends this data to servers that train transfer functions. Periodically, the phone downloads updated functions and organizes them according to clusters' centroids.

4.2 Online Prediction

4.2.1 *Verification of Model Accuracy*

The online process predicts the raw sensing data on the wearable from the raw sensing data on the phone. In practice, the user may not be carrying both devices at the same time, precluding the use of Telepath. Therefore, the online process must include a verification stage, assuring that the current placement of the devices are suitable for data prediction.

During offloading, a small set of sensing data is periodically collected and sent from the wearable to the phone. If data streams from the two devices do not strongly correlate, Telepath informs the wearable that offloading is canceled and the wearable proceeds with local execution mode.

We use Kendall's rank correlation coefficient to measure the extent that ranks of the two series match each other. To quantify the correlation, the two series are filtered to remove noise and transformed into rankings. Then, the Kendall Tau-b coefficient and a corresponding p-value is computed on a two-sided test of the null hypothesis, *H0: the two time series are not correlated*.

The p-value is then compared against a 0.01 significance level. If $p \geq 0.01$, we do not have sufficient confidence that the phone's data can predict the wearable's. In such cases, the phone informs the wearable of its limitations and the wearable cancels the request for offloading.

4.2.2 *Prediction Pipeline*

When the phone accepts a request for offloading, it predicts the wearable's sensing data in several steps.

1. During feature extraction, the phone's real-time sensing data enters a buffer of width 2.56s and overlap 50% (to match the sampling window during training). When the buffer is full, its data is processed to produce features (see Table 1).

2. During classification, because multiple transfer functions are generated for different clusters of motion, the prediction process must identify the cluster corresponding to the measured motion. Extracted features are classified to a cluster based on their distance to cluster's centroids. Then, the corresponding transfer function model is fetched.

3. During prediction, the transfer function estimates the wearable's raw sensing data from the phone's.

5. IMPLEMENTATION

5.1 Predictor Implementation

Offline. We implement the offline training pipeline in R. In particular, we used `kmeans()` in `stats` and `cl_predict()` in `clue` for clustering and classification, and `auto.arima()` and `forecast()` in the `forecast` package for transfer function learning and prediction.

On a 48-core AMD Opteron processor, training on a pair of time series with 65k data points and 21 extracted features completes in 6.6 minutes. The predictor module exports trained transfer functions as coefficients to json files. The module exports k-means clusters as their centroids to text files.

Online. On a daily basis, TelepathPhone downloads from the cloud the model files, which contain transfer functions' coefficients and clusters' centroids. At runtime, when sensing is offloaded, TelepathPhone loads the files with `org.json` library. Given sensing data features and a set of centroids, the closest cluster is identified and the appropriate transfer function model is selected. The prediction function is implemented in Android based on R `forecast` library.

5.2 Runtime Implementation

As mentioned earlier, a Telepath app has two instances, TelepathWear and TelepathPhone, which run on the respective devices. They implement separate modules that manage interfaces and offloading decisions.

TelepathWear. For sensing apps, TelepathWear sends a 1-second buffer of data to the phone every 15 minutes to determine if offloading is suitable. If the phone rejects the request for offloading, TelepathWear executes locally. Otherwise, it starts (or continues) remote execution.

In local execution, TelepathWear registers local sensors, listens to sensor events, and receives sensory data. TelepathWear calls compute(), which is defined by the developer and classifies the activity before returning results to UI.

In remote execution, when an offloading request is accepted, TelepathWear sends a "start" command, along with sensor IDs, their sampling frequencies and the frequency that results are sent from the phone. Instead of listening for sensor events, TelepathWear now listens to incoming data over Bluetooth, receives classification results from the phone, and returns results to UI. When a remote execution halts, TelepathWear sends a "stop" command, with sensor IDs, instructing the phone to stop sensing and computing.

TelepathPhone. TelepathPhone is a background service that runs continuously, monitoring the data link and listening for commands from TelepathWear. If the command is "bootstrapping," it receives and stores app configurations and

Classifier	Description	Category
C5.0	Decision tree boosting	Decision Tree
glm	Logistic regression	Linear Classifiers
knn	k-Nearest neighbor	Kernel estimation
lda	Linear discriminant analysis	Linear Classifiers
lssvmRadial	Least squares SVM	SVM
mlpWeightDecay	Multi-layer perceptron	Neural Networks
nb	Naive Bayes	Linear Classifiers
nnet	Single-layer perceptron	Neural Networks
parRF	Parallel randomforest	Decision Tree
svmRadial	SVM with radial basis kernel	SVM

Table 2: Benchmark classifiers.

models. If the command is "start," it receives sensor IDs and frequencies that are used to begin sensing and computing. It sends results to TelepathWear periodically based on the configured result frequency. If the command is "stop," TelepathPhone unregisters the sensors and stops both sensing and computing.

TelepathPhone guards against inaccurate prediction and classification. Upon receiving an offload request, the phone obtains buffered sensing data from the watch and compares it against Telepath's prediction. Kendall's rank correlation is computed and the p-value is checked (see Section 4). Offloading requests are rejected when the p-value exceeds 0.01, which indicates that the phone is used in ways that preclude accurate sensing and prediction.

6. EXPERIMENTAL METHODS

Activities and Classifiers. Our evaluation studies the performance and efficiency of Telepath for sensing applications. We survey the characteristics of the top fourteen free activity tracking apps on the wearable app market. They constantly check motion sensors, acquiring wake locks and motion sensors in their permissions, to record running, walking, or cycling activities during the day.

We evaluate Telepath's accuracy in recognizing these three activities. Because we cannot know the algorithms used in these commercial apps, we test Telepath on ten popular machine learning classifiers (see Table 2). The classifiers are diverse and vary in complexity, from the simple decision tree to more complex neural networks.

Data. During data collection, a user wears a Samsung Gear Live watch on her left wrist and places a Samsung Galaxy Nexus phone in her right pocket. A user performs each activity—running, walking, cycling— for 15 minutes to produce approximately 270K raw sensing data points in total. Both devices record data at the highest sampling rate. The user labels data with corresponding activity to supply groundtruth. We sample 70% of the data for training and use an equal amount of data from each activity.

7. EVALUATION

We compare Telepath against alternatives with less sophisticated data analysis: (1) Telepath without using gyroscope, (2) Telepath without using multivariate transfer function models, (3) Telepath without de-noising and (4) directly using the phone's raw sensing data without Telepath's prediction.

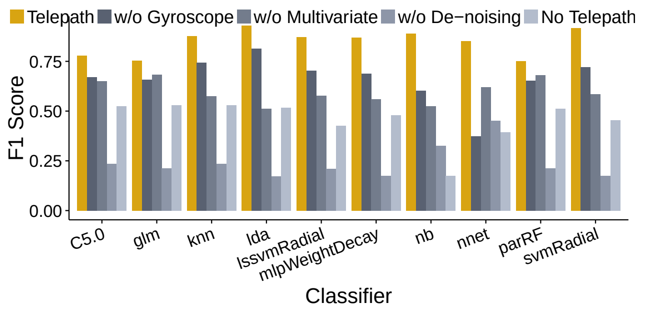

Figure 3: Activity classification accuracy. F1 scores are normalized to those when using groundtruth sensing data.

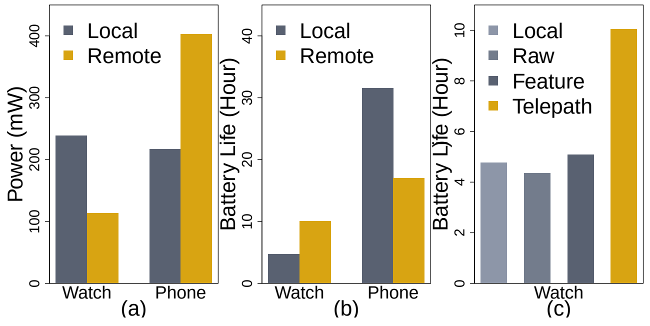

Figure 4: (a) Power and (b) battery life under local and remote (Telepath) execution. (c) Battery life under variants of remote execution that transmit raw data, transmit extracted features, or transmit nothing by relying on Telepath prediction.

Groundtruth is using sensing data collected from the watch.

7.1 Classification Accuracy

We define "app accuracy" as activity classification accuracy when using predicted sensing data. Our metric is the F1 score. Given a classifier, for each activity i, Precision$_i$ is the fraction of instances classified as activity i that indeed correspond to i. Recall$_i$ is the fraction of instances of activity i that are recognized as such. The F1 score accounts for recall and precision on a scale of zero (worst) to one (best).

In Figure 3, we evaluate app accuracy with the motion recognition classifiers in Table 2. The F1 Scores are averaged across activities and normalized to those when using groundtruth data, the watch's sensing data for activity recognition. Telepath outperforms other variants and achieves, on average, 87% of groundtruth accuracy. Observe that gyroscope data has a moderate impact on improving accuracy, while multivariate modeling is crucial for accurate data prediction and activity classification. Low scores without de-noising mean that Telepath is sensitive to noise and data pre-processing is essential to training.

7.2 Energy Efficiency

We compare system power in local and remote execution modes, profiling TelepathWear and TelepathPhone with Android Batterystats. We model a Telepath app that continuously recognizes biking, running and walking activities from accelerometer readings.

Figure 4(a) shows how watch power decreases by 2× while phone power increases when switching from local to remote execution. The watch dissipates power during remote execution because it receives classification results from the phone every second (configured by app). Similarly, the phone dissipates power during local execution because it listens continuously for Telepath commands.

Figure 4(b) shows that remote execution increases the watch's battery life from 4.8 hours to 10.0 hours but reduces the phone's battery life to 17 hours. Note that Telepath could further increase the watch's battery life with asynchronous recognition. When activity logging apps are neither real-time nor interactive, the phone can buffer and send classification results less frequently to reduce transmission costs.

Figure 4(c) shows that Telepath's offloading strategy benefits battery life more than traditional strategies. Invoking the phone for both sensing and computing extends the wearable's battery life to 10 hours. In contrast, relying on the watch's sensors but transmitting raw data or extracted features to the phone for computation does little for battery life. Telepath outperforms these strategies by 2.3x and 2.0x, respectively. Indeed, these strategies perform no better than using the watch's local resources.

7.3 Costs and Overheads

Training. Telepath models must train quickly to permit frequent updates. When the dataset has 6.5k paired data points, Telepath takes roughly 7 minutes to train models for 10 clusters. Training time can be reduced by increasing the number of clusters.

Prediction. During remote execution, the transfer function predictor on the Samsung Galaxy Nexus phone requires 3.42ms, on average, to compute features, classify, and predict. This prediction latency is shorter than the sensors' 10ms sampling period, which means the predictor may be invoked for every sensor event and does not affect app performance.

8. RELATED WORK & DISCUSSION

Offloading has been extensively studied for mobile-cloud computing (MCC). Mobile devices offload compute-intensive apps—image, video, audio processing, gaming—to the cloud to reduce execution time, reduce energy, or improve service quality [5]. Telepath adapts offloading to extend battery life for wearable devices. Similar to prior offloading works, Telepath needs to make the trade-off between energy and accuracy. Future works include quantifying this trade-off for different users and device placements. In addition, polices for duty-cycle sensors could consider Telepath's impact on the aggregate battery life of both devices.

Sensing apps can benefit from hardware accelerators that reduce the power and latency in signal processing and activity classification [10]. Telepath is orthogonal to these approaches as it allows wearable devices to preserve energy and utilize idle resources on phones, regardless of how they are implemented. We view Telepath as more than a temporary solution for current wearable devices, but instead, a sensing resource sharing technique for future wearable usages.

Admittedly, Telepath is constrained to motion sensors such as accelerometers and gyroscopes. Wearables are equipped with other sensors such as GPS and heart rate monitors, which do not yet benefit from Telepath. GPS sensing can be offloaded without Telepath's sophisticated signal processing [9]. Heart monitoring might require more comprehensive learning and training, as there is no correlated sensor on the phone.

Acknowledgments

This work is supported by the National Science Foundation under grants CCF-1149252 (CAREER), CCF-1337215 (XPS-CLCCA), SHF-1527610, and AF-1408784. This work is also supported by STARnet, a Semiconductor Research Corporation program, sponsored by MARCO and DARPA. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of these sponsors

9. REFERENCES

[1] D. Anguita, A. Ghio, L. Oneto, X. Parra, and J. L. Reyes-Ortiz. Human activity recognition on smartphones using a multiclass hardware-friendly support vector machine. In *International Conference on Ambient Assisted Living and Home Care (WAAL)*, 2012.

[2] D. Anguita, A. Ghio, L. Oneto, X. Parra, and J. L. Reyes-Ortiz. A public domain dataset for human activity recognition using smartphones. In *European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN)*, 2013.

[3] S. Bhattacharya and N. D. Lane. From smart to deep: Robust activity recognition on smartwatches using deep learning. In *International Conference on Pervasive Computing and Communication Workshops (PerCom)*, 2016.

[4] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl. Maui: Making smartphones last longer with code offload. MobiSys, 2010.

[5] N. Fernando, S. W. Loke, and W. Rahayu. Mobile cloud computing: A survey. *Future Generation Computer Systems*, 29(1):84–106, 2013.

[6] H. Ghasemzadeh, V. Loseu, and R. Jafari. Wearable coach for sport training: A quantitative model to evaluate wrist-rotation in golf. *Journal of Ambient Intelligence and Smart Environments*, 2009.

[7] R. M. Helmer and J. K. Johansson. An exposition of the box-jenkins transfer function analysis with an application to the advertising-sales relationship. *Journal of Marketing Research*, pages 227–239, 1977.

[8] S. S. Intille, K. Larson, J. Beaudin, J. Nawyn, E. M. Tapia, and P. Kaushik. A living laboratory for the design and evaluation of ubiquitous computing technologies. In *Conference on Human Factors in Computing Systems (CHI)*, 2005.

[9] J. Liu, B. Priyantha, T. Hart, H. S. Ramos, A. A. Loureiro, and Q. Wang. Energy efficient gps sensing with cloud offloading. In *Conference on Embedded Network Sensor Systems*, 2012.

[10] D. Mahajan, J. Park, E. Amaro, H. Sharma, A. Yazdanbakhsh, J. K. Kim, and H. Esmaeilzadeh. Tabla: A unified template-based framework for accelerating statistical machine learning. In *International Symposium on High Performance Computer Architecture (HPCA)*, 2016.

[11] M. Shoaib, 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.

[12] M. Zhang and A. A. Sawchuk. Usc-had: A daily activity dataset for ubiquitous activity recognition using wearable sensors. In *Conference on Ubiquitous Computing*, 2012.