

Exploring Hardware Heterogeneity to Improve Pervasive Context Inferences

Chenguang Shen and Mani Srivastava, University of California, Los Angeles

Context-aware inference apps have become pervasive as a result of the Internet of Things (IoT). However, most of these apps run continuously on a single device, resulting in limited sensor coverage and high energy consumption. Recent advances in IoT devices, specifically hardware heterogeneity, can be leveraged to improve the accuracy and energy efficiency of context inferences.

Advances in VLSI technologies have led to the integration of unprecedented sensing, communication, and computation capabilities in today's devices, allowing them to continuously observe individuals and their spaces. Smartphones have evolved from communication devices to powerful personal computing platforms, and connected devices such as smartwatches, cameras, motion sensors, thermostats, and energy meters—collectively dubbed the Internet of Things (IoT)—have rapidly permeated our lives, leading to the emergence of an ecosystem of context-aware apps. These apps—such as Moves, Map-MyRun, Strava, Runkeeper, and Vimofit—use sensors

like GPS, microphones, accelerometers, and gyroscopes to make diverse inferences about user activities and contexts, including transportation mode, location, physical exercise, mood, and stress level. However, there are two major drawbacks associated with the execution of context inferences on IoT devices: limited sensor coverage and inference accuracy, and high energy consumption.

Sensor coverage and inference accuracy are critical for inference fidelity, which largely depends on the relative position of the sensing device to the user. Most of today's inference apps run on a single device, such as a smartphone, and tend to rely on phone inertial sensor readings, requiring users to carry their smartphone with

them. This means that whenever users leave the phone elsewhere (for example, on a desk) to focus on other tasks or to charge the phone, the sensors cannot capture any meaningful data, resulting in limited sensor coverage and poor inference accuracy.

In addition, many IoT devices are battery powered and have a limited energy budget. The algorithms used by context inference apps, including feature computations and classifications, are computationally expensive and power hungry. They often need to run continuously in the background for just-in-time feedback, notifications, and interventions. Currently, most always-on context inferences run on the device's main app processor, representing a significant portion of the device's overall workload and energy consumption. This high power consumption is usually exacerbated by the use of power-hungry sensors like GPS and cellular radio.

In response, we propose leveraging advancements in IoT devices, specifically hardware heterogeneity, to improve the accuracy and energy efficiency of context inferences. This includes both device and processor heterogeneity.

HARDWARE HETEROGENEITY

Using smartwatches as an example, we demonstrate the use of heterogeneous IoT devices and processors to improve context inferences.

Device heterogeneity

Unlike previous wearable devices dedicated to fitness tracking, today's commercial off-the-shelf (COTS) smartwatches such as the Apple Watch (iOS) and the Moto 360 (Android) benefit from powerful hardware resources such as CPUs and RAM similar to those of modern smartphones. With seamless connection to phones via Bluetooth Low Energy (BLE), these wearables can act as user portals for smartphone apps or host standalone apps. These devices also provide a rich set of sensors to support context inferences. Their lightweight nature makes them ideal for always-on context monitoring. Moreover, watches are less obtrusive than phones because wearing them is less likely to interfere with a user's daily routine. By coordinating smartwatches and smartphones, sensing and inference workloads can be alternated between an available device at any time for improved sensor coverage and inference accuracy. Using a watch and a phone for inference executions, we achieved up to 37 percent improvement in inference accuracy and up to 61 percent less energy consumption.

Processor heterogeneity

Modern mobile processors such as the Qualcomm Snapdragon (www.qualcomm.com/products/snapdragon/processors) are sophisticated systems on chip (SoCs), where the main app processors are complemented by heterogeneous coprocessors. We demonstrate the improvement of energy efficiency by offloading computation from the main app processor (CPU) to a secondary low-power processor. Recently, mobile chip vendors have undertaken efforts to make these previously hidden coprocessors—such as the digital signal processor (DSP)—programmable. For instance, the Snapdragon 820 series SoC includes a Hexagon 680 DSP (developer.qualcomm.com/software/hexagon-dsp-sdk/dsp-processor) that can be custom-programmed and operates in the ultra-low-power range. Offloading the classification stage commonly seen in an inference pipeline to a DSP resulted in up to 60 percent energy savings with a negligible effect on latency.

IoT device heterogeneity

To explore the benefits of leveraging device and processor heterogeneity, we showcase the specifications of both IoT devices and mobile SoCs.

Table 1 compares the hardware specifications of popular smartwatches and smartphones. Smartwatches today have rather powerful CPUs, RAM, and radios, enabling them to execute standalone apps without any assistance from smartphones. Both the Apple Watch and the Moto 360 contain radios and sensors similar to today's smartphones, including optical-based heart-rate sensors. However, because of volume and weight limits, smartwatches have a much smaller battery capacity than smartphones. Due to the similarity in smartwatch hardware, we chose the Moto 360 as an example platform to benchmark our app scenarios.

To further analyze the feasibility of running context inferences on smartwatches, we created basic power profiles of a Moto 360 watch and a Nexus 5 phone (see Table 2). We measured the power consumption of the watch and the phone—both connected via BLE—and profiled both devices for screen off (sleeping) and screen on, because their normal power consumption is typically between these two extreme cases. Compared with the Nexus 5 phone, the Moto 360 watch generally has lower power consumption. However, the power difference between screen off and screen on is much more significant on the watch than on the phone, requiring a detailed study of the power and energy tradeoffs of inference executions.

TABLE 1. Comparison of device hardware platforms.

Device	System on chip (SoC)	CPU	RAM	Storage	Radio	Battery	Weight	Sensors
Apple Watch	Apple S1	520 MHz S1	512 Mbytes	8 Gbytes	Bluetooth Low Energy (BLE)/Wi-Fi/near-field communication (NFC)	3.8 V 205 mAh (milliamp hours)	25 g	Accelerometer, gyroscope, pedometer, heart rate, microphone
Moto 360 watch	TI OMAP 3630	1 GHz OMAP3	512 Mbytes	4 Gbytes	BLE/Wi-Fi	3.8 V 320 mAh	49 g	Accelerometer, gyroscope, pedometer, heart rate, microphone, light
Nexus 5 phone	Snapdragon 800	2.3 GHz Krait	2 Gbytes	16/32 Gbytes	BLE/Wi-Fi/NFC	4.3 V 2300 mAh	130 g	Accelerometer, gyroscope, pedometer, microphone, compass, proximity, light, GPS

TABLE 2. Comparison of device power profiles.

Device	Power (W)	Current (mA)	Lifetime (h)
Moto 360 watch (screen off)	0.013	3.283	97.472
Moto 360 watch (screen on)	0.550	142.520	2.245
Nexus 5 phone (screen off)	0.254	58.913	37.343
Nexus 5 phone (screen on)	1.853	435.260	5.057

Heterogeneous mobile SoCs

Mobile processors are no longer simply app processors, but are sophisticated SoCs where the main app processors are complemented by a set of heterogeneous processors. Table 3 summarizes the specifications of mainstream mobile SoCs today. The types of heterogeneity in mobile SoCs can be categorized as loosely coupled or tightly coupled.

Loosely coupled. Mobile SoCs include a set of embedded processors such as DSPs and GPUs that handle specialized work such as media processing and low-level sensor I/O. These loosely coupled heterogeneous processors typically have no cache coherence with the app processor, and have a different instruction-set architecture than app processors. The coprocessors are usually hidden from the app developers and are instead limited to running prebuilt firmware provided by the platform manufacturer. Recently, mobile processor vendors have been making conscious efforts to expose the coprocessor heterogeneity to the app developers, allowing them to program the DSPs. The Hexagon DSP is a representative example of such an embedded processor, so we used it as an example of a secondary processor to showcase the benefits of leveraging processor heterogeneity.

Tightly coupled. ARM’s big.LITTLE (www.arm.com/products/processors/technologies/biglittleprocessing.php), which couples relatively slower, lower-power processor cores with

more powerful and power-hungry ones, creates a multi-core processor that can adjust better to dynamic computing needs and uses less power than clock scaling alone. big.LITTLE is a tightly coupled heterogeneous architecture because the big cores and the little cores typically have cache coherence and share the same instruction-set architectures. Nevertheless, they have different power-performance operating points. As shown in Table 3, major mobile SoC manufacturers today all adopt the big.LITTLE architecture by pairing big cores with LITTLE cores as app processors.

SYSTEM DESIGN

Our system design approach leverages hardware heterogeneity to improve inference accuracy and to reduce the energy consumption of always-on context inferences.

Context-aware apps employ a suite of machine-learning algorithms to extract semantically meaningful inferences from sensor data. Underlying the various types of context inferences, we abstracted out a canonical inference pipeline, shown in the upper part of Figure 1. Apps today typically subscribe to raw sensor datastreams, and extract features to reduce the dimensionality of sensor data. Apps then perform classification over the stream of extracted features using pretrained machine-learning models. The classification returns context labels (for example, which class the current context falls under) for notifications and corresponding actions.

TABLE 3. Specifications of latest mobile SoCs.

Mobile SoC	App processor (CPU)	Coprocessor	Used by
Qualcomm Snapdragon 821	Dual Kryo 2.15 GHz + Dual Kryo 1.59 GHz	Hexagon 680 digital signal processor (DSP), Adreno 530 GPU	Google Pixel, Pixel XL
Apple A10	Dual Hurricane 2.34 GHz + Dual Zephyr (ARMv8-A)	M10 motion processor, 6-core GPU	Apple iPhone 7, iPhone 7 Plus
Samsung Exynos 8890	Quad Exynos M1 2.3 GHz + Quad Cortex-A53 1.6 GHz	ARM Mali-T880 GPU	Samsung Galaxy S7, Galaxy Note 7
HiSilicon Kirin 950/955	Quad Cortex-A72 2.5 GHz + Quad Cortex-A53 1.8 GHz	ARM Mali-T880 GPU	Huawei Mate 8, Huawei P9
NVIDIA Tegra X1	Quad Cortex-A72 1.9 GHz + Quad Cortex-A53 1.3 GHz	Maxwell GPU	NVIDIA Shield TV, Google Pixel C

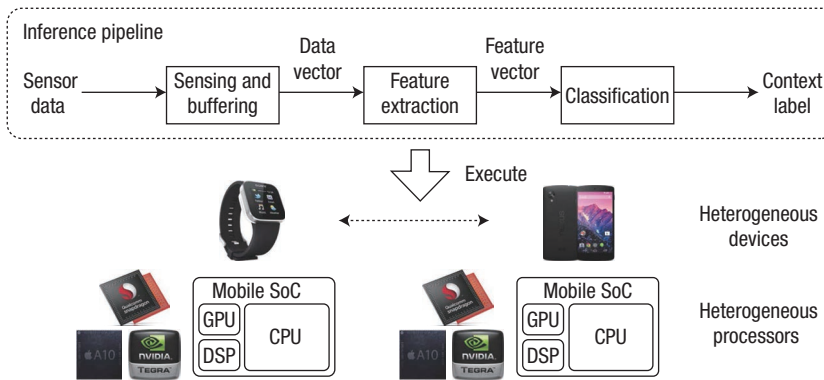


FIGURE 1. Overview of the proposed system architecture, which uses both heterogeneous devices and heterogeneous processors to achieve better inference accuracy and energy efficiency. A canonical inference pipeline, which outputs context labels from the sensor datastream, is shown in the upper section. SoC: system on chip; DSP: digital signal processor.

Increasing inference accuracy

Instead of running inferences on a single smartphone, the inference pipeline can run across heterogeneous devices. The execution can be offloaded to another IoT device, such as a smartwatch, whenever the phone is unavailable for better sensor coverage and inference accuracy.

The hardware comparison previously mentioned suggests that today’s smartwatches have powerful computation resources and sensors that enable apps to draw inferences. Smartwatch users are more likely to wear the watch throughout the day than they are to carry their phone with them, except for specific periods such as sleep. Because phone placement can greatly affect sensor readings (in other words, a phone cannot capture meaningful human movements when not being carried by the user), watches can help increase sensor coverage and inference accuracy by continuing the inference execution even when the phone is away from the user. Such situations can be identified when movements are detected by the watch but not by the phone.

Optimizing energy consumption

The energy benefit of leveraging hardware heterogeneity is two-fold.

Offloading to low-power processors. To show that considerable energy savings can be achieved by offloading the execution of inference algorithms from the main app processor to the DSP available in mobile SoCs, we offloaded the classification stage as an example. Compared to feature extraction, classification computation is more unstructured, possibly leading to larger energy savings by the DSP offload.

Replacing high-energy sensors. Sensors on IoT devices, such as smartwatch inertial sensors, can provide additional information about user behaviors and contexts.

We use smartwatches as an example to represent heterogeneous IoT devices. By using smartwatches to run inferences, we can reduce the energy consumption of apps by eliminating the use of high-power sensors such as a phone’s GPS.

PRELIMINARY RESULTS

Here, we present a set of preliminary experimental results to demonstrate the improvements in inference accuracy and energy efficiency as a result of exploiting hardware heterogeneity.

Example app scenarios

To investigate the feasibility of offloading classification algorithms to the DSP and to show potential gains in energy efficiency, we developed an activity recognition app and a speaker recognition app as example context inferences.

Activity recognition. Motivated by prior work in the mobile sensing space (see the “Related Work” sidebar), our activity

RELATED WORK

recognition app uses smartwatch accelerometers in addition to traditional phone accelerometers. The app uses machine-learning models to generate activity labels and can be extended to identify more classes of activities.

We used two datasets to train models and to benchmark the app. The first one was collected by TU Darmstadt¹ and includes phone accelerometer data over a seven-day time period and activity labels of {dinner, commuting, lunch, work, undefined}. The second dataset was collected from our own user study where participants recorded synchronized watch accelerometer, phone accelerometer, and phone GPS data as well as labels of {still, walking, commuting}. The total duration of this dataset is 6.33 h.

Speaker recognition. In this classical speaker recognition app, we trained a Gaussian mixture model (GMM) for each speaker. A likelihood score was calculated for each sound clip, representing the probability of the sample being generated by the GMM. The speaker corresponding to the maximum likelihood was set as the output. We used the TIDIGITS dataset (catalog.ldc.upenn.edu/ldc93s10) for training, and a set of Mel-frequency cepstral coefficients (MFCCs) featured on a 3-s sound clip at 8 kHz was used as one sample.

Experiment: improving inference accuracy

We first quantified the accuracy improvements of running activity recognition using both a Moto 360 watch and a Nexus 5 phone instead of using only a single phone.

Inference composition. The inference of activity recognition was composed using several different sensor combinations (SCs):

Although additional sensors and computational offloading are found in prior approaches, they often rely on custom hardware to improve the inference accuracy and energy efficiency. Our work quantifies the benefit of leveraging hardware heterogeneity seen in commercial off-the-shelf devices. Recent work has started using smartwatches to assist context inferences running on smartphones, such as exercise tracking¹ and driving habit detection.² In the field of computation offloading, Moo-Ryong Ra and his colleagues examined the partitioning of different modules in a sensing pipeline between processors to reduce the overall energy consumption.³ Felix Xiaozhu Lin and his colleagues compared loosely coupled heterogeneous processor cores with tightly coupled ones.⁴ DSP.Ear⁵ and DeepEar⁶ demonstrate the performance and energy implications of executing inferences as digital signal processor (DSP) modules, including both traditional models and deep neural networks.

We previously wrote about leveraging processor heterogeneity in a workshop paper that focused on mobile phones but did not cover device heterogeneity.⁷ The cross-device framework presented here in the main text is related to our previous work⁸ but has a different and more specific implementation.

References

1. B.J. Mortazavi et al., "Determining the Single Best Axis for Exercise Repetition Recognition and Counting on Smartwatches," *Proc. 11th Int'l Conf. Wearable and Implantable Body Sensor Networks (BSN 14)*, 2014; doi:10.1109/BSN.2014.21.
2. L. Liu et al., "Toward Detection of Unsafe Driving with Wearables," *Proc. 2015 Workshop Wearable Systems and Applications (WearSys 15)*, 2015, pp. 27–32.
3. M.-R. Ra et al., "Improving Energy Efficiency of Personal Sensing Applications with Heterogeneous Multi-processors," *Proc. 2012 ACM Conf. Ubiquitous Computing (UbiComp 12)*, 2012; doi:10.1145/2370216.2370218.
4. F.X. Lin, Z. Wang, and L. Zhong, "Supporting Distributed Execution of Smartphone Workloads on Loosely Coupled Heterogeneous Processors," *Proc. 2012 USENIX Conf. Power-Aware Computing and Systems (HotPower 12)*, 2012; dl.acm.org/citation.cfm?id=2387869.2387871.
5. P. Georgiev et al., "DSP.Ear: Leveraging Co-Processor Support for Continuous Audio Sensing on Smartphones," *Proc. 12th ACM Conf. Embedded Network Sensor Systems (SenSys 14)*, 2014, pp. 295–309.
6. N.D. Lane, P. Georgiev, and L. Qendro, "DeepEar: Robust Smartphone Audio Sensing in Unconstrained Acoustic Environments Using Deep Learning," *Proc. 2015 ACM Int'l Joint Conf. Pervasive and Ubiquitous Computing (UbiComp 15)*, 2015, pp. 283–294.
7. C. Shen et al., "Exploiting Processor Heterogeneity for Energy Efficient Context Inference on Mobile Phones," *Proc. Workshop Power-Aware Computing and Systems (HotPower 13)*, 2013, article no. 9.
8. C. Shen et al., "Beam: Ending Monolithic Applications for Connected Devices," *Proc. 2016 USENIX Ann. Technical Conf. (USENIX ATC 16)*, 2016, pp. 143–157.

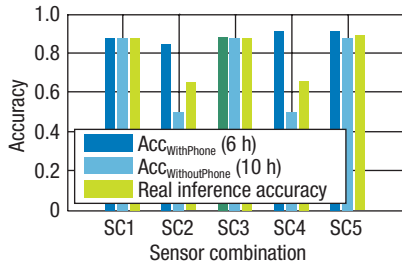


FIGURE 2. Real inference accuracy of activity recognition. Using smartwatch sensors can result in a 35 percent improvement in inference accuracy compared with using smartphone sensors. SC1: watch accelerometer; SC2: phone accelerometer; SC3: watch and phone accelerometer; SC4: phone accelerometer and phone GPS; SC5: watch and phone accelerometer and phone GPS.

- › SC1: watch accelerometer only.
- › SC2: phone accelerometer only.
- › SC3: watch accelerometer and phone accelerometer.
- › SC4: phone accelerometer and phone GPS.
- › SC5: watch accelerometer, phone accelerometer, and phone GPS.

We then calculated features from the raw sensor datastream by applying a 1-s classification window, and trained classification models such as decision tree, random forest, and support vector machine (SVM) using each SC. Based on our experiments, we chose to use the decision tree for activity recognition considering both accuracy and simplicity.

Results. Motivated by observations from our user study where participants were asked to perform workout exercises and activities while carrying a smartphone and wearing a smartwatch, we created an example user daily schedule. Based on the schedule, a user wears a smartwatch throughout the entire day except while sleeping (16 h). On the other hand, a user only carries a smartphone for half of the working day (4 h) plus while driving (1.5 h) and during lunch (0.5 h). The phone is with the user (and thus can capture meaningful sensor data) for only 6 of the 16 hours in a day, and is placed elsewhere for the remaining 10. Therefore, the total duration of a user carrying both the watch and the phone ($T_{WithPhone}$) is 6 h each day, while the total duration of a user carrying only the watch ($T_{WithoutPhone}$) is 10 h.

We define real inference accuracy as a weighted average of inference accuracy numbers considering the location of the device:

$$Acc_{Real} = \frac{1}{T_{Total}} \left(Acc_{WithPhone} \times T_{WithPhone} + Acc_{WithoutPhone} \times T_{WithoutPhone} \right). \tag{1}$$

If the inference samples sensor data from the watch (such as SC1, SC3, and SC5), $Acc_{WithPhone}$ will be the inference

accuracy using sensors from both the phone and the watch, and $Acc_{WithoutPhone}$ will be the accuracy using only watch sensors. If the inference takes only phone sensor data (such as SC2 and SC4), $Acc_{WithPhone}$ will be the accuracy using only phone sensors. However, when the phone is placed elsewhere ($Acc_{WithoutPhone}$), the inference cannot capture any meaningful inertial data and the app has no way to infer the user’s current activity other than performing a random guess. Given that there are other heuristics to infer the user activity, such as using GPS and time, we set $Acc_{WithoutPhone}$ in these cases to be 50 percent. Finally, T_{Total} is the total time of the day except sleep (16 h).

Figure 2 shows the real inference accuracy of activity recognition considering the routine above. Relying solely on the phone sensors for activity recognition will lead to poor real inference accuracy due to the phone’s limited sensor coverage, as seen in SC2 (64.78 percent) and SC4 (65.51 percent). Accuracy is notably improved with the help of smartwatches. Using only the watch sensor (SC1) results in a real accuracy of 87.50 percent, or a 35.1 percent improvement compared with using the phone only. Moreover, fusing the watch and phone sensors together improves the real accuracy of activity recognition to 87.85 percent, or by 35.6 percent, without the GPS (SC3), and to 89.01 percent, or by 37.4 percent, with the GPS (SC5). Note that the classification model used here is relatively simple, so the additional information provided by fusing both the phone and watch accelerometers does not significantly improve accuracy. However, using more sensors would detect a wider range of movements if the app targets complex activities such as exercise.

Experiment: optimizing energy consumption

Here, we describe the energy optimization of continuous context inferences with the help of both processor heterogeneity and device heterogeneity.

Offloading classifications to DSP. To investigate the feasibility of offloading classification algorithms to the DSP and to show potential gains in energy efficiency, we implemented both activity recognition using SVMs and a speaker recognition app using GMMs. For the implementation, we used mobile development platforms with open programmable DSPs, including a Qualcomm Snapdragon development tablet and a TI Pandaboard, both running Android. We used Agilent 34410A digital multimeters for board-level energy measurements.

Table 4 shows the latency profiling result for activity recognition (SVM) and speaker recognition (GMM) classifications, executed on both a TI Pandaboard (OMAP4460 SoC) and a Qualcomm tablet (Snapdragon 800 SoC). We define latency as the time to classify one sample. Although running the classification on DSPs instead of CPUs incurs certain overhead in terms of latency, it is still well within the window size for feature computations: 400 ms for activity recognition and 3 s for speaker recognition (in other words, the current classification result will be valid until the next complete window). Offloading classifications to the DSP will lead to only negligible latency overhead.

We then explored the energy implication brought by offloading computation to the DSP with the intuition that despite the increased latency, the DSP's specialized instruction-set architecture will result in overall improvement in energy efficiency. Using the SVM classification in activity recognition as an example, we profiled the entire platform's energy savings due to the offload.

As shown in Table 5, offloading can result in 17 percent and 60 percent board-level energy savings from continuous inference execution on the OMAP4 Pandaboard and the Qualcomm Snapdragon tablet, respectively. Note that the Snapdragon SoC has a more powerful and power-hungry CPU as well as a lower-power DSP than the OMAP4. This means that offloading computation to the DSP results in much larger energy savings on the Snapdragon.

Replacing high-energy sensors. As shown previously, using watch and phone accelerometers (SC3) for activity recognition can result in similar or even better real inference accuracy compared with solutions that only use phone GPS (SC4 and SC5). Table 6 shows the average power consumption of a phone when activity recognition is running using different SCs. By replacing the phone's location sensor used in activity recognition with the watch accelerometer, the phone consumes 61.0 percent and 35.5 percent less energy than when GPS locations and network locations are used, respectively. The improved inference accuracy and energy efficiency show that smartwatch sensors are capable of replacing high-energy phone sensors.

We also consider the optimal partitioning of an inference pipeline between devices. Performing the entire inference pipeline in Figure 1 locally on the smartwatch (0.112 W) can help reduce up to 67 percent watch energy consumption, compared with performing only sampling and buffering on smartwatches and sending data to the

TABLE 4. Performance overhead from offloading classifications.

Processor	Activity recognition latency (ms)	Speaker recognition latency (s)
OMAP4 CPU	1.320	0.043
OMAP4 DSP	9.232	0.704
Snapdragon CPU	0.357	0.018
Snapdragon DSP	3.625	0.075

TABLE 5. Device-level energy comparison with 20-s continuous support vector machine classification execution.

Platform	Energy (J)		Savings (%)
	CPU	DSP	
Pandaboard	48.64	40.1	17
Snapdragon	45.66	18.49	60

TABLE 6. Nexus 5 energy comparison when running activity recognition with different sensor combinations.

Sensor combination	Phone power consumption (W)
Phone accelerometer + GPS location	0.788
Phone accelerometer + network location	0.508
Phone accelerometer + watch accuracy	0.307

phone via BLE (0.343 W). This ensures that executing context inferences on smartwatches will not significantly affect their battery lives.

CROSS-DEVICE FRAMEWORK

To assist inference apps in leveraging watch-phone collaboration for better inference accuracy and energy efficiency, we created an open source framework (github.com/nestl/ContextAwarenessToolkit) for composing context inferences across devices, illustrated in Figure 3. Here, inferences are managed by the inference manager running on the watch and the phone, which also coordinates the communication if the same inference has both watch and phone components. The framework requires app developers to encapsulate inference apps into a set of modules, enabling

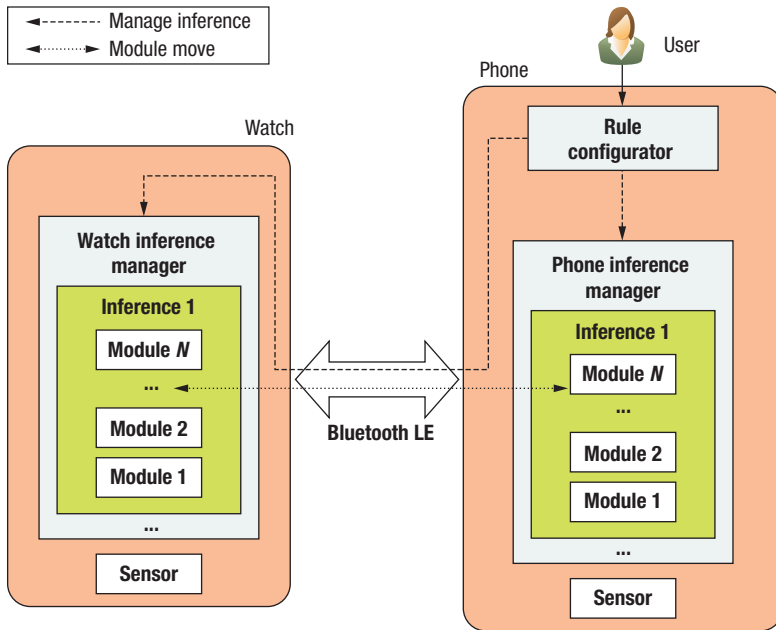



FIGURE 3. Proposed inference framework across smartphones and smartwatches. Modular context inferences can be executed across different devices. LE: Low energy.

controlling the inference managers on both devices accordingly, such as executing inferences on other available devices if one device runs out of battery power or cannot capture meaningful sensor data. With this framework, the combined battery life of the phone, the watch, and other possible devices can be maximized based on the user's preferences.

We propose that context inferences can be executed across smartwatches and smartphones, and that certain stages of the inference pipeline can be off-loaded from main app processors to DSPs to improve energy efficiency and inference accuracy in IoT devices. In the future, we plan to further demonstrate the benefits of hardware heterogeneity, such as more efficient resource usage and task scheduling in concurrent inference executions, and investigate other aspects of context inferences, including data privacy and cross-platform support. 

ABOUT THE AUTHORS

CHENGUANG SHEN is a PhD candidate in computer science at the University of California, Los Angeles (UCLA). His research focuses on developing a mobile sensing framework for context awareness. Shen received an MS in computer science from UCLA. Contact him at cgshen@ucla.edu.

MANI SRIVASTAVA is a professor in the Electrical Engineering and Computer Science Department at UCLA. His research interests include wireless networking, embedded systems, sensor networks, mobile and ubiquitous computing, low-power and power-aware systems, and embedded technologies for health and sustainability. Srivastava received PhD degrees in electrical engineering and computer science from the University of California, Berkeley, and was with Bell Laboratory Research before joining UCLA as a faculty member in 1997. Contact him at mbs@ucla.edu.

the inference manager to move them across devices by setting their execution target (for example, phone/watch). The framework can also be extended to support scheduling inference executions across heterogeneous processors when COTS smartphones and smartwatches ship with open-programmable low-power cores.

Users configure the inference execution using a rule configurator and can prioritize saving the phone battery or the watch battery. The configurator enforces policies by

REFERENCE

1. T. Huynh, M. Fritz, and B. Schiele, "Discovery of Activity Patterns Using Topic Models," *Proc. 10th Int'l Conf. Ubiquitous Computing (UbiComp 08)*, 2008, pp. 10-19.

myCS

Read your subscriptions through the myCS publications portal at <http://mycs.computer.org>