SpeakerSense: Energy Efficient Unobtrusive Speaker Identification on Mobile Phones

Hong Lu[1], A.J. Bernheim Brush, Bodhi Priyantha, Amy K. Karlson and Jie Liu

Microsoft Research, One Microsoft Way, Redmond, WA, 98052, USA
hong@cs.dartmouth.edu,{ajbrush, bodhip, karlson, liuj}@microsoft.com

Abstract. Automatically identifying the person you are talking with using continuous audio sensing has the potential to enable many pervasive computing applications from memory assistance to annotating life logging data. However, a number of challenges, including energy efficiency and training data acquisition, must be addressed before unobtrusive audio sensing is practical on mobile devices. We built SpeakerSense, a speaker identification prototype that uses a heterogeneous multi-processor hardware architecture that splits computation between a low power processor and the phone's application processor to enable continuous background sensing with minimal power requirements. Using SpeakerSense, we benchmarked several system parameters (sampling rate, GMM complexity, smoothing window size, and amount of training data needed) to identify thresholds that balance computation cost with performance. We also investigated channel compensation methods that make it feasible to acquire training data from phone calls and an automatic segmentation method for training speaker models based on one-to-one conversations.

Keywords: Continuous audio sensing, mobile phones, speaker identification, energy efficiency, heterogeneous multi-processor hardware.

1 Introduction

Forgetting the name of the person you are talking with can be an awkward and uncomfortable experience. Imagine being able to glance unobtrusively at a mobile device to see the name of the person who is speaking and perhaps a few other details about them. Several research projects, most notably SenseCam [2] have explored aiding people's memory using technology. However, these systems provide memory support *retrospectively*, by recording information (e.g., lifelogging) automatically and allowing the user to review events at a later time. With the advances in sensing and processing power, today's mobile phones offer the potential to provide memory assistance to the user in realtime when a memory problem occurs.

Using audio sensing for memory assistance is particularly attractive because all phones have built-in microphones and audio data is less sensitive to the location and orientation of the phone as compared with other common sensors such as cameras and

[1] Current address: Dept. of Computer Science, Dartmouth College, Hanover NH, 03755, USA

accelerometers. In addition to memory assistance, identifying a co-located speaker has a number of other possible uses including allowing people to set "person-based reminders" (e.g., "remind me the next time I speak to John to ask him about his vacation"), filtering social networking status feeds presented to a user based on the people they interact with face to face, or tagging life-logging data artifacts (e.g., photos) to facilitate retrieval at a later point.

Several previous prototypes have studied capturing audio to support retrospective memory assistance [e.g., 1, 5, 17] and have explored speaker identification using additional sensors [12] or multiple phones [7]. However, two key challenges must be addressed before continuous audio sensing for speaker identification is practical:

- Energy efficiency. To be useful for memory assistance and other scenarios, speaker identification must run continuously and unobtrusively in the background and be ready when needed. However, as we show in Section 4, although microphones use very little energy, *on an off-the-shelf smart phone* sampling audio data requires the phone's application processer to run and the ~335mW power consumption of continuous recording will quickly drain a phone's battery.
- Training data acquisition. Training data quality is a key factor in determining the performance of a speaker identification system. For most prototypes, researchers manually gather voice data, label them, and train a speaker model for each participant. Yet, in practice users are unlikely to be willing or able to manually train a system.

In this paper, we present the design and implementation of *SpeakerSense*, a practical, energy efficient, and unobtrusive speaker identification system. SpeakerSense tackles the above challenges by using a heterogeneous multi-processor (HMP) based mobile phone architecture, a set of robust speaker identification methods, and a novel training data collection mechanism via phone calls.

Our experiments show that using an HMP architecture to sample audio, detect high-quality voice data on a low-powered secondary processor and engage the speaker identification pipeline on a phone's processor only when necessary makes continuous audio sensing very efficient. Our system uses ~4.29mW when sensing in the background, and ~771mW on a phone when actively performing speaker identification. Using SpeakerSense, we also benchmarked several system parameters (sampling rate, GMM complexity, smoothing window size, and amount of training data needed) to identify thresholds that balance computation cost with performance. For example, our data show that 3 minutes of audio is a reasonable minimum for the amount of training data needed to train robust speaker models.

To address the challenge of acquiring training data, we investigated several unobtrusive data acquisition methods including using phone calls, one-to-one conversations, and sharing models across phones. We identify the appropriate channel compensation methods that make it feasible to train speaker models using audio data from phones calls, and an automatic segmentation method that can be used for training based on one-to-one conversations. We also validate the feasibility of sharing speaker models trained on one phone with another phone. While work remains to develop SpeakerSense into a system that can be studied with people with memory deficits, our contributions address technical challenges for speaker identification on mobile phones and move continuous audio sensing another step closer to reality.

2 Related Work

In our project, we were inspired by the use of technology to help people with memory deficits. Kapur [4]'s survey of memory aids describes the wide variety in common use which span both technological (e.g., reminders on watch alarms, mobile phones or pillboxes) and non-technological (e.g., wall calendars, notebooks) solutions, and highlights the potential for advances in technology on mobile phones, cameras and location detection devices to provide further memory assistance. Research using the SenseCam [e.g., 2, 5] has shown the potential of lifelogging to help people with memory impairments retrospectively review captured information to assist them in recalling events. Looking specifically at retrospective aids based on audio logging, Vemuri *et al.* developed iRemember [17], a memory retrieval prototype which recorded and transcribed everyday conversations so they could be later searched.

In addition to supporting retrospection, the computing power available in smartphones makes it possible to provide assistance in realtime based on sensed events. The Personal Audio Loop (PAL) system [1] is a near-term audio-based memory aid that continuously records audio into a buffer; when users need assistance recalling something they recently heard (e.g., the name of the person they were just introduced to), the buffer can be played back on-demand. SoundSense [6] explores using audio beyond memory assistance, continuously sensing and classifying audio events to recognize general sound types heard by users (e.g., voice or music) and specific activities (e.g., walking, driving cars, riding elevators). These classifications enable a number of different applications including an audio daily diary and music detection service, which were both prototyped by the authors.

SoundSense and other continuously sensing applications raise concerns about battery efficiency which have been identified and studied. A variety of duty-cycling schemes have been proposed to alleviate battery life issues by sampling intermittently [e.g., 8, 19]. We take a complementary hardware-based approach to enable low power continuous sensing by offloading the initial speech detection task to a low power co-processor and using the main processor on the phone only when needed.

Most related to our interest in on-the-go speaker identification, two systems, Darwin [7] and EmotionSense [12], have recently explored speaker identification using mobile phones. Similar to both of these systems, our goal is not to design new speaker identification algorithms. Instead we leverage well-established techniques such as the MFCCs feature set [20], pitch tracking [11], and GMM classifiers [e.g., 13, 14], which have been proven effective for speaker identification. Our focus is on adapting these techniques to a mobile platform and addressing challenges that arise when using speaker identification on energy constrained mobile phones.

EmotionSense is a platform for social psychology research that aims to continuously sense the emotions of the mobile phone owner. EmotionSense includes a speaker recognition sub-system and silence detector that was used to select non-silent audio for the speaker recognition sub-system. Our work complements the EmotionSense research by exploring different approaches to address challenges in continuous audio sensing. For example, EmotionSense gathers training data offline in an explicit setup phase, while we have focused on gathering training data during everyday use. To extend battery life, EmotionSense offloads computation to a remote server, while we explore using an on-board low power processor. Finally, EmotionSense uses

Bluetooth identifiers from other phones to narrow down the number of possible speakers, while in SpeakerSense we focus on the case where the user's phone must work independently without relying on anyone else to be running the same program.

Darwin [7] uses speaker identification as the example application to demonstrate a collaborative sensing platform that uses data collected across many phones to evolve classifiers, share them across multiple phones, and then collaboratively infer state (e.g., who is speaking). The authors demonstrate the potential for speaker identification using multiple phones in three experimental scenarios. Again our work is related but complementary; we focus on improving speaker recognition on a phone running independently, which does not need or assume many phones running the same system. Our research to improve speaker recognition on a single phone could contribute to overall improvements in Darwin-style collaborative approaches.

3 SpeakerSense

To unobtrusively identify the people a user interacts with face to face, SpeakerSense must run continuously on the phone, sampling and processing audio to detect human speech from other sounds, and attempt speaker identification when speech is detected. A naïve continuous speaker identification algorithm would place a heavy burden on the battery and computational resources of the phone, since the high audio sampling rate prevents the phone from going into the sleep mode, and the speaker identification process is itself computationally expensive. For example, based on our measurements, running speaker identification continuously on an HTC Touch Pro 2 (TP2) phone consumes ~771mW, which is considerably larger than the phone's idle power consumption (~11mW) when the phone is asleep and no application is running. In this section, we first introduce a heterogeneous multi-processor mobile phone architecture that can support low power continuous sensing, and then show how we can build a speaker identification system on it.

3.1 SpeakerSense Architecture

We designed SpeakerSense so that we can continuously run speaker identification without significantly impacting the phone's battery life. On current phones, individual sensor energy consumption is generally very low, but the process by which the data is *read* requires the phone's main processor to be active, which has high energy requirements. Furthermore, the high sampling frequencies (e.g 8~16KHz) required for continuous sensing do not allow enough time for the phone's main processor to sleep between sampling cycles, leading to a state of continual high-energy consumption. Recent work has shown that energy requirements for continuous background sensing can be significantly reduced using an HMP architecture, where sampling and processing of sensor data is offloaded to a low-power processor [10]. The power consumption of a modern low-power processor is similar to that of a typical sensor; and due to the simple architecture, a low power processor can make transitions between sleep and active modes very quickly.

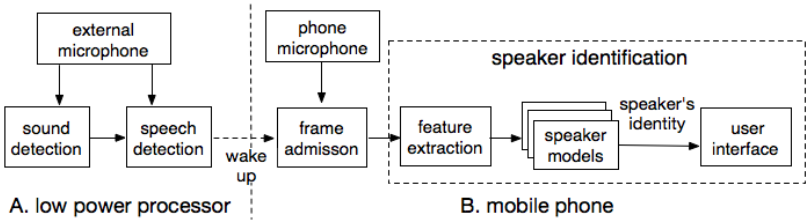

Fig. 1. The SpeakerSense architecture.

SpeakerSense uses an HMP architecture where the phone's main processor is augmented with a low power MSP430F5438 processor. This processor consumes 15mW and 6μW when active (18MHz) and sleeping respectively, and has a 5μs wakeup time. We attach a typical mobile phone microphone (Knowles Acoustics model SPM0408HD5H) and an amplifier to this processor for sampling audio signals. To further reduce energy consumption, we use a hardware threshold detection circuit to detect the presence of sound, where a signal is activated when the audio level goes above 14% of the full audio scale.

The low-power continuous audio sensing feature of the HMP allows SpeakerSense to leverage the fact that a phone's owner is likely engaged in conversations only a small fraction of the day. Thus, to reduce overall power and computing resource requirements, we designed SpeakerSense as a sequence of 4 stages that have increasing power requirements, and assigned each to the processor most appropriate for the energy consumption required (Figure 1). Each stage activates the higher power stage "on demand." Using the low-power processor, the Sound Detection stage first detects the presence of audio. Next, Speech Detection detects the presence of speech and wakes the phone when needed. On the phone, the Frame Admission stage identifies blocks of audio samples that contain high quality speech frames, which are sent to the Speaker Identification stage. We next describe the functional components of SpeakerSense.

3.2 Sound and Speech Detection

In SpeakerSense, the Sound and Speech detection stages run on a low-power processor. Sound Detection periodically examines the threshold detector output to determine the presence of a strong audio signal. If a strong audio signal is detected more than 50 out of 1000 times within a 0.5s period, the Sound Detection stage assumes the presence of a strong audio signal, and starts the Speech Detection stage on the microcontroller to detect the presence of voice.

The Speech Detection stage samples the audio signal at 8kHz and divides it into frames. Each frame has 256 samples, corresponding to 32ms of data. For each frame, two lightweight time domain features that summarize the sound characteristics are extracted: zero crossing rate (ZCR) [15] and root mean square (RMS) [16]. ZCR, defined as the number of zero-crossings within a frame, is an approximation for the pitch of the voice. Non-speech frames have no inherent pitch, resulting in high ZCR values, while frames with speech, particularly with vowels, typically have low ZCR values. The equation for computing ZCR is:

$$ZCR = \frac{\sum_{i=2}^{N} |sign(s_i) - sign(s_{i-1})|}{2}$$

where $s_{(i = 1 \ldots N)}$ represents the samples in the frame and N is the length of the frame,(256 in this example). The value of *sign(x)* is +1 or -1 depending on whether *(x* > 0*)* or *(x* < 0*)*. For example, if two consecutive samples have opposite signs, indicating a zero crossing, ZCR increases by 1. RMS represents the energy of the sound signal, and is defined as:

$$RMS = \sqrt{\frac{\sum_{i=1}^{N} {s_i}^2}{N}}$$

Since floating point calculations incur high processing overhead on most low power CPUs, we use a simple integer approximation as an alternative given by:

$$RMS_{approx} = \frac{\sum_{i=1}^{N} |s_i|}{N}$$

For every window of 64 frames (approximately 2 seconds of data), Speech Detection extracts window-based features and performs classification procedures. Human speech is characterized by rapidly changing fluctuations resulting from the interleaving of consonants and vowels, which other types of sound are less likely to exhibit [15]. Using the ZCR and RMS features for each frame, we calculate four window-level features to capture the sound patterns in a window: the mean of ZCR, the variances of ZCR RMS, and the Low Energy Frame Rate, which is defined as the number of frames within the window that have an RMS value less than 50% of the mean RMS for the entire window [16]. For efficiency, we approximate the variances of ZCR and RMS as follows:

$$VAR_{approx} = \frac{\sum_{i=1}^{N} |s_i - \mu|}{N}$$

Once the features are extracted, an offline-trained decision tree classifier decides whether the sound is human speech. If speech is detected, the Speech Detection stage wakes up the phone to run the final two stages that we describe next.

3.3 Frame Admission and Speaker Identification on the Phone

SpeakerSense runs frame admission and speaker identification on the phone.

Frame admission. The role of the frame admission stage is to pick high quality speech frames from the sampled audio and discard low quality speech frames as well as silence frames that occur naturally from brief pauses in human speech. Human speech can be divided into voiced speech and unvoiced speech [15]. Voiced speech is defined as speech generated from vibrations of the vocal chords, and includes all the vowel sounds. In contrast, unvoiced speech does not involve the vocal chords, and generally includes the consonant sounds. Because there is more energy in voiced speech than in unvoiced speech, voiced speech tends to be more resilient to background noise. Thus to increase robustness within the phone context and to improve overall system efficiency, we use a frame admission policy in SpeakerSense that only forwards voiced speech for speaker analysis, skipping pauses (silence) and unvoiced frames.

Frame admission is accomplished using thresholds on two metrics, ZCR, which we also use in the Speech Detection stage, and spectral entropy [3]. In the frequency domain, voiced frames have a series of strong peaks in the spectrum, corresponding to the pitch and formant of the voice, which results in low spectral entropy. On the other hand, the spectrum of unvoiced frames or non-speech frames is fairly flat, and yield relatively high spectrum entropy. Unlike standard speech processing systems, we do not adopt any commonly used energy features here, such as RMS [16], because the energy of the sampled audio is sensitive to the context of use (e.g., distance from the speaker, orientation of microphone etc.), while frequency-related features are relatively robust to environment conditions.

Speaker Identification. The speaker identification stage is based on a Gaussian Mixture Model (GMM) classifier [13] with some modifications to improve robustness and computation efficiency for use on the mobile phone. We use as our feature vector pitch [11] and the Mel-frequency cepstral coefficients (MFCCs) [20] computed for each admitted frame. SpeakerSense computes 20-dimensional MFCCs, and then ignores the first coefficient, which represents the energy of the frame, and instead focuses on the spectral shape, represented by the 2^{nd} through 20^{th} coefficients. While most traditional speaker identification systems use frames that overlap by 50% in order to capture subtle changes in the voice, this approach leads to sections of frames being analyzed multiple times. Given the resource constraints of the phone, we use non-overlapping frames to reduce the amount of computation required.

As is typical, we use a smoothing window, a fixed-length of successive frames, to improve system performance. In theory, a longer smoothing window increases the system performance. However, we have found that in practice, the smoothing window needs to be less than 10s, due to the latency introduced by the smoothing window and the uncertainty of turn-takings in conversations. The speaker identification algorithm attempts to identify a speaker by matching GMM speaker models to each smoothing window. Each speaker's GMM model estimates the log likelihood of the speaker, given every frame's feature vector. Assuming independence of frames, the speaker identified as the one talking during the window is the one whose speaker model gives the highest sum of log likelihood across the smoothing window. We train the GMM speaker models offline on a server. We use a universal background GMM to represent all unknown speakers [13]. To reduce the impact of noisy training data, the variance limiting technique [14] is applied with a standard expectation maximization (EM) algorithm [13] to train the GMM speaker models.

We have our low-powered processor attached to a prototype phone, so we decided to validate Frame Admission and Speech Detection on off-the-shelf phones for a better indication of feasibility and performance on currently available hardware. We implemented prototypes on an HTC Touch Pro 2 (TP2) and an Apple iPhone 3Gs. The HTC TP2 runs Microsoft Window Mobile 6.5; the iPhone 3Gs runs Apple iOS 3.1.3. All signal processing and classification algorithms are implemented in approximately 1000 lines of C code to achieve high efficiency and portability between different platforms. Other system components (e.g., UI, communication, etc.) are written in the default languages for each phone, C# for the TP2 and Objective C for the iPhone 3Gs. The offline server side training code is implemented primarily in Matlab.

In our implementation, the prototype pipeline is optimized for lower CPU usage at the cost of a larger memory usage. Whenever applicable, we pre-compute parameters offline, serialize them into configuration files, and load them into the memory when the application initializes. For example, in our prototype implementations, the GMM models directly use the pre-computed inverse and determinant of the covariance matrix as the model parameters rather than the covariance matrix itself.

4 Evaluation

We evaluated several aspects of SpeakerSense including efficiency and trade-offs between computational cost and performance for system parameters. We first describe our data collection methodology and then our experiments.

4.1 Data collection

For the evaluations we collected voice data from 17 speakers (10 males, 7 females) using the two mobile phones on which we implemented our prototype (TP2 and iPhone 3Gs). We were motivated to seek participants with a range of ages due to our interest in memory assistance applications where people would likely be interacting with family members of many different ages. The age distribution of our participants included four participants older than 45, four between 30-45, seven between 15 and 30, and two under 15. The median age was 29 and mean was 34.

For each speaker, we collected approximately 10 minutes of voice both locally and remotely though a phone call. The local recording was done by the two mobile phones simultaneously in both 8kHz 16bit mono and 16kHz 16bit mono formats. The phone call was recorded only in 8kHz, the phone channel standard. All the speech was collected in a normal office/home environment where the participant's voice was the dominant signal and there was modest background noise, such as air conditioning, desktop computer fan, or people talking remotely. All phones were placed within 1.5 meters from the subject's head. For each of the adult participants, we also collected 5 minutes of conversation with the researcher. This gave us a 15-person (eight male, seven female) one-to-one conversation database.

To train the Sound and Speech Detection stages on the low-powered processor, we also collected common ambient noises found in office and home settings. These sounds included the rubbing sound made by a phone in a pocket, the sound of walking, keyboard typing, mouse clicking, printers, copy machines, different types of fan noise, air conditioners, flowing water, street traffic, vacuuming, and various types of music. This ambient noise dataset contained approximately 250 minutes of audio.

4.2 Trade-offs between Computational Cost and Accuracy

For continuous audio sensing and speaker identification on mobile phones there are a number of possible trade-offs for system parameters that affect computational cost and accuracy. With SpeakerSense we investigated: the sampling rate and length of

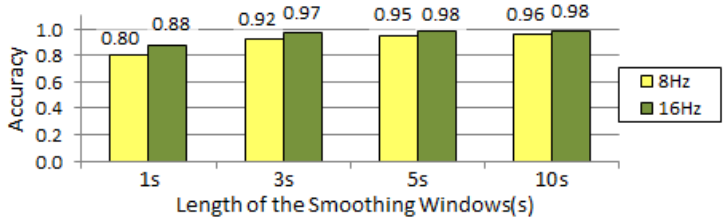

Fig. 2. SpeakerSense accuracy based on sampling rate and length of the smoothing windows.

the smoothing window, the amount of training data needed, and number of GMM components to use.

Sampling rate and length of smoothing window. A higher sampling rate captures more high frequency characteristics of the speaker's voice, improving the speaker identification accuracy at the cost of computation overhead. In Figure 2, we compare two audio sampling rates, 8kHz and 16kHz with different smoothing lengths. In this experiment, we use 10-fold cross validation. For each of the 17 speakers, the original 10-minute voice sample is partitioned into 10 subsets, nine of which are used for training and one for testing. The cross-validation process is repeated 10 times and then the average accuracy is calculated, with each of the 10 subsets used exactly once as the test data.

The benefit of a higher sampling rate is shown clearly in Figure 2. The accuracy on the 16kHz samples is consistently better than the 8kHz samples, although it doubles the amount of data to be processed. The performance difference between the two sampling rates is most apparent when the smoothing window is small, and the advantage of the higher sampling rate decreases as the length of the smoothing window increases. We therefore observe that the accuracy of speaker identification is less sensitive to sampling rate when longer smoothing windows are used. The choice of the window length dictates how fast the system responds to changes in conversations (e.g., onset of the voice and speaker turn-taking). Considering the turn-taking patterns of everyday conversation, we believe that 3s or 5s smoothing windows represent a good balance between delay and accuracy. Therefore for applications that require quick response and high accuracy, for example realtime memory assistance, a higher sampling rate with smaller smoothing window setup would be preferred. In contrast, for applications that are insensitive to latency or require higher efficiency, such as life logging, an 8kHz sampling rate with longer smoothing window would be more appropriate.

Amount of training data. Intuitively, identification accuracy increases as the amount of training data increases, because the speaker model can likely encode more information. However, collecting properly labeled training data is generally expensive and cumbersome (we discuss ways to minimize this burden in Section 5). In this experiment, we investigate the minimum amount of data needed for reliable performance. Figure 3 shows the accuracy results for the 8kHz and 16kHz dataset respectively. The training data samples were sequentially taken from our voice datasets for training. For both datasets, the accuracy for 1, 3, 5, and 10 second smoothing windows are given for training data up to 360 seconds.

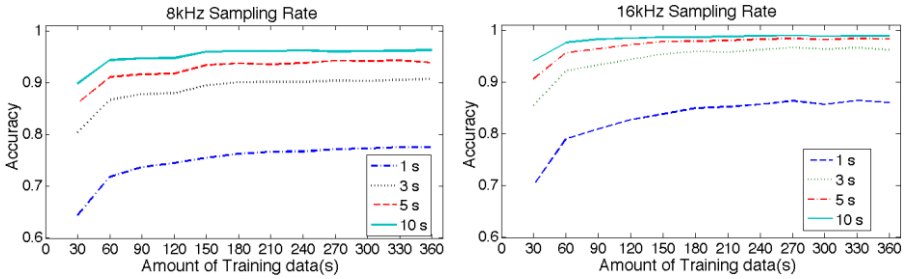

Fig. 3. Accuracy vs. amount of training data for the 8kHz (left) and 16kHz (right) datasets.

We can make several observations based on the results. First, there is a sharp increase of accuracy between 30s to 60s of training data and then accuracy levels off above 120s for 8kHz and 180s for 16kHz. These values are likely lower bounds on the amount of training data necessary to adequately model the speakers in our mobile phone setting. Second, the performance of smaller smoothing windows shows the fastest improvement. Longer smoothing windows are less sensitive to the increase of training data. Lastly, the 16 kHz system seems to benefit from more training data as accuracy increases up through 180 seconds. The reason may be that the training phase needs to model a wider frequency range, so it benefits from more data to learn from. Our experiments suggest for both 8 kHz and 16 kHz, 180 seconds (3 minutes) is a good minimum amount of training data.

The number of GMM mixture components. There is no well-established way to determine the optimal number of GMM components to model a speaker adequately. Using too few components results in an oversimplified speaker model, that is not sufficient to encode the characteristics of a speaker's voice. On the other hand, using too many components leads to a complicated speaker model with a large number of parameters, which requires a large amount of data and computation to train and makes the classification on the phone costly. Our goal is to choose the minimum number of components necessary to adequately model the speakers. Shown in Figure 4, we investigated 5 different model complexities: 4, 8, 16, 32, and 64 component GMMs with 1, 3, and 5 second smoothing windows. We used 180 seconds of training data based on evaluation of the amount of training data necessary.

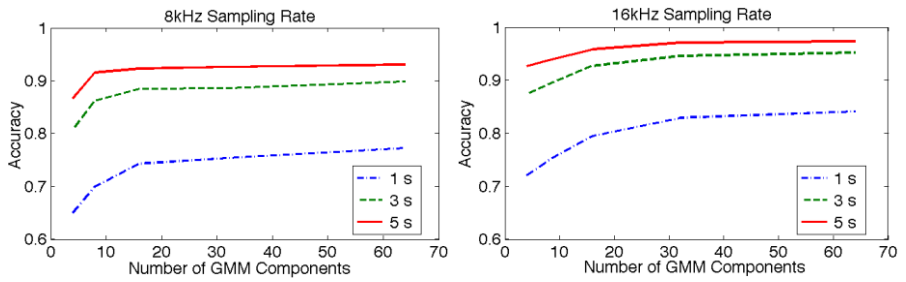

Fig. 4. Accuracy vs. number of GMM components for 8kHz (left) and 16kHz (right) datasets.

With more GMM components, identification accuracy improves as expected. The accuracy for shorter smoothing windows has bigger performance gains. The greatest increase in performance occurs for all smoothing window sizes when the number of GMM components grows from 4 to 16. Above 32 GMM components, the accuracy gain levels off. Since the computation cost of identifying the speaker is proportional to the number of GMM component, we believe 32 components is a good choice to balance accuracy and efficiency.

4.3 Efficiency of SpeakerSense

We now evaluate the processing overhead and power consumption of SpeakerSense as well as the accuracy of speech detection on the MSP430 low-powered processor.

SpeakerSense Processing Overhead. On the low-powered processor, Sound and Speech Detection consume very little processing resources. Sound Detection runs as a periodic task with a 1.5μs processing time every 500μs, using only 0.3% of the processing cycles. The Speech Detection stage takes 90.9ms to process 2s of audio data, resulting in only 4.4% processor utilization.

On the phone, we evaluated the processing overhead of Frame Admission and Speaker Identification using the speaker models of the 17 participants. In the experiments, we ran 18 GMMs in parallel (17 speakers plus 1 universal speaker model for unknown speakers) to perform speaker identification. All experiments sample at 16kHz, 16-bit, mono audio from the phone's built-in microphone. Based on the experiments described in the previous sub-section, we use a 32-component GMM and a 3s smoothing window.

The TP2 uses 24.8ms to process a 32ms frame of voice data when the full pipeline is engaged (i.e., processing voiced frames). The iPhone 3Gs, which has a slightly faster processor, takes 21.7ms to process a frame. Using the profiling tool provided in the iPhone SDK, we benchmarked the resource usage of different processing stages, shown in Table 1. The memory usage stays at about 8.25 MB, including the user interface that displays the current speaker, since we preload all the models and pre-allocate the memory required for processing. We can see that when the pipeline is fully engaged, SpeakerSense uses less than 50% of the CPU, which leaves enough resources for other applications to run concurrently.

In summary, we see the full advantage of using a HMP architecture. The Sound and Speech detection stages, which likely dominate the execution time in typical usages, consume very little processor resources on the low-power processor. Frame Admission and Speaker Identification on the phone, which run only when active conversations are detected, require more resources, but still leave computing resource for other concurrent applications.

Table 1. CPU usage for Speaker Identification on the iPhone 3Gs prototype.

Pipeline Stage:	Idle	Silence	Feature Extraction	Speaker Model Evaluation
CPU Usage:	<1%	<3%	5.6% ~ 11.3%	35.1% ~ 44.9%

Table 2. HTC TP2 Power Consumption.

Pipeline Stage:	Sampling Audio	Frame Admission	Speaker ID	Total
Average Power (mW):	335	21	415	771

Table 3. Confusion matrix of the accuracy of the voice and ambient noise classifiers.

Ground Truth \ Classified As	**Voice**	**Ambient Noise**
Voice	85.36%	14.64%
Ambient Noise	7.28%	92.72%

SpeakerSense Power Consumption. Energy life is a scarce resource on mobile phones so the power consumption of SpeakerSense must be small for continuous speaker identification to be practical. First, we measured the power consumption of Sound and Speech detection stages on the low-power processor. We found that the Sound Detection stage consumes 0.73mW on average, which is an order of magnitude smaller than the 11mW idle power consumption of TP2. So, when the environment is quiet, SpeakerSense does not add noticeable burden to the battery. The Speech Detection stage consumes 4.29mW, which is still much less than TP2's idle power.

To evaluate the power consumption of the Frame Admission and Speaker Identification stages, we measure the power consumption of TP2 using the Power Monitor tool [9] (the iPhone hardware does not allow this type of measurement). Table 2 shows the total power consumption under different operating conditions with a dimmed backlight. We first observe that just continuously sampling the audio on TP2 consumes 335mW, which highlights the advantage of using a low-power processor for sampling audio for sound and speech detection. The Frame Admission stage adds 21mW, which is much smaller than the overhead due to audio sampling. The infrequently invoked Speaker Identification routine adds 415mW to the average power consumption for a total of 771mW.

Accuracy of Speech Detection on Low-Powered Processor. We have shown that introducing the low-power processor results in considerable energy savings compared to using the current mobile phone architecture. However, we also need to validate that the Speech Detection on the low-powered processor is reasonably accurate, since an efficient but incorrect detection is useless. Table 3 shows the confusion matrix for Speech Detection on the low-powered processor (these numbers implicitly evaluate both Sound and Speech Detection since Sound Detection triggers Speech Detection).

The precision of the Speech Detection stage—a decision tree classifier with depth 7—is fairly high, around 93%, at the expense of a low recall at 85%. The latter is due to the fact that during pruning, the classifier is tuned to keep the false positive rate low in order to reduce the chance of waking up the phone unnecessarily. Although more sophisticated and demanding voice detectors can achieve better performance [e.g., 3, 16], we prefer the current design that has very low resource consumption.

5 Training Speaker Models

Speaker recognition requires training a speaker model for each individual who needs to be recognized. Acquiring the necessary training data is a practical challenge that is often ignored in research prototypes, where researchers collect the data required for training. Collecting training data directly from participants, typically by recording a single participant's voice in a quiet environment, yields the best speaker models because the audio data is of high quality and contains data from a single known person. However, this method is labor intensive and requires participants to contribute their voice and time to train a model. While people might be willing to provide explicit training data to help a loved one with memory assistance (e.g., a grandparent or parent), this explicit training step is unappealing in general.

In Section 4.2 we showed that to build a robust speaker model, SpeakerSense needs at least 3 minutes of voice data from the speaker. In this section, we explore three additional ways beyond an explicit training phase to acquire training data for the people that a user interacts with in everyday life: using phone calls, one-to-one conversation, and by sharing speaker models across phones. We discuss the advantages and disadvantages of each method, as well as additional processing required, and compare performance to the "gold standard" of user contributed data for the 17 participants we collected data from.

5.1 Using a Phone Call to Train a Model

Using data collected during phone calls to train speaker models has many potential advantages. First, the identity of the person talking on the phone is typically known using caller identification. Second, the audio generated by the phone's owner and the caller is automatically segmented because the voices pass through two different audio channels: the owner's voice is received by the local microphone while the caller's voice is received over the telephone network.

There is no question that phone calls are an excellent way to train the speaker model for the phone's owner. The user's own voice is recorded locally by the mobile phone's microphone and can be used directly for training without any additional processing. The most striking difference between audio recorded during a phone call and user contributed data is that speech from phone calls will exhibit large segments of silence or background noise when the caller is talking. However, because SpeakerSense is already designed to select only those segments that contain voiced speech for training, breaks in the dialog are handled automatically.

Training speaker models for the caller is more challenging. The caller's voice is sampled by the phone and transmitted through the phone network where it undergoes band limit filtering and some spectral shaping because the telephone line is a narrowband communication channel. This makes it problematic to directly use the audio recording from another caller as input to the speaker model training since there is a mismatch between the narrowband speech data gathered from the phone line and the wideband testing data recorded by the phone's microphone. To determine if it is feasible to recover useful training data for a caller from a phone call, we investigated

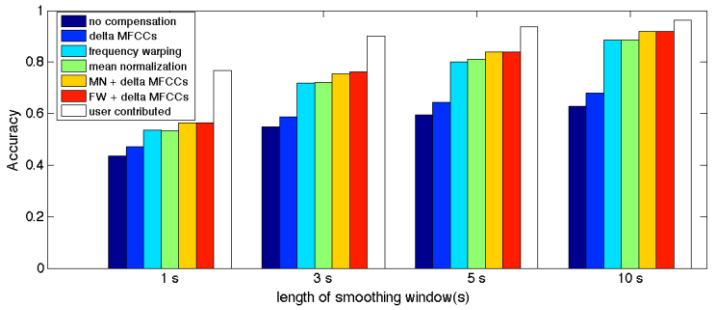

Fig. 5. Comparisons of channel compensation methods for different length smoothing windows.

three lightweight channel compensation techniques to address the acoustic distortion produced by the telephone network: phone frequency warping [14], mean normalization [18], and the delta MFCCs features [13].

The average speaker identification accuracy for all 17 participants using the different compensation techniques is shown in Figure 5. Assuming the phone is able to provide precise caller identification information, in the experiment we manually labeled the speaker's identity for each of the mobile phone call recordings. We trained the speaker models using 5-minute call recordings and tested on 5 minutes of clean locally recorded voice data from our 8kHz dataset to match the phone channel standard. We used a 32-component GMM and the variance limiting was set to 0.1 in order to be more robust to the noisy channel. For the frequency warping method, the typical phone channel bandwidth of 300~3300 Hz was linearly warped to the full bandwidth 0~4000 Hz. When using delta MFCCs features, 19 difference coefficients from a 32ms interval around the current frame are used. This introduces a 32ms delay, but is negligible compared to the smoothing window delay.

It is clear from Figure 5 that without further processing, the performance of speaker identification was vastly reduced by using recorded telephone speech directly for training. Even with a 10s smoothing window, the accuracy is still poor, coming in at only 63% accuracy as compared to greater than 94% when using user contributed data. When used individually, frequency warping and mean normalization are nearly equally effective, with both contributing more than a 20% accuracy gain when the smoothing window is 3s or greater. Adding delta MFCCs to the feature vector produced a moderate 5% increase in accuracy when used alone. However, when Delta MFCCs was combined with frequency warping or mean normalization, it contributed an additional 4% performance gain. Because combining mean normalization with frequency warping provided no significant improvement over the use of each method individually, we omitted this combined condition from the figure. Although our experiments found frequency warping and mean normalization to be equally effective, we recommend using frequency warping over mean normalization for mobile applications because frequency warping is applied to each frame independently, which avoids maintaining a running average that introduces delay into the pipeline.

Based on this analysis, our prototype system uses frequency warping and delta MFCCs together when handling phone recorded speech. Given the ease of collecting training data using phone calls, the trade-off of slightly reduced accuracy rates compared to user contributed data seems worthwhile. It is important to note that while

we have addressed the technical feasibility of gathering training data from phone calls, the cultural and legal acceptability of training speaker models based on phone calls requires further investigation.

5.2 Using One-to-One Conversations to Train Speaker Models

Another possibility for training speaker models is to collect voice data from everyday conversations, particularly those between the phone owner and one other person. While more practical than asking people to explicitly contribute training data, using data recorded during a one-to-one conversation is more complicated than using phone calls because the audio collected is not automatically labeled with the person speaking, and the recorded data likely includes speech from both the phone owner and the unknown speaker.

To process data recorded in one-to-one conversations, SpeakerSense requires the phone owner to manually mark the start and end of the (entire) conversation and enter the name of the other speaker. SpeakerSense then applies an automatic segmentation method that runs the phone owner's speaker model and the universal model on all the voiced frames. Intuitively, the user's own voice will be identified correctly, and the other speaker's voice will be marked as an unknown speaker. Once enough data from the other speaker has been accumulated, the system can train a model accordingly.

To analyze the effectiveness of this automatic segmenting approach for harvesting training voice data from one-to-one conversations we used the conversations that we collected with each of the 15 adult participants. Each conversation was sampled at 16kHz by the phone, and the start and end was marked by the researcher whose speaker model was pre-obtained. Informed by our previous experiments we used 32-component GMMs. Figure 6 compares the accuracy of speaker models trained using 5 minutes of conversation data and 5 minutes of user contributed data.

As expected, the speaker models trained from user contributed data has better accuracy, since it starts with perfectly segmented voice data containing only one speaker. In contrast, we expected that our algorithm for automatically selecting audio segments for a second speaker would not be able to perfectly segment the conversation and thus that some number of the audio segments used in training the speaker model may contain voiced data from the phone owner or both speakers. However, as Figure 6 shows, accuracy for the speaker models trained using the conversation data only marginally lag behind the speaker models trained on user

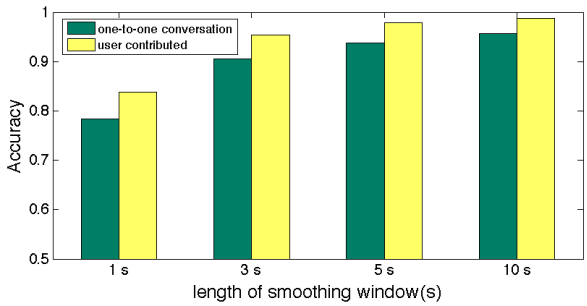

Fig. 6. Speaker models trained on one-to-one conversations compared to user contributed data.

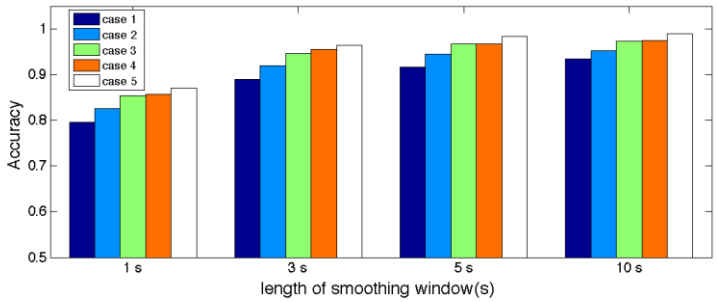

Fig. 7. Performance of sharing models between phones. Case 1: training with iPhone and testing with TP2. Case 2: training with TP2 and testing with iPhone. Case 3: training with iPhone and testing with TP2 with delta MFCCs features. Case 4: training with TP2 and testing with iPhone with delta MFCCs features. Case 5: training and testing with the same phone.

contributed data. The difference is consistently less than 5% for all smoothing lengths. This shows that using automatically segmented conversation data could be a reliable and practical source for training data, especially given the reduced user effort necessary compared to getting user contributed data.

5.3 Sharing Speaker Models Between Phones to Train Speaker Models

The final speaker model training solution that we explored was sharing speaker models between phones. This approach was suggested by Darwin [7], one of the only other systems we are aware of that considers the challenge of collecting training data. In Darwin's speaker identification prototype, each phone learns a speaker model for its owner and then exchanges its model with other phones, in a process termed 'model pooling.' We were intrigued by the feasibility of exchanging speaker models across different types of devices and how well speaker models trained on one device would work on another device, which was not described by [7]. If possible, sharing speaker models would greatly reduce the training effort because each person would only need to train their own speaker model.

The main challenges in sharing speaker models are the differences between the microphones on the devices used to capture the original audio. Although the built-in microphones on mobile phones are usually optimized for human voice, their frequency responses can differ. We hypothesized that using a channel normalization technique, such as using delta MFCCs features, could reduce the effect that different microphones have on the data. Using the 16kHz iPhone and HTC TP2 datasets, we conducted experiments to explore the impact of microphone variation when sharing speaker models between different phones and the effectiveness of applying delta MFCCs features to reduce the effect of the differences.

As Figure 7 shows, the mismatch between the training and testing microphone when sharing models between the two phones decreased the system accuracy by about 7%. However, using a longer smoothing window helps reduce the negative impact of using different microphones. Furthermore, adding the delta MFCCs

features, which are insensitive to the microphone differences, improves the accuracy by about 4%. Thus our results from the TP2 and iPhone suggest that combining delta MFCCs features with models trained on different phones yields in an overall accuracy loss of only 3%. While it would be worth validating this small loss across more phones, we believe it is likely that speaker models can be shared across phones with only a negligible performance loss. It is important to note that while we have evidence that it is technically feasible to share speaker models across phones, user acceptance of sharing or exchanging models has not been investigated by Darwin or our research and is important to consider moving forward.

6 Conclusions and Future Work

Our research with SpeakerSense has addressed two challenges for using continuous audio sensing for speaker identification: efficient performance that enables continuous audio sensing and scalable methods for gathering the training data needed for speaker models. Through our experiments with data gathered from 17 participants in an indoor office environment, we have identified trade-offs between computational cost and performance that enable robust speaker identification on mobile phones by evaluating sampling rate options (8 kHz vs. 16 kHz), the length of the smoothing window, the number of GMM components needed to model a speaker adequately, and identified lower-bounds on the amount of training data needed to construct robust speaker models for the phone.

To address efficiency we prototyped SpeakerSense using HMP hardware. We demonstrate that splitting computation across a dedicated low-power processor that detects sound and voice and using the phone's main processor to run the computationally intensive speaker identification pipeline only when necessary enables continuous and efficient speaker identification. We believe other continuous sensing applications could benefit from a similar hardware-based approach. Lastly, we presented and evaluated methods for gathering the training data necessary for constructing speaker models during everyday activities, identifying channel compensation methods that make it feasible to gather training data from phone calls, and an automatic segmentation method for training speaker models using one-to-one conversations. Furthermore, we validate the feasibility of sharing speaker models between different phone platforms.

Our research has addressed many technical issues necessary to make continuous audio sensing and speaker identification practical, enabling the future work needed to study SpeakerSense, and similar continuous sensing applications, in day-to-day use. Field deployments will be valuable to test our approaches for gathering training data in real use, to gather data about the performance of SpeakerSense across a variety of environments, and to evaluate whether the information provided by SpeakerSense can provide memory assistance for people with memory impairment in practice. We are excited about the potential pervasive computing applications that we believe are enabled by our research.

References

1. Hayes, G., Patel, S., Truong, K, Iachello, G., Kientz, J., Farmer, R., Abowd, G., The Personal Audio Loop: Designing a Ubiquitous Audio-Based Memory Aid. Proc. Mobile HCI 2004.
2. Hodges, S., Williams, L, Berry, E., Izadi, S., Srinivasan, J., Butler, A., Smyth, G., Kapur, N., Wood, K., SenseCam: A Retrospective Memory Aid. Proc. UbiComp 2006. pp. 177-193.
3. Huang, L., and Yang, C., A Novel Approach to Robust Speech Endpoint Detection in Car Environments, in ICASSP '00, Istambul, Turkey, May 2000, vol. 3, pp. 1751-1754.
4. Kapur, N. Compensating for Memory Deficits with Memory Aids. In: Wilson, B (ed.) Memory Rehabilitation Integrating Theory and Practice Guilford Press. pp. 52 – 73.
5. Lee, M. and Dey, A., Lifelogging Memory Appliance for People with Episodic Memory Impairment. Proc. UbiComp 2008. pp. 44-53.
6. Lu, H., Pan, W., Lane, W., Choudhury, T., and Campbell. A., SoundSense: scalable sound sensing for people-centric applications on mobile phones. Proc. MobiSys 2009, pp. 165–178.
7. Miluzzo, E., Cornelius, C., Ramaswamy A., Choudhury, T., Liu, Z., Campbell, A., Darwin Phones: the Evolution of Sensing and Inference on Mobile Phones. Proc. MobiSys 2010, pp 5-20.
8. Miluzzo, E., Lane, N., Fodor, K., Peterson, R., Lu, H., Musolesi, M., Eisenman, S., Zheng, X., and Campbell. A., Sensing meets mobile social networks: The design, implementation and evaluation of the CenceMe application. Proc. SenSys08, pp. 337–350.
9. Power Monitor, http://www.msoon.com/LabEquipment/PowerMonitor/
10. Priyantha, B., Lymberopoulos, D., and Liu, J., LittleRock: Enabling Energy Effcient Continuous Sensing on Mobile Phones. IEEE Pervasive Computing Magazine, April- June 2011.
11. Rabiner, L. R., Cheng, M. J., Rosenberg, A. E. and McGonegal, C. A. A comparative performance study of several pitch detection algorithms, IEEE Trans. Acoust., Speech, and Signal Processing, pp. 399-418, Oct. 1976.
12. Rachuri, K., Musolesi, M, Mascolo, C., Rentfrow, P., Longworth, C., Aucinas, A., EmotionSense: A Mobile Phone based Adaptive Platform for Experimental Social Psychology Research. Proc. UbiComp 2010, pp. 281-290.
13. Reynolds, D. A. An Overview of Automatic Speaker Recognition Technology, Proc. Int. Conf. Acoustics, Speech, and Signal Processing, vol. 4, pp. 4072 - 4075, 2002.
14. Reynolds, D.A. and Rose, R.C. Robust text-independent speaker identification using Gaussian mixture speaker models, IEEE Trans. Speech Audio Process. 3 (1995), pp. 72–83.
15. Saunders, J., Real time discrimination of broadcast speech/music, Proc. Int. Conf. Acoustics, Speech, Signal Processing (ICASSP), pp. 993 - 996, 1996.
16. Scheirer, E., and Slaney, M., Construction and evaluation of a robust multifeature speech/music discriminator, Proc. ICASSP-98, May 1998.
17. Vemuri, S., Schmandt, C., Bender, W., iRemember: a Personal, Long-term Memory Prosthesis. Proc. CARPE 2006.
18. Viikki, O., and Laurila, K., Cepstral domain segmental feature vector normalization for noise robust speech recognition, Speech Communication, vol. 25, pp. 133–147, 1998.
19. Wang, Y., Lin, J., Annavaram, M., Jacobson, Q., Hong, J., Krishnamachari, B., and Sadeh N., A framework of energy efficient mobile sensing for automatic user state recognition. Proc. MobiSys, pp. 179–192.
20. Zheng F., Zhang, G., Song, Z., Comparison of different implementations of MFCC, J. Computer Science & Technology, 16(6):582-589, Sept. 2001.