SoundSifter: Mitigating Overhearing of Continuous Listening Devices

Md Tamzeed Islam
UNC at Chapel Hill
tamzeed@cs.unc.edu

Bashima Islam
UNC at Chapel Hill
bashima@cs.unc.edu

Shahriar Nirjon
UNC at Chapel Hill
nirjon@cs.unc.edu

ABSTRACT

In this paper, we study the overhearing problem of continuous acoustic sensing devices such as Amazon Echo, Google Home, or such voice-enabled home hubs, and develop a system called *SoundSifter* that mitigates personal or contextual information leakage due to the presence of unwanted sound sources in the acoustic environment. Instead of proposing modifications to existing home hubs, we build an independent embedded system that connects to a home hub via its audio input. Considering the aesthetics of home hubs, we envision SoundSifter as a smart sleeve or a cover for these devices. SoundSifter has hardware and software to capture the audio, isolate signals from distinct sound sources, filter out signals that are from unwanted sources, and process the signals to enforce policies such as personalization before the signals enter into an untrusted system like Amazon Echo or Google Home. We conduct empirical and real-world experiments to demonstrate that SoundSifter runs in real-time, is noise resilient, and supports selective and personalized voice commands that commercial voice-enabled home hubs do not.

1. INTRODUCTION

Having reached the milestone of human-level speech understanding by machines, continuous listening devices are now becoming ubiquitous. Today, it is possible for an embedded device to continuously capture, process, and interpret acoustic signals in real-time. Tech giants like Apple, Microsoft, Google, and Amazon have their own versions of continuous audio sensing and interpretation systems. Apple's Siri [10] and Microsoft's Cortana [16] understand what we say, and act on them to fetch us a web page, schedule a meeting, find the best sushi in town, or tell us a joke. Google and Amazon have gone one step further. Android's 'OK Google' feature [13], Amazon's Echo [2], and Google Home [8] devices do not even require user interactions such as touches or button presses. Although these devices are activated upon a hot-word, in the process, they are continuously listening to everything. It is not hard to imagine that sooner or later someone will be hacking into these cloud-connected systems and will be listening to every conversation we are having at our home, which is one of our most private places.

Furthermore, there is a recent trend in the IoT world that many third-party, commercial IoT devices are now becoming voice enabled by using the APIs offered by the voice-controlled personal assistant devices like Echo or Google Home. Henceforth, we will refer to these devices interchangeably as smart hubs, home hubs, or simply hubs. For example, many home appliances and web services such as Google Nest thermostats [9], Philips Hue lights [15], Belkin WeMo switches [5], TP-Link smart plugs [19], Uber, and Amazon ordering are now 'Alexa-Enabled' – which means, we can send voice commands to an Amazon Echo device to actuate electrical devices and home appliances. Because it enables actuation and control of real-world entities, a potential danger is that they can be activated by false commands (e.g., sounds from a TV) and/or unauthorized commands (e.g., an outsider commands someone's home hub to control his home appliances, places a large purchase on his Amazon account, or calls a Uber driver). A careful scrutiny of voice commands is therefore a necessity to ensure safety and security.

Unfortunately, none of the existing voice-enabled home hub devices take any of these vulnerabilities into account while processing the audio. They merely apply standard noise-cancellation [22, 50, 52] to suppress non-speech background sounds in order to improve the signal-to-noise ratio. However, this process alone cannot eliminate acoustic signals from unwanted acoustic sources that happen to be present in the environment and overlap in time and/or frequency with a user's voice signals. There is also no support for personalization of speech commands in these devices.

In this paper, we study the 'overhearing' problem of acoustic sensing devices, and develop a system that mitigates personal or contextual information leakage due to the presence of unwanted sound sources in the acoustic environment. Instead of developing a special-purpose, application-specific embedded system that works only for voice commands, we address the problem in a generic setting where a user can define a specific type of sound as primary (i.e., a relevant or essential sound type for the application), or secondary (i.e., a non-essential and potentially privacy concerning sound). For example, the voice of a user issuing a command is a primary source for home hubs, whereas any identifiable background noises such as the sounds from appliances, other conversations are examples of secondary sounds. Furthermore,

MobiSys'17, June 19-23, 2017, Niagara Falls, NY, USA

DOI: http://dx.doi.org/10.1145/3081333.3081338

instead of proposing modifications to existing home hubs, we build an independent embedded system that connects to a home hub via its audio input. Considering the aesthetics of home hubs, we envision the proposed system as a smart sleeve or a cover for these home hubs. The proposed system has necessary hardware and software to capture the audio, isolate signals from distinct sound sources, filter out signals that are from unwanted sources, and process the signals to enforce policies such as personalization before the signals enter into an untrusted system like Amazon Echo or Google Home. The device is programmable, i.e., an end user is able to configure it for different usage scenarios.

Developing such an embedded system poses several challenges. First, in order to isolate acoustic sources, we are required to use an array of microphones. Continuously sampling multiple microphones at high rates, at all times, and then processing them in real-time is extremely CPU, memory, and time demanding for resource-constrained systems. Second, to train the system to distinguish primary and secondary sources, we are required to collect audio samples from an end user to create person or context specific acoustic models. To the users, it would be an inconvenience if we require them to record a large number of audio samples for each type of sound in their home. Third, because no acoustic source separation is perfect, there will always be residuals of secondary sources after the source separation has been completed. These residuals contain enough information to infer personal or contextual information, and hence, they must be eliminated to ensure protection against information leakage.

In this paper, we address all these challenges and develop a complete system called the *SoundSifter*. The hardware of SoundSifter consists of a low-cost, open-source, embedded platform that drives an array of microphones at variable rates. Five software modules perform five major acoustic processing tasks: to orchestrate the sampling rates of the microphones, to align the signals, to isolate sound sources, to identify primary source, and to post process the stream to remove residuals and perform speaker identification. At the end of the processing, audio data is streamed into the home hub. We thoroughly evaluate the system components, algorithms, and the full system using empirical data as well as real deployment scenarios in multiple home environments.

The contributions of this paper are the following:

- We describe SoundSifter, the first system that addresses the overhearing problem of voice-enabled personal assistant devices like Amazon Echo, and provides an efficient, pragmatic solution to problems such as information leakage, and unauthorized or false commands due to the presence of unwanted sound sources in the environment.
- We devise an algorithm that predicts a spectral property of incoming audio to control the sampling rates of the microphone array to achieve an efficient acoustic source separation.
- We devise an algorithm that estimates noise directly from the secondary sources and nullifies residuals of secondary signals from the primary source.
- We conduct empirical and real-world experiments to demonstrate that SoundSifter runs in real-time, is noise resilient, and supports selective and personalized voice commands that commercial voice-enabled home hubs do not.

2. BACKGROUND

We provide background on source separation, a specific source separation algorithm, and terminology that is used later in the paper.

2.1 Source Separation

The term *'blind source separation'* [26] refers to the generic problem of retrieving N unobserved sources only from the knowledge of P observed mixtures of these sources. The problem was first formulated to model neural processing of human brains [34], and has later been extended and studied in many other contexts such as biomedical applications [42, 60], communication [29, 27], finance [23, 25], security [49, 48], and acoustics [61, 33, 65]. To the acoustic processing community, this problem is popularly known as the *'cocktail party problem,'* where sources represent human voices.

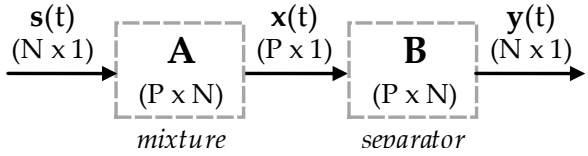

Figure 1: Generic model for source separation.

In this paper, all sources are acoustic, and each microphone observes a weighted combination of N sound sources. Assuming $\mathbf{s}(t) = (s_1(t), ..., s_N(t))^T \in \mathbb{R}^N$ denotes the sources, $\mathbf{x}(t) = (x_1(t), ..., x_P(t))^T \in \mathbb{R}^P$ denotes the observed mixtures, $(.)^T$ stands for matrix transpose operation, and $\mathbf{A}$ denotes an unknown mapping from $\mathbb{R}^N$ to $\mathbb{R}^P$, we can write: $\forall t \in \mathbb{Z} \quad \mathbf{x}(t) = \mathbf{A}\mathbf{s}(t)$.

The equation above is of a *linear instantaneous* model, which is the most commonly used model for source separation. This does not explicitly model noise, as it can be implicitly modeled as an additional source. In Figure 1, $\mathbf{A}$ is a *mixing* matrix that mixes N acoustic sources to produce P output streams. Retrieving the sources is equivalent to finding $\mathbf{B}$, the inverse or *separator* matrix. The separated outputs are expressed as: $\forall t \in \mathbb{Z} \quad \mathbf{y}(t) = \mathbf{B}\mathbf{x}(t)$. When $N \leq P$, $\mathbf{A}$ is invertible. But for $N > P$, additional assumptions (e.g., sparsity [21]) may be required.

2.2 Fast ICA

There are many solutions to the source separation problem that make different assumptions about sources and use different mixing systems [40, 25, 32]. *Independent Component Analysis* (ICA) is one of the most popular solutions. This approach assumes that the acoustic sources are statistically independent from each other – which is in general true for our application scenario. For example, voice commands to an Amazon Echo device and unwanted background sounds are unlikely to have any statistical correlations among themselves.

Fast ICA [38] is a popular, efficient independent component analysis-based source separation algorithm. It isolates the sources by iteratively maximizing a measure of mutual independence among the sources. In Fast ICA, *non-Gaussianity* [26] of the sources is taken as the measure.

Using matrix notation, $\mathbf{x}(t)$ is expressed as $X \in \mathbb{R}^{P \times T}$. FastICA iteratively updates a weight vector $W \in \mathbb{R}^P$ to maximize non-Gaussianity of the projection $W^T X$ using the

following two steps in a loop:

$$\begin{aligned} \mathrm{W}^{+} &\leftarrow \mathrm{E}\{\mathrm{X}\phi(\mathrm{W}^{\mathrm{T}}\mathrm{X})\} - \mathrm{E}\{\phi'(\mathrm{W}^{\mathrm{T}}\mathrm{X})\}\mathrm{W} \\ \mathrm{W} &\leftarrow \mathrm{W}^{+}/\ ||\mathrm{W}^{+}|| \end{aligned} \quad (1)$$

Here, $\phi(\mathrm{x}) = \tanh(\mathrm{x})$, $\phi'(\mathrm{x})$ is its derivative, and E{} is average over columns of a matrix. W is initialized to a random vector, and the loop stops when there is no significant change in it. Note that, for simplicity, we only show how to isolate one source in Equation 1; for multiple sources, this needs to be repeated for each source. We also omit the preprocessing steps that involves prewhitening [26] matrix X.

2.3 Measure of Residual Signals

Because no source separation is perfect, there are always residues of secondary sources within the isolated stream of audio that is supposed to carry signals from the primary source only. We use a metric to quantify this residual with the following equation:

$$\xi_i = ||x_i(t) - y_i(t)||_2 \quad (2)$$

Here, ξ_i denotes the amount of residuals in the i^{th} source after source separation, which is expressed as the l^2-norm of the difference between primary signals before and after source separation. We use this metric in our evaluations to quantify the quality of source separation.

3. OVERVIEW OF SOUNDSIFTER

SoundSifter is motivated by the need of a smart acoustic filter that inspects audio signals and takes proper actions such as filtering sounds from unwanted secondary sources and checking the content of primary signals before letting them into a voice-enabled home hub or any such continuous listening devices.

SoundSifter connects to a home hub or a mobile device's audio jack, and can be thought of as an extension to their on-board audio I/O subsystem. It captures and processes all incoming audio streams, isolates audio signals from distinct sources, identifies and blocks out any sound that has not been labeled 'primary' by a user during its installation, and only lets processed audio enter into a hub for further application-specific processing.

3.1 Basic Workflow

Figure 2(a) shows how SoundSifter sits between audio sources and a home hub. Further details of its internal processing blocks are shown in Figure 2(b). The system has an *Audio I/O Controller* that controls an array of microphones (required for source separation), a speaker, Bluetooth and an audio jack to support external audio I/O. The *Source Separator* executes the acoustic source separation algorithm by controlling the microphones via the audio I/O controller and using precomputed acoustic models. The *Post Processing* module further filters and obfuscates the already separated primary stream to eliminate traces of residuals from other sources and to enforce policies (read from a configuration file) such as personalized commands. The policies are further described in 3.3 as usage scenarios. Finally, the processed audio output is let go into the home hub via the audio jack.

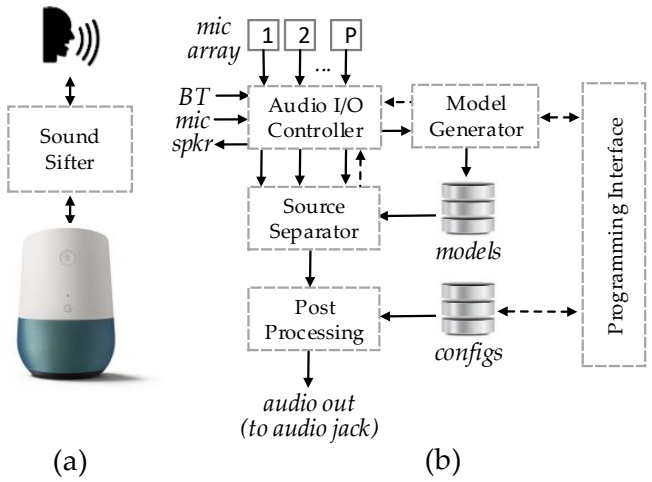

Figure 2: (a) System interface, and (b) block diagram of SoundSifter.

3.2 Initial Setup and Programming

SoundSifter needs to be programmed once for each use case (described in the next section). Programming the device essentially means creating acoustic models for each type of *primary* sound involved in a scenario. Because these sounds are often user- or home-specific, this process requires the active engagement of a user. To make the process simple, a user is provided with a mobile app that is as easy as using a media player. The mobile app connects to SoundSifter over Bluetooth, and interacts with it by sending programming commands and receiving responses. *No audio data are exchanged between the devices.*

In the app, the user is guided to send commands to SoundSifter to record $10s - 30s$ audio for each type of primary sound, label them, and specify in which scenario they will be used. The system also needs some examples of a few common types of secondary sounds for the application scenario. However, it does not require a user to record and label all possible secondary sounds. We empirically determined that as long as SoundSifter has 2-3 types of secondary sounds per use case, it is capable of detecting primary vs secondary sounds with a high accuracy and a negligible false positive rate.

Once the recording and labeling phase is over, SoundSifter uses an algorithm to create acoustic models, deletes all raw audio data, and only the labeled models are stored inside the device in a configuration file.

3.3 Usage Scenarios

We describe two motivating scenarios for SoundSifter that are specific to voice-controlled home hubs.

Voice-only mode: Continuous listening devices of today hear everything in their surrounding acoustic environment. The goal of *voice-only* mode is to ensure that only speech signals enter into these devices while all other sound sources in a home environment such as sounds from TV, appliances, non-voice human sounds such as laughter, crying, coughing, sounds of an activity such as cooking, cleaning, etc. are completely filtered out of the system. This is achieved in SoundSifter by using a combination of source separation, recognition, and suppression.

Personalization: Command personalization may be of multiple types: voice commands containing an exact sequence of words or an utterance, voice commands that contain a certain keyword or a set of keywords in it and voice commands of a certain person. These are achievable by first applying the voice-only mode, and then performing additional acoustic processing such as speech-to-text and speaker identification. Although we only mention home hub related usage scenarios of SoundSifter in this section, the generic notion of primary and secondary sound allows us to configure the system for other applications where it can filter in/out different primary/secondary sounds as well. Furthermore, we did not implement a speech-to-text converter in our system due to time constraints, which we leave as future work.

4. STUDYING THE PROBLEM

Prior to the development of SoundSifter, we performed studies and experiments to understand the nature of the challenge.

4.1 Need for Source Separation

As an alternative to source separation, we looked into simpler solutions such as filtering and noise cancellation. However, those attempts failed since the sounds that may be present in the environment overlap with one another in the frequency domain.

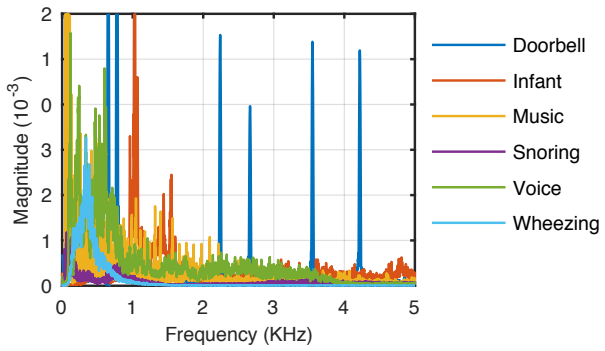

Figure 3: Frequency plots of variety of sounds.

In Figure 3 we plot frequency characteristics of a selected set of sounds. For example, speech (< 4 KHz) covers the full range of snoring (< 1.5 KHz) and asthmatic wheeze (< 1.3 KHz). Crying infants and doorbell sounds range from 500 Hz to 1.6 KHz and 4.2 KHz, respectively. Music overlaps with all sounds. Besides these, we also analyzed home appliances such as a blender, a washing machine, door slams, toilet flushes, speech signals of different sexes and age groups, and asthmatic crackling, and came to the conclusion that spatial information is the most effective method for identifying and isolating primary information containing signals from other types of unwanted sounds in a general purpose setting.

4.2 Number of Microphones

For effective source separation, we are required to use an array of microphones. In theory, the number of microphones should equal the number of simultaneously active sources. However, for sparse sources like audio (sources that do not produce a continuous stream) , source separation can be performed with fewer microphones.

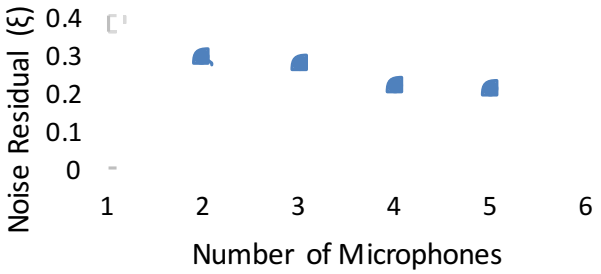

Figure 4: Source separation error for different number of microphones.

To determine an adequate number of microphones for source separation, we perform an experiment where we use an array of five microphones that captures audio signals from 2–5 simultaneously active sources: voice (primary sound), ringing phone, piano, songs, and television. Figure 4 shows the quality of source separation in terms of mean residuals (the lower the better) as we vary the number of microphones for different number of active sources. We observe that the residuals drop with more microphones. However, the reduction is not great for more than four microphones. Hence, we use four microphones in our system and use an additional noise removal step to nullify the remaining residuals.

4.3 Benefit of Rate Adaptation

According to the Nyquist theorem [55], the sampling rate of each microphone must be at least twice of the maximum frequency of any source. However, sampling an array of microphones at a very high rate costs significant CPU, memory, and power consumption. Note that our system is suitable for portable home hub devices, where a plug-in power is not always an option. In such cases, limited processing power is a challenge.

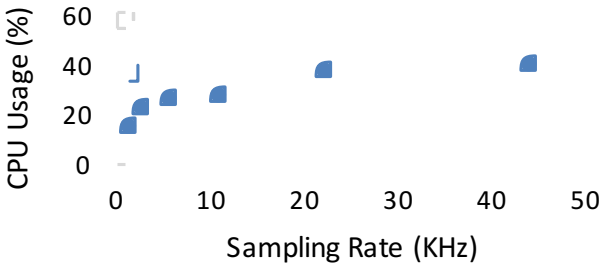

Figure 5: CPU usage increases with sampling rate.

To validate this, we conduct an experiment using an ARM Cortex A8-based microcontroller. In Figure 5, we observe that as the sampling rate is increased, CPU usage increases sharply. Memory consumption also increases from 22 KB to 704 KB as we vary the sampling rate. Based on this observation, we decide not to sample the microphones at the highest rate at all times; instead, we adopt a predictive scheme where we probabilistically choose a sufficient sampling rate for the microphone array based on previous knowledge on sound sources and signals that the system has just seen.

4.4 Modeling Sounds

After isolating the sources, for SoundSifter to determine which sounds to let in and which ones to block, it has to

identify the source that represents the primary sound. Because primary sounds in different usage scenarios are highly subjective, SoundSifter must obtain a sufficiently large number of audio samples directly from the end user to create a robust and accurate sound classifier. On one hand, we need a large amount of training audio from the user for a robust classification. On the other hand, requiring a user to collect these data is likely to be error prone and also an inconvenience to them. Hence, it is customary to investigate techniques to create robust acoustic classifiers that generate accurate and robust models based on a limited amount of training data.

4.5 Need for Residue Removal

A crucial observation during our study has been that even after source separation, when we look into the stream of primary signals, we find traces of secondary sources. We realize that even though the residues are too weak to be heard, using machine learning, they can be identified and recognized.

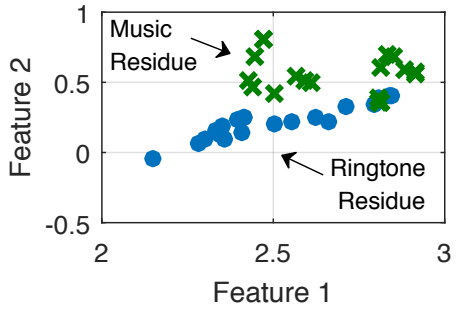

Figure 6: Effect of residue on primary source.

To illustrate the vulnerability of signal residue, we conducted a small scale study. We performed source separation for two cases – speech with: 1) music, and 2) a ringing phone in the background. In both cases, the same set of 20 utterances is synthetically mixed (to keep the primary sound identical) with the two background sounds. After source separation, we take the separated speech streams, compute Mel-frequency cepstral coefficient (MFCC) [57] features, and plot the utterances (total 40) in a 2D feature space as shown in Figure 6. In this figure, feature 1 and feature 2 denote the two highest ranked components of MFCC. It is interesting to observe that even though the residues of music and ring tone are not audible, their presence in the speech stream is statistically significant – which is enough to distinguish the two cases.

4.6 Analog Data Acquisition

A practical engineering challenge that we face with multi-channel audio data acquisition has been the inability of commodity embedded hardware platforms to sample analog signals at a very high rate. For example, considering the worst case where we are required to drive four analog microphones at 44.1 KHz, we need an aggregate sampling rate of 176.4 KHz. Achieving such a high rate of analog reading using off-the-shelf Linux-based embedded platforms such as Arduinos, Raspberry Pis, or Beaglebones is non-trivial. We address this implementation challenge and believe our solution will be helpful to anyone who wants to read analog audio at a high (e.g. MHz) rate.

5. ALGORITHM DESIGN

The audio processing pipeline inside SoundSifter has five major stages. Figure 7 shows the stages and their interconnections. The first two stages prepare the audio streams from the microphone array for the source separation stage. These two stages together implement the proposed sampling rate adaptation scheme in order to lower the CPU and memory consumption of SoundSifter. We use FastICA [38] to perform the actual source separation. The last two stages perform further processing to identify the primary source and to nullify the residuals of other sources in it.

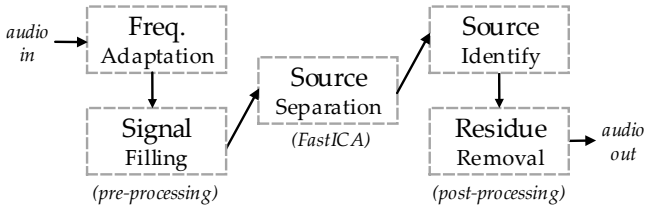

Figure 7: Components and interconnections inside SoundSifter's audio processing pipeline.

In this section, we describe the first and the last two stages of SoundSifter's audio processing pipeline, i.e. the pre-processing and post-processing stages to demonstrate how these stages work in concert to improve the efficiency and effectiveness in mitigating information leakage from unwanted acoustic sources with SoundSifter.

5.1 Frequency Adaptation

SoundSifter adopts a predictive scheme to determine an adequate sampling rate for the microphone array. Sampling rate adaptation in SoundSifter happens periodically, and the predicted rate is estimated based on the knowledge of the portion of the audio that the system has already seen. SoundSifter keeps an evolving Markov [53] model, whose transition probabilities help determine the expected maximum frequency of incoming audio, given the measured maximum frequencies of the audio samples received during the last few ms.

Predicting the absolute value of the maximum frequency of incoming audio signals is practically impossible in a generic setting. However, if we divide the full range of audible frequencies into discrete levels, the transitions from one level to another can be predicted with a higher confidence. For this, SoundSifter uses six ranges of frequencies by dividing the maximum sampling rate into six disjoint sets $44.1/2^h$ KHz, where $0 \leq h \leq 5$. We denote these by the set $\{f_i\}$, where $0 \leq i \leq 5$.

Furthermore, since we are aiming at applying the *past predicts the future* principle, a question that immediately comes up is how much past data to use for an accurate prediction of the desired sampling rate? To determine this, we conduct an experiment to quantify the look-back duration. We experiment with two different Markov models:

- **1-Step Look-back:** Uses a 6-state Markov model, where each state corresponds to a frequency in $\{f_i\}$, resulting in

a 6×6 transition matrix having transitions of the form $f_i \rightarrow f_j$.

- **2-Step Look-back:** Uses a 36-state Markov model, where each state corresponds to a pair of frequencies (f_i, f_j), resulting in a 36×36 transition matrix having transitions of the form $(f_i, f_j) \rightarrow (f_j, f_k)$.

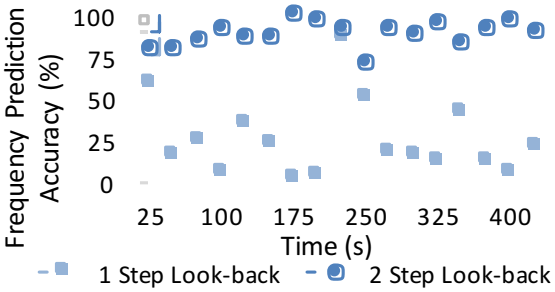

Figure 8: 2-Step Look-back performs better than 1-Step Look-back for loud music

Figure 8 shows a comparison of these two frequency prediction approaches for an 8-minute recording of loud music. Sampling frequency is adapted every 500 ms, compared with the ground truth, and the prediction accuracy is reported every 25 seconds. We observe that the 2-step look-back, i.e. the 36 state Markov model is a significant improvement over the 1-step look-back and has an average prediction accuracy of 88.23% for this very challenging case. Higher than 2-step look-back might show a slightly better accuracy, but such a model would result in an explosion of states. Hence, we use the 2-step look-back model in SoundSifter.

The proposed Markov model evolves over time. Initially, all transition probabilities in the model are set so that the microphones are sampled at the highest rate. As the model starts to make predictions, the probabilities are updated following a simple reward and punishment process where every correct/incorrect predict is rewarded/punished by increasing/decreasing the probability. Both the processes of making a prediction and updating the probability are $\mathcal{O}(1)$ operations.

Ideally one would expect SoundSifter to adapt all its microphones to the minimum required frequency of the moment, but there is a small caveat. Because frequency prediction is not 100% accurate, there is a chance that occasionally we will have mispredictions. Cases when the predicted frequency is lower than the desired one, we will not be able to correct it in the next step unless we have an oracle. To address this, we keep the sampling frequency of one microphone fixed at 44.1 KHz, while the sampling rates of all other microphones are adapted as described above. In case of mispredictions, this microphone serves as the oracle and helps determine the correct frequency.

5.2 Signal Prediction and Filling

Adapting sampling rates at runtime has its benefits, but it also introduces an *alignment* problem during source separation. By default, standard source separation algorithms such as the FastICA assume that all microphones are sampled at a known fixed rate. In SoundSifter, this assumption does not hold anymore, i.e. different microphones may be sampled at different rates. Therefore, we are required to re-align samples from all the microphones and fill the gaps (missing samples) in each of the low rate microphones. Because we work with a small amount of signals at a time, although we expand the low rate samples, the process does not increase the total memory consumption significantly.

We illustrate this using two microphones: $x_i(t)$ and $x_j(t)$, supposing they are sampled at 44.1 KHz and 11.025 KHz, respectively. Within a certain period of time (e.g., between two consecutive frequency adaptation events), the first microphone will have four times more samples than the second microphone. If we align them in time, there will be three missing values in the second stream for each sample. If we assume a matrix of samples where each row corresponds to one microphone, the resultant matrix after alignment would look like the following:

$$\begin{bmatrix} x_{i1} & x_{i2} & x_{i3} & x_{i4} & x_{i5} & x_{i6} & x_{i7} & x_{i8} & x_{i9} & \dots \\ x_{j1} & ? & ? & ? & x_{j5} & ? & ? & ? & x_{j9} & \dots \end{bmatrix}$$

To fill the missing values in the matrix, we formulate it as an interpolation problem and try to predict the missing values in two ways:

- **Linear Interpolation:** Uses line segments to join consecutive points and any intermediate missing value is predicted as a point on the line segment.
- **Cubic Spline Interpolation:** Uses piece-wise third order polynomials [20] to construct a smooth curve that is used to interpolate missing samples.

The benefit of linear interpolation is that it is faster (e.g., 58 times when compared to cubic spline for one second audio) than its higher order counterpart, but it performs very poorly in predicting audio samples if the gaps between given points are large. Cubic spline produces smoother curves and performs comparatively better for large gaps in missing values, but its running time is slower. We pick linear interpolation and a suitable length for interpolation, so that it runs fast and is also fairly accurate.

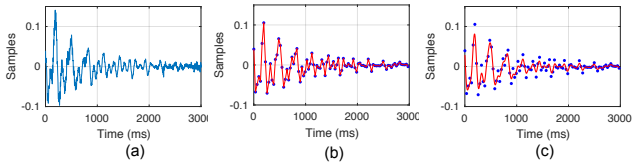

Figure 9: Predicting missing values of a signal(a) using linear(b) and spline(c) interpolation.

Figure 9 compares both linear and spline methods for signal interpolation to *raise up-sample* a signal from 1.378 KHz to 44.1 KHz. We observe that for 10 ms sample windows, linear interpolation produces faster and better results than cubic splines.

5.3 Modeling and Recognizing Sounds

After source separation, SoundSifter classifies each isolated source as primary or secondary. If we had sufficient training examples for each type of sound from an end user, the problem would be as simple as creating a standard machine learning classifier. However, to reduce the burden of data collection on a user and to improve the robustness of

the created classifier in a principled way, we perform an additional *data augmentation* step prior to creating the classifier model. Recall that this is a one time step that happens during the installation and programming of SoundSifter for a particular usage scenario.

5.3.1 Data Augmentation

The basic principle behind data augmentation [28, 44] is to generate new training examples from existing ones by perturbing one or more acoustic characteristics. In SoundSifter, we use two specific types of augmentation techniques to increase the user contributed data size by about 13 times. These techniques are listed in Table 1.

Action	Description
f-warping	Remapping the frequency axis: $f_i \rightarrow \alpha f_i$, where $\alpha \in [0.8, 1.2]$.
Inject Noise	Superimposing simulated noise. Noise models are created offline.

Table 1: Data augmentation techniques applied to increase user-contributed samples.

To apply frequency warping [41], each sample goes through a mapping phase 10 times, each time using a different α that is chosen uniformly at random. Simulated random noise [56] is applied to about one-third of these examples to further augment the data set. Overall, we obtain a 13.33-fold boost in the number of training examples giving us original samples as well as their perturbed versions that are resilient to noise and changes in frequency due to environmental effect.

5.3.2 Feature Extraction and Classification

For each audio frame, a 13-element MFCC feature vector is computed. Following common practice, a number of consecutive feature vectors are used to calculate 13 deltas and 13 double deltas to obtain 39-element feature vectors for each frame. Finally, the mean, standard deviation, and range of these vectors are taken to obtain a single 39-element feature vector. A random forest [36] classifier is used to create the final classifier.

For speaker identification, we extend the feature vector of the previous step to include *pitch* as an extra feature. Following [51], we estimate pitch using the *zero crossing rate* (ZCR) of the audio in the time domain:

$$ZCR = 0.5 \times \sum_{i=2}^{n} |sign(s_i) - sign(s_{i-1})| \quad (3)$$

Here, s_i represents audio samples, n is the length of a frame and $sign(x)$ is either $+1$ or -1 depending on whether $(x > 0)$ or $(x < 0)$.

5.4 Eliminating Secondary Residues

After source separation and source identification, we obtain a primary and a number of secondary sources. As seen previously in Figure 4, the residuals were not zero even in the best case, and in Figure 6 we observed that such residuals are significant enough to recognize secondary sources embedded within the isolated primary stream.

In order to nullify these residuals we employ a customized *adaptive noise cancellation* technique [62]. Standard noise cancellation algorithms either assume simple Gaussian noises or use sophisticated hardware to capture noise sources to 'subtract' noise spectra from the main audio stream. In SoundSifter, we are lucky to have the separated secondary source streams readily available as a by product of source separation. These secondary streams are used as negative feedback to remove their respective residues from the primary stream.

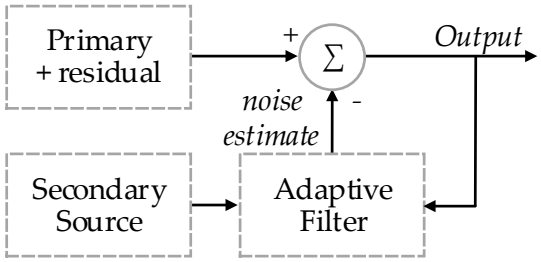

Figure 10: Leveraging isolated secondary sources to nullify their residue through a customized adaptive noise cancellation process.

Figure 10 shows the noise cancellation process where secondary sources are iteratively used to estimate residual noise signals contributed by each of them, which are then subtracted from the primary source to obtain residue-free signals. As an illustration, we consider p and n_i as the primary and secondary sources after the source separation step, respectively. An input to the noise canceller is $(p + n_i^r)$, where n_i^r denotes the residuals. The secondary source n_i is another input to the canceller that is passed through an adaptive filter to produce an output $\widehat{n}_i^r$ that is a close replica of n_i^r. The output of the adaptive filter is subtracted from the primary input to produce the system output $z = p + n_i^r - \widehat{n}_i^r$, which indicates the error signal for the adaptive process.

We adjust the filter using least mean squares (LMS) [63] algorithm that minimizes the output power. We use a stochastic gradient descent method where the filter weights are adapted based on the error at the current time step. The weight update function for the least mean squares algorithm is:

$$\mathrm{W_{n+1}} = \mathrm{W_n} - \mu \nabla \epsilon[\mathrm{n}] \quad (4)$$

Here, ϵ represents the mean-square error, and μ is a constant that controls the speed of convergence.

6. IMPLEMENTATION NOTES

Due to space limitations we only describe some key implementation issues.

6.1 Amazon Alexa Voice Service

To demonstrate SoundSifter, we must connect it to a home hub such as Amazon Echo or Google Home. However, at present, none of these devices come with an audio input where we can pipe in the processed audio from SoundSifter. Hence, we use Amazon Voice Service (AVS) to turn a Raspberry Pi into an Alexa-enabled device which provides us with multiple options for audio inputs, i.e. audio jacks as well as Bluetooth. We followed the step-by-step procedure [1] that Amazon recommends to enable their voice service API in Raspberry Pi platform. Figure 11 shows our custom home hub in a 3D printed case, which is functionally identical to an Amazon Echo device.

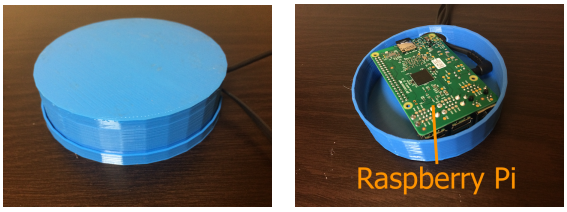

Figure 11: Alexa-enabled Raspberry Pi in a case.

6.2 SoundSifter in a BeagleBone Black

We have developed SoundSifter based on an open-source hardware platform called the Beaglebone Black [3]. An advantage of BeagleBone Black over other open platforms is that, besides the main ARM Cortex-A8 CPU, it has two additional cores known as the *programmable real-time units* (PRUs) [4]. Each PRU provides fast (200 MHz, 32-bit) real-time access to a number of I/O pins. This lets us sample multiple analog audio inputs at a very high rate.

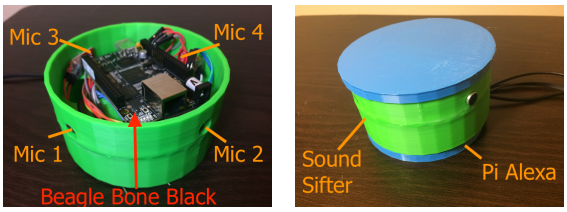

Figure 12: (Left) SoundSifter in its open case. (Right) the case sits on top of Alexa as a cover.

We use a PRU enabled BeagleBone Black as SoundSifter's processor. For the microphone array, we use four Electret microphones [17]. For aesthetics, we 3D-print a case that contains the complete SoundSifter system. In Figure 12 we show the SoundSifter inside a green open case (left). The lower half of this case is hollow, which allows us to put this on top of the Alexa device of Figure 11 as a cover (right).

6.3 Libraries

For Fast ICA we used the Modular toolkit for Data Processing (MDP) [12], which is a Python-based data processing framework. We use the Pandas [14] library for interpolation. We use libpruio [11] for PRUs, which is designed for easy configuration and data handling at high speed. To use this library, we load kernel driver uio_pruss and enable PRU subsystems by loading the universal device tree overlay [6].

7. EVALUATION

We perform three types of experiments. First, we evaluate the execution time, CPU, and memory usage of SoundSifter. Second, we evaluate the four key algorithms of SoundSifter using an empirical dataset. Third, we perform an end-to-end evaluation of different use cases of SoundSifter and compare the performance with an Amazon Echo device.

7.1 Microbenchmarks

Figure 13 shows execution times of five major components inside SoundSifter's audio processing pipeline for processing one second of audio. Frequency adaptation is a constant time operation that takes only 1 ms. It is essentially a matrix look-up operation in the transition matrix. Signal alignment and filling using linear interpolation takes 35 ms on average. As expected, the most time consuming operation in SoundSifter is the source separation step that takes about 179 ms. For source identification, SoundSifter takes 121 ms to calculate the features and 3 ms for classification. The last component of SoundSifter residue removal takes 54.68 ms. Overall, SoundSifter's execution time is 394 ms for processing 1 second audio, which means, the system runs in real-time.

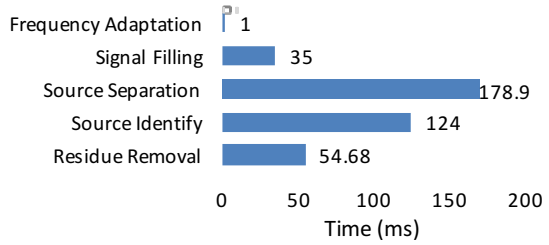

Figure 13: Run-time Analysis.

Table 2 lists the CPU and memory usage for the components of the audio processing pipeline. The most expensive operation is source separation. It requires 57.4% of the CPU and 5.2% memory. Other tasks such as source identification, signal filling, and residue removal use 50.8%, 33.6% and 11.5% of the CPU, respectively.

	CPU (%)	Memory (%)
Signal Filling	33.6	2.8
Source Separation	57.4	5.2
Source Identify	50.8	6.3
Residue Removal	11.5	2.5

Table 2: CPU and memory usage.

7.2 Algorithm Evaluation

To evaluate various components of SoundSifter, we use an empirical dataset which is segmented into three categories of sounds as described in Table 3. We collect this data in a 15 square feet room. For some of the analyses, such as (Figure 14, 16, 17, 19), where we want to evaluate the performance of the system at the maximum sampling rate (i.e. 44.1 KHz), we have used a multi-channel microphone setup [18]. This has allowed us to evaluate the proposed algorithms over a wider range of sampling rates. In the final prototype of SoundSifter, we, however, use Electret microphones [7] which have a frequency range of 20 Hz to 20 kHz. This is not an issue as long as the application, where SoundSifter is in use, deals with signals < 20 kHz. For example, in our chosen applications, the maximum frequency of any sound type in the environment is 10 KHz.

7.2.1 *Frequency Adaptation*

We compare SoundSifter's 2-step look-based frequency adaptation algorithm's accuracy with that of a 1-step look back Markov model in four test scenarios. These scenarios represent: 1) a person talking, 2) two-person conversation and

Dataset	Count	Examples	Length
Speech	150	conversations (1-10 persons)	200 *min*
Home	54	TV, phone, mouse, keyboard	75 *min*
Song	10	rock, country	55 *min*

Table 3: Description of the empirical dataset.

a TV in the background, 3) two-person conversation with a loud song in the background, and 4) two-person conversation while both TV and loud music are playing.

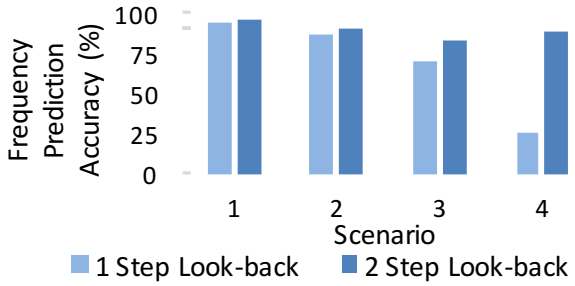

Figure 14: SoundSifter has higher accuracy than 1 Step Look-back for all scenarios.

In Figure 14, we observe that although the performance of both approaches drop as the scenarios become harder, the 2-step look-back model always performs better than the 1-step look-back method. The 2-step look-back model maintains an accuracy in the range of 88.24%-95.26%, whereas the 1-step look-back model's accuracy drops from 94.63% to 25.18%.

A reason behind SoundSifter's frequency adaptation was to reduce resource consumption of the embedded system. To evaluate this, we compare SoundSifter's CPU usage with that of a BeagleBone Black that is sampling four sensors at 44.1 KHz. We consider three scenarios demonstrating three different types of acoustic environments: 1) a noisy environment where a high-volume song is playing, 2) a living room where two persons are talking and a television is running, and 3) a relatively quiet environment where two persons are talking. The ranges of frequencies in these three scenarios are 11.03–5.51 KHz, 5.51–2.76 KHz, and ≤2.76 KHz, respectively.

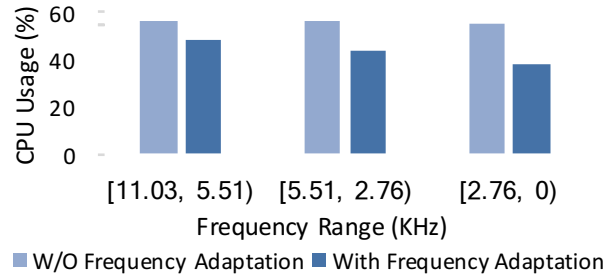

Figure 15: Frequency adaptation reduces CPU usage especially for lower frequency ranges.

Figure 15 shows that as the range of frequencies of the sound sources vary, because of the adaptive nature of SoundSifter, its CPU usage decreases from 48.3%-38.1%. The CPU usage of the BeagleBone remains fixed at 55.8% at all times.

7.2.2 *Signal Prediction and Filling*

We measure the performance of signal prediction and filling of SoundSifter by comparing its performance with a cubic spline-based interpolation scheme. We take a 44.1 KHz audio and down-sample it to produce five signal streams having the rates of 1.378 KHz, 2.756 KHz, 5.512 KHz, 11.025 KHz, 22.05 KHz, respectively. These are then up-sampled again to get back to 44.1 KHz, by using two interpolation methods: linear (as done in SoundSifter) and cubic spline. The quality of interpolation is measured in terms of their correlation with the original 44.1 KHz signals. We run this experiment on 150 speech clips of 200 minutes, 10 song clips of 50 minutes and four sound clips of 50 minutes.

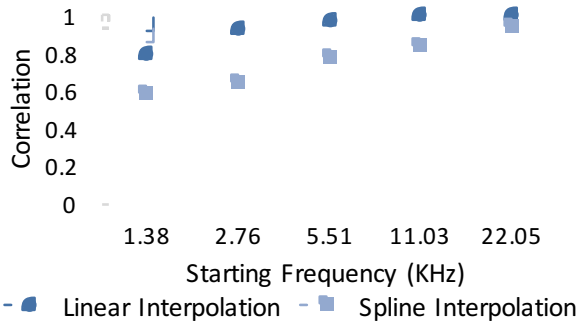

Figure 16: Linear Interpolation is more effective than Spline Interpolation for Signal Prediction and Filling.

Figure 16 shows that for 22.05 KHz, the predicted signals in SoundSifter show 98% correlation, whereas spline interpolation shows 93%. For lower frequencies, spline's performance drops significantly. SoundSifter's predicted signal shows more than 90% correlation even for 2.756 KHz. At 1.378 KHz, SoundSifter' correlation drops to 79%, which is still significantly higher than that of spline interpolation.

7.2.3 *Sound Modeling and Recognition*

We illustrate the performance of SoundSifter's sound modeling and recognition with the help of two scenarios: 1) voice-only mode, and 2) personalized mode.

For the voice only mode, we test the accuracy of primary and secondary source recognition after source separation, where speech is kept as the primary source.

In Figure 17, we observe that SoundSifter's data augmentation technique outperforms a classifier that is trained without data augmentation. For 2-3 sources, SoundSifter shows 89%-90.14% accuracy, whereas without data augmentation, a classifier achieves only 66%-70% accuracy. As the number of secondary sources increases, we see that without data augmentation the accuracy drops below 50% for 4-5 sources. SoundSifter's classification accuracy remains stable even when the number of sources is high. For 4-5 sources, SoundSifter achieves 89.13%-92.78% accuracy, and its false positive rate has always been 0%.

For the personalized mode, we want to recognize the particular user's commands and ignore anything else. For this, we collect 64 voice commands from three persons as primary sources and 53 voice commands from seven persons as

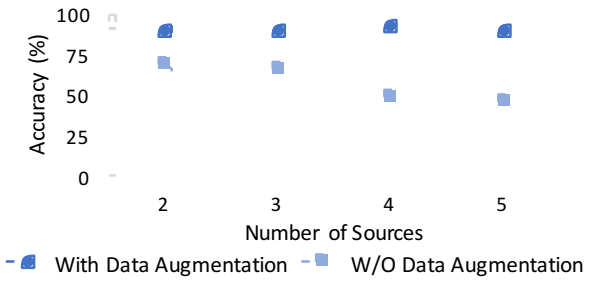

Figure 17: SoundSifter's data augmentation technique improves modeling accuracy

secondary sources. For primary speaker recognition, SoundSifter achieves 94.87% accuracy. From Figure 18, we see that SoundSifter is able to detect all the secondary users' commands, resulting in a 0% false positive rates. For primary speaker detection, SoundSifter was able to detect 58 out of 64 primary commands. Hence, this shows that SoundSifter is able to accept or reject commands based on personalization.

		Predicted: Primary	Predicted: Secondary
Actual	Primary	58	6
	Secondary	0	53

Figure 18: Confusion matrix for primary speaker recognition among up to 10 persons.

7.2.4 *Residue Removal*

We quantify the effectiveness of residue removal by showing that the residue in the separated signals is reduced after applying the process. For this experiment, we consider audio clips from a varying number of sound sources and apply Fast ICA with and without residual removal and compare the noise residual (ξ) we get from these two approaches.

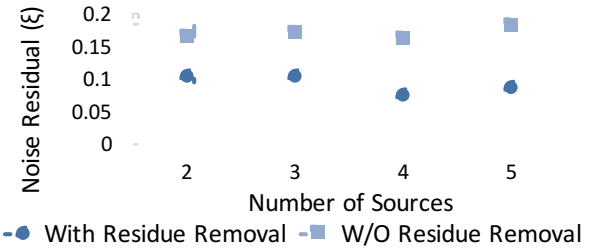

Figure 19: Residue Removal reduces noise residual from the primary signal.

From Figure 19, we see that for 2, 3, 4 and 5 sources Fast ICA without residual removal has 0.164, 0.169, 0.159 and 0.181 noise residual, whereas Fast ICA with residual removal has 0.102, 0.101, 0.073 and 0.084 noise residual, respectively. From this result, we deduce that using Fast ICA with residual removal has better performance for removing residue from the signal than using only Fast ICA.

7.3 Full System Evaluation

7.3.1 *Scenarios*

To demonstrate the effectiveness of SoundSifter, we compare its performance with an Amazon Echo device in three different scenarios. Table 4 lists the scenarios.

Scenario	Participants	Noise Level
Voice Mode (normal)	15	13 dB
Voice Mode (noisy)	10	51 dB
Personalized Mode	10	13 dB

Table 4: Description of the scenarios.

These scenarios are designed to illustrate SoundSifter's performance under different conditions and demands. In the first scenario, we consider a normal environment where only one person commands SoundSifter and Echo at the same time. In this experiment, we want to demonstrate that the source separation and residue removal steps do not damage the quality of the audio signals in SoundSifter. 50 commands from 15 different participants are used in this experiment.

For the second scenario, we consider a noisy environment. We ask our participants to issue commands while loud music is playing in the background. We want to show that due to source separation and residue removal, SoundSifter is more resilient to noise than Amazon Echo. We have collected 50 commands from 10 different participants for this scenario.

In the third scenario, we want to demonstrate that SoundSifter is personalized for a particular user and an intruder is not able to issue arbitrary commands to SoundSifter. Amazon Echo does not have this feature. For this experiment, we consider one of the ten participants as the primary user and the other nine users as intruders. 50 voice commands for both the primary user and the intruders are used in this experiment.

7.3.2 *Comparison with Amazon Echo*

From Figure 20, we see that in the first scenario, both SoundSifter and Echo are able to respond to all 50 commands. Hence, the source separation and residue removal steps of SoundSifter did not affect the quality of audio. In the second scenario, SoundSifter responds to 46 out of 50 commands, whereas Echo was only able to respond to 26 of them. This shows that SoundSifter is more resilient than Echo in a noisy environment.

In the third scenario, because Echo does not have any personalization support, it was unable to detect any intruders. On the other hand, from Figure 21, we find that SoundSifter is able to detect all 50 of the intruder's commands. This shows the strength of the special feature of SoundSifter that is not currently supported by any commercial home hub devices.

8. DISCUSSION

At present, home hub devices like Google Home and Amazon Echo do not have any audio input port where we can pipe in processed audio from SoundSifter. This is why we

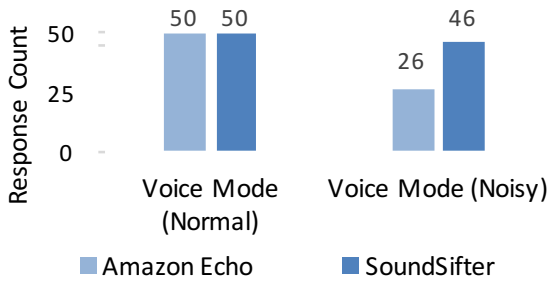

Figure 20: For noisy voice mode, SoundSifter was able to respond to 46 commands whereas, Echo responded to only 26 commands.

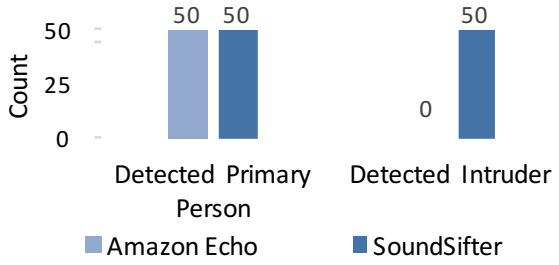

Figure 21: For personalized mode, SoundSifter was able to detect all 50 intruder commands, Amazon Echo fails to detect any of them.

had to develop a custom Amazon Echo clone using a Raspberry Pi. For our system to work seamlessly with a commercial home hub, either the device has to have an audio jack or it has to support open source software, so that our algorithms can be run on them.

In SoundSifter, the calculation of audio features takes a significant amount of computation time. Hence, the system is not fully ready for applications that require a very low latency. To address this issue, as future work, we aim to develop new acoustic features that are lightweight but similar to MFCC features in terms of performance.

The accuracy of the sound classifier depends on training, which is done by the end user who has to record audio samples of his voice commands and secondary sound types. This may be cumbersome to many users, and if the classifier is not trained sufficiently, the accuracy of primary source classification may drop. To address this issue, as a future extension, we plan to develop new classifiers that require a small number of training examples.

SoundSifter is not connected to the Internet, and audio processing happens inside the device. After processing, SoundSifter deletes the audio stream periodically. However, our system is secure as long as a hacker does not tamper with the device physically (e.g. by connecting extra wires to get raw data) or inject software Trojans (e.g., by altering the source code to dump the memory to files or by disabling/bypassing the algorithms during processing).

9. RELATED WORK

Solutions to the source separation problem under different assumptions on sources and mixing systems fall into three main categories. The first category is based on *Independent Component Analysis* (ICA) [40], where statistical independence of sources is exploited to estimate the sources. Although efficient implementations exist [38, 39], these algorithms have certain fundamental limitations, e.g., they support at most one Gaussian source, and do not exploit signal properties such as non-negativity or sparsity. The second category applies *Non-negative Matrix Factorization* (NMF) [25] to exploit non-negativity of real-world signals. However, their lack of statistical assumptions on data does not guarantee a correct decomposition of sources. The third type is based on *Sparse Component Analysis* (SCA) [32], which exploits sparsity in signals, e.g., acoustic signals. In this paper, we could have used any of the three methods, but we implement FastICA [39, 38] which is a proven and well-used algorithm.

In many digital systems, dynamic voltage and frequency scaling [47, 59, 24, 35, 31] are applied to scale up/down the clock frequency of the entire system to lower the power consumption. Our problem is significantly different and harder than that since we are required to dynamically adapt sampling frequencies of an array of microphones (each one is assigned a different rate) and make sure that the independent component analysis framework for source separation still works.

Standard practices toward dealing with limited training data fall broadly into two categories – *data augmentation* and *classifier fusion*, which are often used together. Data augmentation techniques [28, 44] generate new training examples by perturbing acoustic characteristics of existing examples, e.g., frequency warping [41], modifying tempo [43], and adding simulated reverberation [30], and noise [56]. In classifier fusion [58, 54], outputs of multiple classifiers are combined to improve the overall accuracy. These techniques vary from simple voting and averaging [45], to ranking [64, 37], to learning decision templates [46, 37]. In this paper, we only use data augmentation and avoided classifier fusion since that would require sharing and exchanging classifier models of different users – which may violate their privacy.

10. CONCLUSION

This paper describes a system that mitigates information leakage due to the presence of unwanted sound sources in an acoustic environment when using voice-enabled personal assistant devices like Amazon Echo. We developed new algorithms to make acoustic source separation CPU and memory efficient, and to remove residuals of unwanted signals from the main audio stream. The performance of the system has been compared with that of commercial continuous listening devices to show that it accurately filters out unwanted sounds and thus protects against personal and contextual information leakage where existing devices fail.

11. REFERENCES

[1] Alexa AVS. https://github.com/alexa/alexa-avs-sample-app/wiki/Raspberry-Pi.

[2] Amazon Echo: Always Ready, Connected, and Fast. www.amazon.com/echo.

[3] BeagleBone Black. http://beagleboard.org/black.

[4] Beaglebone black pru. http://beagleboard.org/pru.

[5] Belkin WeMo Switch. http://www.belkin.com/us/Products/home-automation.

[6] Device tree overlay. https://github.com/beagleboard/bb.org-overlays.

[7] Electret mic. https://www.adafruit.com/product/1713.

[8] Google Home. https://madeby.google.com/home/.

[9] Google Nest. https://nest.com/.

[10] iOS SiRi from Apple. http://www.apple.com/ios/siri/.

[11] libpruio. http://users.freebasic-portal.de/tjf/Projekte/libpruio/doc/html/index.html.

[12] Modular toolkit for data processing (mdp). http://mdp-toolkit.sourceforge.net/.

[13] OK Google Voice Search and Actions. https://support.google.com/websearch/answer/2940021?hl=en.

[14] Pandas library. http://pandas.pydata.org/pandas-docs/stable/generated/pandas.Series.interpolate.html.

[15] Phillips Hue. http://www2.meethue.com/en-us/.

[16] Say hello to Cortana. http://www.microsoft.com/en-us/mobile/experiences/cortana/.

[17] SparkFun Electret Microphone Breakout. http://www.ti.com/product/OPA344.

[18] Tascam. http://www.kraftmusic.com/tascam-trackpack-4x4-complete-recording-studio-bonus-pak.html.

[19] TP Link Smart Plug. http://www.tp-link.com/en/products/list-5258.html.

[20] B. A. Barsky, R. H. Bartels, and J. C. Beatty. *An Introduction to Splines for Use in Computer Graphics and Geometric Modeling*. Los Altos, Calif.: M. Kaufmann Publishers, 1987.

[21] P. Bofill and M. Zibulevsky. Underdetermined blind source separation using sparse representations. *Signal processing*, 81(11):2353–2362, 2001.

[22] S. Boll. Suppression of acoustic noise in speech using spectral subtraction. *IEEE Transactions on acoustics, speech, and signal processing*, 27(2):113–120, 1979.

[23] S.-M. Cha and L.-W. Chan. Applying independent component analysis to factor model in finance. In *International Conference on Intelligent Data Engineering and Automated Learning*, pages 538–544. Springer, 2000.

[24] K. Choi, R. Soma, and M. Pedram. Fine-grained dynamic voltage and frequency scaling for precise energy and performance tradeoff based on the ratio of off-chip access to on-chip computation times. *IEEE transactions on computer-aided design of integrated circuits and systems*, 24(1):18–28, 2005.

[25] A. Cichocki, R. Zdunek, A. H. Phan, and S.-i. Amari. *Nonnegative matrix and tensor factorizations: applications to exploratory multi-way data analysis and blind source separation*. John Wiley & Sons, 2009.

[26] P. Comon and C. Jutten. *Handbook of Blind Source Separation: Independent component analysis and applications*. Academic press, 2010.

[27] S. Cruces-Alvarez, A. Cichocki, and L. Castedo-Ribas. An iterative inversion approach to blind source separation. *IEEE Transactions on Neural Networks*, 11(6):1423–1437, 2000.

[28] X. Cui, V. Goel, and B. Kingsbury. Data augmentation for deep neural network acoustic modeling. *IEEE/ACM Transactions on Audio, Speech and Language Processing (TASLP)*, 23(9):1469–1477, 2015.

[29] K. I. Diamantaras and T. Papadimitriou. Mimo blind deconvolution using subspace-based filter deflation. In *Acoustics, Speech, and Signal Processing, 2004. Proceedings.(ICASSP'04). IEEE International Conference on*, volume 4, pages iv–433. IEEE, 2004.

[30] R. F. Dickerson, E. Hoque, P. Asare, S. Nirjon, and J. A. Stankovic. Resonate: reverberation environment simulation for improved classification of speech models. In *Proceedings of the 13th international symposium on Information processing in sensor networks*, pages 107–118. IEEE Press, 2014.

[31] W. R. Dieter, S. Datta, and W. K. Kai. Power reduction by varying sampling rate. In *Proceedings of the 2005 international symposium on Low power electronics and design*, pages 227–232. ACM, 2005.

[32] R. Gribonval and S. Lesage. A survey of sparse component analysis for blind source separation: principles, perspectives, and new challenges. In *ESANN'06 proceedings-14th European Symposium on Artificial Neural Networks*, pages 323–330. d-side publi., 2006.

[33] S. Haykin and Z. Chen. The cocktail party problem. *Neural computation*, 17(9):1875–1902, 2005.

[34] J. Herault and C. Jutten. Space or time adaptive signal processing by neural network models. In *Neural networks for computing*, volume 151, pages 206–211. AIP Publishing, 1986.

[35] S. Herbert and D. Marculescu. Analysis of dynamic voltage/frequency scaling in chip-multiprocessors. In *Low Power Electronics and Design (ISLPED), 2007 ACM/IEEE International Symposium on*, pages 38–43. IEEE, 2007.

[36] T. K. Ho. Random decision forests. In *Document Analysis and Recognition, 1995., Proceedings of the Third International Conference on*, volume 1, pages 278–282. IEEE, 1995.

[37] T. K. Ho, J. J. Hull, and S. N. Srihari. Decision combination in multiple classifier systems. *IEEE transactions on pattern analysis and machine intelligence*, 16(1):66–75, 1994.

[38] A. Hyvarinen. Fast and robust fixed-point algorithms for independent component analysis. *IEEE transactions on Neural Networks*, 10(3):626–634, 1999.

[39] A. Hyvärinen and E. Oja. A fast fixed-point algorithm for independent component analysis. *Neural computation*, 9(7):1483–1492, 1997.

[40] A. Hyvärinen and E. Oja. Independent component analysis: algorithms and applications. *Neural networks*, 13(4):411–430, 2000.

[41] N. Jaitly and G. E. Hinton. Vocal tract length perturbation (vtlp) improves speech recognition. In

Proc. ICML Workshop on Deep Learning for Audio, Speech and Language, 2013.

[42] T.-P. Jung, S. Makeig, C. Humphries, T.-W. Lee, M. J. Mckeown, V. Iragui, and T. J. Sejnowski. Removing electroencephalographic artifacts by blind source separation. *Psychophysiology*, 37(02):163–178, 2000.

[43] N. Kanda, R. Takeda, and Y. Obuchi. Elastic spectral distortion for low resource speech recognition with deep neural networks. In *Automatic Speech Recognition and Understanding (ASRU), 2013 IEEE Workshop on*, pages 309–314. IEEE, 2013.

[44] T. Ko, V. Peddinti, D. Povey, and S. Khudanpur. Audio augmentation for speech recognition. In *Proceedings of INTERSPEECH*, 2015.

[45] L. I. Kuncheva. A theoretical study on six classifier fusion strategies. *IEEE Transactions on pattern analysis and machine intelligence*, 24(2):281–286, 2002.

[46] L. I. Kuncheva, J. C. Bezdek, and R. P. Duin. Decision templates for multiple classifier fusion: an experimental comparison. *Pattern recognition*, 34(2):299–314, 2001.

[47] E. Le Sueur and G. Heiser. Dynamic voltage and frequency scaling: The laws of diminishing returns. In *Proceedings of the 2010 international conference on Power aware computing and systems*, pages 1–8, 2010.

[48] S. Li, C. Li, K.-T. Lo, and G. Chen. Cryptanalyzing an encryption scheme based on blind source separation. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 55(4):1055–1063, 2008.

[49] Q.-H. Lin, F.-L. Yin, T.-M. Mei, and H. Liang. A blind source separation based method for speech encryption. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 53(6):1320–1328, 2006.

[50] P. C. Loizou. *Speech enhancement: theory and practice*. CRC press, 2013.

[51] H. Lu, A. B. Brush, B. Priyantha, A. K. Karlson, and J. Liu. Speakersense: energy efficient unobtrusive speaker identification on mobile phones. In *International Conference on Pervasive Computing*, pages 188–205. Springer, 2011.

[52] R. Martin. Spectral subtraction based on minimum statistics. *power*, 6:8, 1994.

[53] S. P. Meyn and R. L. Tweedie. *Markov chains and stochastic stability*. Springer Science & Business Media, 2012.

[54] E. Miluzzo, C. T. Cornelius, A. Ramaswamy, T. Choudhury, Z. Liu, and A. T. Campbell. Darwin phones: the evolution of sensing and inference on mobile phones. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*, pages 5–20. ACM, 2010.

[55] S. K. Mitra and J. F. Kaiser. *Handbook for digital signal processing*. John Wiley & Sons, Inc., 1993.

[56] N. Morales, L. Gu, and Y. Gao. Adding noise to improve noise robustness in speech recognition. In *INTERSPEECH*, pages 930–933, 2007.

[57] S. Nirjon, R. F. Dickerson, P. Asare, Q. Li, D. Hong, J. A. Stankovic, P. Hu, G. Shen, and X. Jiang. Auditeur: A mobile-cloud service platform for acoustic event detection on smartphones. In *Proceeding of the 11th annual international conference on Mobile systems, applications, and services*, pages 403–416. ACM, 2013.

[58] D. Ruta and B. Gabrys. An overview of classifier fusion methods. *Computing and Information systems*, 7(1):1–10, 2000.

[59] G. Semeraro, G. Magklis, R. Balasubramonian, D. H. Albonesi, S. Dwarkadas, and M. L. Scott. Energy-efficient processor design using multiple clock domains with dynamic voltage and frequency scaling. In *High-Performance Computer Architecture, 2002. Proceedings. Eighth International Symposium on*, pages 29–40. IEEE, 2002.

[60] G. Srivastava, S. Crottaz-Herbette, K. Lau, G. Glover, and V. Menon. Ica-based procedures for removing ballistocardiogram artifacts from eeg data acquired in the mri scanner. *Neuroimage*, 24(1):50–60, 2005.

[61] E. Vincent, R. Gribonval, and C. Févotte. Performance measurement in blind audio source separation. *IEEE transactions on audio, speech, and language processing*, 14(4):1462–1469, 2006.

[62] B. Widrow, J. R. Glover, J. M. McCool, J. Kaunitz, C. S. Williams, R. H. Hearn, J. R. Zeidler, J. E. Dong, and R. C. Goodlin. Adaptive noise cancelling: Principles and applications. *Proceedings of the IEEE*, 63(12):1692–1716, 1975.

[63] B. Widrow and S. D. Stearns. Adaptive signal processing. *Englewood Cliffs, NJ, Prentice-Hall, Inc., 1985, 491 p.*, 1, 1985.

[64] K. Woods, W. P. Kegelmeyer, and K. W. Bowyer. Combination of multiple classifiers using local accuracy estimates. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(4):405–410, 1997.

[65] M. Zibulevsky and B. A. Pearlmutter. Blind source separation by sparse decomposition in a signal dictionary. *Neural computation*, 13(4):863–882, 2001.