Crowd-supervised Training of Spoken Language Systems

by

Ian Carmichael McGraw

B.S., Stanford University, California, USA (2005)
M.S., Massachusetts Institute of Technology, Cambridge, USA (2008)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctorate of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2012

Author . . . .
Department of Electrical Engineering and Computer Science
June 14, 2012

Certified by . . . .
Stephanie Seneff
Senior Research Scientist
Thesis Supervisor

Accepted by . . . .
Randall Davis
Professor of Electrical Engineering
Chairman, Area II Department Committee on Graduate Students

Crowd-supervised Training of Spoken Language Systems

by

Ian Carmichael McGraw

Submitted to the Department of Electrical Engineering and Computer Science
on June 14, 2012, in partial fulfillment of the
requirements for the degree of
Doctorate of Philosophy

Abstract

Spoken language systems are often deployed with static speech recognizers. Only rarely are parameters in the underlying language, lexical, or acoustic models updated on-the-fly. In the few instances where parameters are learned in an online fashion, developers traditionally resort to unsupervised training techniques, which are known to be inferior to their supervised counterparts. These realities make the development of spoken language interfaces a difficult and somewhat ad-hoc engineering task, since models for each new domain must be built from scratch or adapted from a previous domain.

This thesis explores an alternative approach that makes use of *human computation* to provide crowd-supervised training for spoken language systems. We explore human-in-the-loop algorithms that leverage the collective intelligence of crowds of non-expert individuals to provide valuable training data at a very low cost for actively deployed spoken language systems. We also show that in some domains the crowd can be incentivized to provide training data for free, as a byproduct of interacting with the system itself. Through the automation of crowdsourcing tasks, we construct and demonstrate *organic* spoken language systems that grow and improve without the aid of an expert.

Techniques that rely on collecting data remotely from non-expert users, however, are subject to the problem of noise. This noise can sometimes be heard in audio collected from poor microphones or muddled acoustic environments. Alternatively, noise can take the form of corrupt data from a worker trying to game the system – for example, a paid worker tasked with transcribing audio may leave transcripts blank in hopes of receiving a speedy payment. We develop strategies to mitigate the effects of noise in crowd-collected data and analyze their efficacy.

This research spans a number of different application domains of widely-deployed spoken language interfaces, but maintains the common thread of improving the speech recognizer's underlying models with crowd-supervised training algorithms. We experiment with three central components of a speech recognizer: the language model, the lexicon, and the acoustic model. For each component, we demonstrate the utility of a crowd-supervised training framework. For the language model and lexicon, we explicitly show that this framework can be used hands-free, in two organic spoken language systems.

Thesis Supervisor: Stephanie Seneff
Title: Senior Research Scientist

Acknowledgments

This thesis simply would not have been possible without my advisor, Stephanie Seneff. Over the years her guidance and friendship have been invaluable. Stephanie's enormous creativity coupled with her incredible enthusiasm for new ideas, gave me the confidence to push this research in new directions, no matter how "out there" they might have seemed at first. She has seen me through the inevitable ups and downs of getting a doctorate, and for that I am extremely grateful.

From my early years in the Spoken Language Systems (SLS) group, Jim Glass has also been a mentor to me. Under his guidance, I had the amazing opportunity to TA MIT's automatic speech recognition course (6.345). I may have learned more in that enlightening semester of leading sections and holding office hours, than through all my other MIT courses combined. Since then, Jim has collaborated and commented on many of my papers.

Victor Zue has been another extremely positive influence on my academic trajectory over the past few years. Even with his absurdly busy schedule, he has always found time to check on my progress. This keen awareness that my progress needed checking kept me from slipping on more than a few important deadlines, including the submission of this very document. Victor, Stephanie and Rob Miller form my thesis committee, whose expertise I have relied on throughout my research, and for which, I am extremely grateful.

My years here at MIT have been the best of my life, and this is largely due to the fact that the SLS is comprised of some truly great people. Working with Scott Cyphers on everything from the latest *X-Browser* to the little-known task of transmogrifying audio has been rewarding. Marcia Davidson has made my life easier in countless ways, from managing an ever-growing list of Amazon Mechanical Turk accounts to keeping me on track with reimbursements. In fact, I should really thank Marcia twice, both because she deserves it and because she was accidentally omitted from my last theses acknowledgements.

For me, these people form the heart of the SLS group. They were here when I came, and they are here as I am wrapping up. There are many others who I have met along the way, however, that have also had a profound impact on my career as a graduate student. Lee Hetherington's knowledge about the guts of our speech recognizer proved indispensable on many occasions. Alex Gruenstein has been a role model for me both as a researcher and as an engineer. Without his work on the WAMI Toolkit, much of the contents of this thesis would not have been possible. Working with Ibrahim Badr on the pronunciation mixture model was also refreshing, especially since the small, self-contained problem turned out to have a larger impact when combined with crowd-sourcing techniques later developed.

Interacting with other researchers that grew up with me in SLS, including Mitch Peabody, Jingjing Liu, Yushi Xu, and Hung-An Chang was part of what made the experience both fun and educational. Thanks especially to Stephen Shum, Dave Harwath and Timo Mertens for organizing the best reading group I have ever had the pleasure of attending. Thanks to Tuka Al Hanai for putting up with the shenanigans that went on in our office. There are a host of others that deserve credit for getting me to this point: Will Walker, Carrie Cai, Najim Dehak, T.J. Hazen and Jackie Lee. This research, was funded in large part by Quanta Computers Inc., who afforded me the opportunity to travel to Taiwan and meet the amazing people who work there on several occasions.

Finally, I would like to thank my family for their support – especially my wife, Ming.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Spoken language systems are increasingly prevalent in our everyday lives. Voice interfaces are now found, not just on personal computers, but in our smartphones and vehicles as well. While initially viewed as a requirement for accessibility, speech is now becoming the preferred interaction modality for many tasks. Siri delights Apple iPhone owners in multiple languages with the ability to send texts or make a quick web search [67]. Numerous high-end automobiles incorporate voice-navigation features directly into the dashboard [32, 28]. It is difficult to watch a few hours of television without seeing at least one speech interface, either within a show or explicitly advertised during the commercials. Some well known speech companies are currently even building voice interfaces for the television itself [50].

While spoken language systems have now truly escaped the laboratory, these systems, together with their counterparts in academia, often have one very large flaw. Despite being deployed to a dynamic environment, the underlying models in a spoken language system are often given fixed parameter values. How, then, can these interfaces adapt to new types of input? For example, an Internet voice-search application must keep up-to-date with the trending news topics, which might include the pronunciations *and* mispronunciations of proper names. A recognizer trained with audio collected from desktop computers will find itself in an entirely different acoustic environment when trained on mobile devices.

Typically, these systems simply wait for help from an expert or rely on unsupervised adaptation methods. The first approach relies on the expert's determination that the current models are unsatisfactory, and requires a manual update, perhaps with a transcribed corpus of training data. The expert may even manually add new pronunciations into the recognizer. While the unsupervised adaptation approach offers an alternative that can be performed on-the-fly, such techniques are almost always improved with labeled training data. This thesis explores an alternative approach that employs resources from crowds of non-experts to semi-automatically update the various components of an automatic speech recognizer.

1.1 Motivation

For many years, the speech research community has dreamt of organic spoken language systems that grow and improve without expert-aid [151]. While these systems are closer than ever to becoming a reality, it is still common for users to simply learn to live with a system's errors. The goal, for many researchers, is to move towards *living* systems that learn *from* their own errors. More generally, systems should be able to adapt to their environment without expert help.

There are many components in speech systems that might automatically adapt according to the environment in which it's deployed. This thesis focuses on the automatic speech recognizer itself. Speech recognizers can be viewed hierarchically, each layer placing additional constraints over a probabilistic search space [47]. When a speech interface targets a new domain, new constraints must be learned by gathering in-domain data and retraining the recognizer's underlying stochastic models. We consider three such models in this thesis. The component closest to the speech signal itself is the *acoustic model*, which scores competing recognition hypotheses on the level of individual sounds. The *lexicon*, which maps sequences of these sounds to words in a pronunciation dictionary, can also be viewed probabilistically. Further constraints are placed on the sequence of words explored during recognition by the *language model*. Each of these components of a recognizer can be learned and adapted given enough training data.

Collecting sufficient training data has never been a trivial task. Successful collection efforts in academia have sometimes required cooperation across multiple institutions and have largely been telephone-based [64]. Audio transmitted over telephone networks, however, is usually sampled at 8kHz, a rate which effectively cuts out frequencies important to certain speech sounds, which can degrade recognition performance [123]. Alternative technologies, either through the phone or over the web, have been slow to develop. This motivates, as part of this thesis, the development of tools that ease the burden of audio collection.

Tools for audio collection are worthless, however, without subjects to provide the spoken language and human interaction necessary to improve a voice interface. The advent of crowdsourcing techniques for speech-related tasks provides a new way to overcome the problem of data-scarcity [22, 114]. Micropayment workforce platforms, such as Amazon Mechanical Turk (mTurk), provide access to non-expert individuals who can provide new data or annotate existing data cheaply. Furthermore, when used programmatically, these services can effectively incorporate human intelligence into complex algorithms, a paradigm referred to as *human computation.* This ushers in a fundamentally new way to train a spoken language system. Part of this thesis will show that, if one can elicit the right form of data from the crowd, it is possible to incorporate this collective intelligence into an entirely hands-free, *crowd-supervised* training process.

The models underlying a spoken language interface are trained using large amounts of in-domain data, typically in the form of the audio signal and a transcript of the words that were spoken. We show, in this work, the feasibility of crowdsourcing the retraining and adaptation of the three previously mentioned stochastic models of the modern speech recognizer. For language modeling, we operate on three different types of models, which can be updated dynamically during a session with one of our spoken language interfaces. In the case of the lexicon, we introduce the pronunciation mixture model as a framework for learning new weighted lexical entries using spoken examples from the crowd. Finally, our acoustic model experiments demonstrate that micropayment platforms are not the only way of procuring transcribed, in-domain speech data. For each task, we address the problem of noisy or corrupt crowdsourced data, again using fully automatic procedures.

This goal of this thesis is therefore to connect two previously independent areas of research: human computation and spoken language systems. In the past, system developers wishing to improve the models of a speech recognizer were given one of two choices – they could either manually transcribe data in an offline procedure for supervised training or they relied on unsupervised training, known to be less effective. In this thesis, we show that there is a middle-ground of *crowd-supervised* training techniques that can provide more robust gains, without requiring the aid of an expert. Moreover, using human computation, we demonstrate that we can wrap crowd-supervised techniques into organic systems that improve on-the-fly. The next section outlines the contributions of this thesis chapter by chapter.

1.2 Thesis Outline

Chapter 2: Background

This chapter presents two areas of literature and one section of background knowledge related to the work in this thesis. First, we familiarize the reader with the latest research of crowdsourcing, and in particular the field of human computation. To give the reader a concrete example of crowdsourcing in practice, we perform a literature review of the field with the help of workers from Amazon Mechanical Turk, arguably the most popular platform for micropayment-tasks. The second area of related research that we describe is that of spoken language systems. We make particular note of the enormous effort involved in past speech data collection undertakings. Third, we arm the reader with an understanding of the probabilistic models within an automatic speech recognizer. Although the lexicon is not often modeled stochastically, we provide its mathematical underpinnings, together with those of the language model, and acoustic models. Finally, we conclude this chapter with a short summary.

Chapter 3: Collecting Speech from the Crowd

This chapter demonstrates the utility of Amazon Mechanical Turk as a platform for speech data collection. We perform two sets of experiments. The first is designed to elicit read speech from workers via prompts. This experiment was open to workers all over the world and required only that they have a web browser and a mTurk account. The entire experiment ran for less than four days, but collected more than 100 hours of audio. Thus, we were often collecting audio at rates faster than real-time. Some of the audio, however, contained artifacts such as background noise or clipping – other times, speakers were simply unintelligible. We therefore explore ways of filtering this large corpus automatically for subsets of clean speech. The second experiment conducted was for a multimodal spoken dialogue system, which we deployed directly inside the mTurk interface. This time, however, workers were limited to the United States. We collect over 1,000 dialogue sessions and evaluate the system using crowd-transcribed utterances. The work in this chapter was previously published in [106].

Chapter 4: Crowd-supervised Language Modeling

This chapter describes a prototype organic system in which the language models are updated in crowd-supervised fashion. We present a prototype system for a photo annotation and search domain. This experiment represents our first example of human computation, in which the role of the human workers is entirely automated by the system itself. As utterances are collected, they are sent back out to mTurk for transcription. When audio transcription is complete, the transcripts are recompiled into a language model that is then transmitted to the speech recognizer for an on-the-fly update. We show that search query accuracy improves over the lifespan of this *living* system. Over a two day span, in two separate experiments, we demonstrate the feasibility of growing an organic language model. The work in this chapter was previously published in [104].

Chapter 5: Crowd-supervised Lexicon Learning

This chapter treats the lexicon as another probabilistic model in the speech recognizer and makes use of this stochasticity to perform crowd-supervised pronunciation learning. We begin by describing the pronunciation mixture model (PMM), a maximum likelihood estimation approach to learning new pronunciations given a set of spoken examples. We present a suite of experiments on the *PhoneBook* corpus of isolated-word speech, which show that the PMM is robust to crowdsourced data. We show that, not only does this model outperform the state-of-the-art letter-to-sound approaches, it produces pronunciations that are on par with those of experts. In the remainder of the chapter, we use both the letter-to-sound (L2S) and the PMM in an organic spoken language system for the cinema voice-search domain. We automate the collection of both transcripts and spoken examples to show that a crowd-supervised framework can learn new words on-the-fly to improve speech recognition accuracy. The pronunciation mixture model was proposed and analyzed in [5, 6], where Ibrahim Badr performed the experimental analysis on the original *PhoneBook* dataset. The *PhoneBook* experiments were then supplemented with a crowdsourced corpus, upon which additional experiments were run. The second set of experiments in this chapter used the *Movie Browser* dialogue system, described in detail in [93]. The crowd-supervised lexicon learning experiments described in this chapter have been submitted for publication [103].

Chapter 6: Crowd-supervised Acoustic Modeling

This chapter moves beyond the mTurk platform and collects self-transcribed audio from educational games, which we use to adapt the acoustic models. We deployed two speech-enabled games to *Quizlet.com*, a popular flashcard website for studying terms and definitions. Over a 22 day period, we collect almost 50 hours of speech. We also show that around one-third of these data can be transcribed simply by examining the game context in which it was collected. We use these utterances to perform self-supervised acoustic model adaptation and show that the system can improve without the help of an external crowd. This work demonstrates that, in some cases, it may be possible to improve the speech recognizer by leveraging the crowd available to any spoken language system: its own user-base. The work on *Voice Race* was originally published in [105] and the work with *Voice Scatter* was published in [53]. Alex Gruenstein assisted with these experiments and, in particular, managed the acoustic model adaptation.

Chapter 7 Conclusions

This chapter summarizes the work presented in this thesis. We review the experiments and delineate the contributions of this thesis. We then point to ways of extending this research in various directions. For each component of the speech recognizer, we describe further experiments that might be performed toward making spoken language systems truly adaptive.

Chapter 2

Background

This chapter covers the related literature and background concepts relevant to this thesis. As previously noted, this thesis is positioned at the intersection of two active areas of research. First and foremost, our goal is to advance spoken language systems research by making the construction and continued improvement of such systems easier and more cost-effective. We accomplish this task by utilizing the collective intelligence of a distributed group of non-expert individuals. We employ crowdsourcing techniques to automate data collection efforts that were previously handled manually in the laboratory. A number of experiments presented here take crowdsourcing a step further by removing the expert not only from the data collection phase, but from the speech recognizer's retraining phase as well. The retraining process now becomes computer-driven rather than human-driven, a distinction that earns it the classification of *human computation*.

There are three areas of related work and background material covered in this chapter. The first is crowdsourcing. We codify the various types of crowd-related computer science research areas and then narrow in on the subfield of human computation, most relevant to our work. The second section of this chapter gives a short history of spoken language systems, and describes the data collection efforts required to train them. Finally, we discuss the probabilistic formulation of an automatic speech recognizer. We concentrate on three stochastic models of the recognizer, which define its search space given an audio signal. For each model, we are careful to describe the types of training data necessary to learn its parameters.

2.1 Collective Intelligence

Researchers in the field of human-computer interaction have recently found fascination in the ability of large, often anonymous groups of people to perform distributed tasks to-

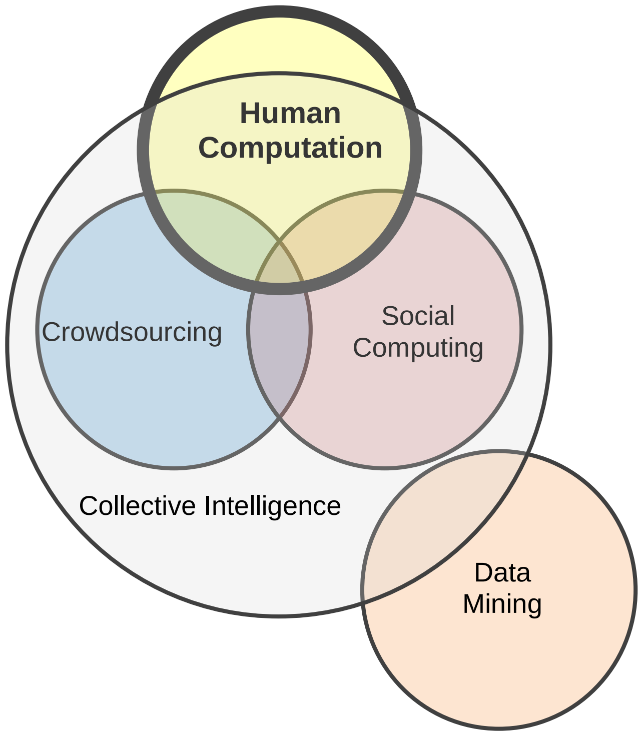


Figure 2-1: A taxonomy of collective intelligence paradigms, from [118]

wards solving a larger problem. In this section, we provide an overview of this field, under the umbrella title of *collective intelligence*, while focusing in particular on an area called crowdsourcing. Crowdsourcing can take advantage of the ever-increasing connectivity of individuals to perform collaborative efforts. At times, individuals in the group are not even aware of the goal at hand or that any collaboration is happening at all. For the most part, participation in a particular collaborative task is not compulsory, making motivations for participation another area of study. Finally, with large crowds it can be difficult to manually oversee the results. For this reason, quality control becomes another concern.

2.1.1 Crowdsourcing vs. Human Computation

There have been a number of attempts to delineate the various subfields of collective intelligence [46, 118]. In this thesis we largely follow the taxonomy of [118], which breaks the field primarily into the overlapping areas of human computation, crowd-sourcing, and social computing, as shown in Figure 2-1. The latter, having largely to do with the use of technology to ease the natural communication between humans, is of little relevance to this work. Of greater importance for this work are the fields of human-computation and crowd-sourcing, and especially their overlap. In this work, we only employ human computation

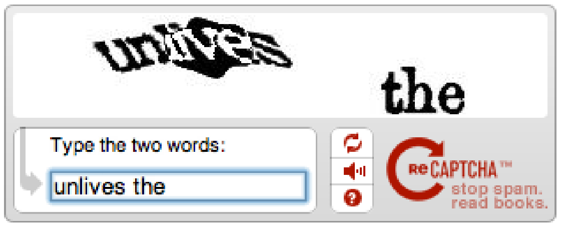


Figure 2-2: *ReCAPTCHA* digitizes books and secures websites simultaneously. A user attempting to access a *ReCAPTCHA*-secured site is presented two digitally scanned words which they must transcribe. One of the words, the “control” word, is known to the algorithm and is used to verify that the user is human. The transcription of the other is compared across other users given the same word. Their agreement is used to transcribe this previously unknown word.

in a crowdsourcing setting, so, for our purposes human computation is a proper subset of crowdsourcing.

Where human computation differs from other types of crowdsourcing, the distinction is a subtle one. Both tend to refer to a paradigm that combines computational processes with human input to solve a problem. For the purposes of this work, we presume that human computation places the additional constraint that the overall process must be directed automatically rather than human-guided. On the other hand, crowdsourcing merely requires that the human component must be carried out by more than one human, often a group of anonymous individuals. The entirety of this thesis will revolve around crowdsourcing techniques for spoken language systems. A subset of our experiments are directed entirely by computational processes, and are thus given the label *human computation*.

One compelling example of crowdsourcing is Wikipedia, which accepts contributions to its online encyclopedia from anyone with a web browser and a little spare time. Ten years since its inception, the English version of the site is approaching four million articles, the contents of which would fill over 1500 physical encyclopedia volumes if printed to paper [132]. While undoubtedly a success, Wikipedia also exemplifies the quality control concerns that are common to many domains in the broader field of collective intelligence [80]. In [118], it is argued that Wikipedia is not an example of human computation, however, because the overall goal of anthologizing information, right down to the choice of which articles should be included, is directed by humans.

A project which illustrates human computation is ReCAPTCHA [136]. A CAPTCHA is a Completely Automated Public Turing test to tell Computers and Humans Apart. They are used by websites to prevent malicious computer programs from accessing material or performing operations meant only for humans. Often CAPTCHAs appear as distorted letters that the user of a website must recognize, type, and submit to prove he or she is human. Figure 2-2 shows an example from the ReCAPTCHA project. Human computation comes into play by formulating CAPTCHAs for which the aggregate human effort can be automatically combined to digitize scanned books. This can be done by using a *control* word and an *unknown* word in each CAPTCHA. The control word is used for security purposes, and checked before the human can proceed. The results for the unknown word are then compared with results from other humans who were given the same unknown word. Matching the input in this way uncovers the orthography of a word that may have been difficult to ascertain using purely computational methods such as optical character recognition. Recent statistics are difficult to come by now that Google has acquired the technology, but estimates computed after its first year of use put the number of words transcribed near half a billion.

Clearly there is a significant difference in flavor between the type of crowdsourcing used to generate Wikipedia and the type used to digitize books via reCAPTCHAs. In the case of reCAPTCHA, book digitization is a byproduct of a human's actions that were geared towards a somewhat unrelated goal. With Wikepedia articles, the end goal of the user *is* to create or improve upon an article. More generally, the compilation of the encyclopedia is human-guided in the case of Wikipedia, whereas the process of book digitization can be guided entirely automatically, using humans as one modular component in a larger digitization algorithm. It is this characteristic, the automation of the higher-level process, that puts reCAPTCHA squarely in the category of human computation.

To clarify this point, we look at another famous example in the field of collective intelligence: the *ESP Game* [137]. The *ESP Game* is a two-player online Game With a Purpose (GWAP). In addition to being fun, GWAPs produce or label data. In the ESP Game, each player is shown the same picture and they each independently label this picture with words based on its contents. When the labels from the two players match, they each earn points. The goal of the game is simply to earn points by matching as many labels as possible within a specified period of time. A byproduct of each game is that the image is tagged with a set of labels that represent its contents. The game labeled almost 300,000 images in the first five months of its deployment. As is, this process of labeling a set of images is a form of human computation, since it can be driven entirely algorithmically. Another layer of confusion of the terms can be added, however, by taking the output of this process and using it as training data for machine learning algorithms.

Since retraining spoken language systems using crowdsourcing techniques is of central concern to this thesis, we explicitly define our terms with respect to this additional layer of computation. In particular, it becomes important to establish whether the retraining phase itself is driven by the human or the machine. In this work, we retain the term human computation when the training data are collected, collated, and the machine learning algorithms are retrained entirely automatically. We use the generic term crowdsourcing when a human, usually an expert, steps into the loop at the retraining phase. Ultimately, removing the expert, and closing that loop is the goal of this work.

2.1.2 Micropayment Workforces

The crowdsourcing examples we have described thus far have had domain-specific incentives for completing the work at hand. A more general, but inherently more costly approach is to pay distributed workers to complete tasks. Micropayment workforce platforms such as CrowdFlower and Amazon Mechanical Turk (mTurk) offer to act as intermediaries between requesters and these anonymous crowds of workers. In this section, we will describe the characteristics of these markets and the types of tasks that can be performed on them.

Amazon Mechanical Turk is arguably the most mature crowdsourcing platform for arbitrary micro-tasks. Each unit of work on mTurk is referred to as a Human Intelligence Task (HIT). HITs can be posted to the crowd through Amazon's simple web interface. To provide an illustration of its capabilities, the literature review given in this section has been compiled using mTurk. The HIT for this literature review requested that workers use any means at their disposal to find academic articles about crowdsourcing. It was suggested that they make use of Google Scholar as well as search through references of papers already found. It should be noted that these workers do not have access to the same resources that an expert in the field might have, such as the ability to scale the pay-walls that frequently surround academic papers or ask colleagues for recommendations. For the gathering phase of a literature review, however, they merely need to be able to identify topics, which can be done with little expertise.

We deployed two tasks to mTurk on different days with different settings. The first cost 5¢ per paper and was open to anyone in more than 100 countries worldwide. The second was restricted to the United States and paid 20¢ per paper found. We collected 125 papers for each task. Before we ran the mTurk tasks, we performed our own independent literature review and had written the basic narrative of this section. The papers ultimately included were chosen to supplement this narrative. A comparison of the papers we found and those collected in our experiment can be seen in the remainder of this section. We have placed an indicator on each citation denoting whether it was found by the expensive task [x]^e, the cheaper task [x]^c of the independent literature review [x]^i.

Although the nomenclature is far from settled in these relatively young fields, for the purposes of this thesis, we will consider this literature review task to be an example of crowdsourcing, but not necessarily one of human computation. As previously described we make this distinction when the process is overseen by a human rather than directed by the machine. Moreover, this task has a few properties that are not typical of an Amazon Mechanical Turk task. It is often assumed that mTurk workers must work independently – or if not independently, at least not simultaneously. In [77]^{c}, however, it is shown that allowing workers to explicitly collaborate on a poem translation task produces fruitful results. For our literature review task, we use a similar approach, requiring workers to add each paper to a growing list using an online collaborative text editor[1]. A worker must check for duplicates before adding a paper to the list, copying it to the HIT, and submitting.

A chat area beside the document allows workers to communicate with each other. Even the requester can log in and shepherd workers to perform the task a particular way, a strategy that has been shown to produce better work and provide a useful alternative to the output agreement or iterative paradigms typically employed to control quality in mTurk tasks [39]^{c}. We employed this technique only once in our first HIT, to tell workers that abstracts of papers behind a pay-wall were acceptable. The subsequent HIT included this explicitly in the instructions.

Although we opened our cheaper task up to many different countries around the world, most of our workers came from either the US and India. This is consistent with a survey of worker demographics which found that the mTurk crowd was dominated by workers from the US, but that the trend is shifting towards workers from India [122]^{e,i}. It was also noted that 10% of Indian workers relied on Amazon Mechanical Turk to make ends meet, whereas only 5% of American workers did so. This, and other reasons, have led some researchers to highlight the ethical questions surrounding the use of mTurk workers as independent contractors, a classification that allows the market to circumvent minimum wage laws (see [131]^{c,e}, [124]^{c} and [101]^{c,i}). Indeed, the motivations of workers, who often work for just a few dollars an hour, was investigated again in [75]^{e,i}. Interestingly, further work has been done in this area to ascertain whether a social desirability bias exists in the workers' reporting of their own motivations [3]^{i}. It was found, for instance, that Indian workers under-reported "fun" as a motivator for task participation, while US workers over-reported the role of money.

Motivations aside, Amazon Mechanical Turk has been used for incredibly interesting and diverse purposes. In the medical domain, labs have experimented with using mTurk to identify cancer cells (see [25]^{c,i} and [108]^{c}). VizWiz is a smartphone app that allows blind users to upload a photo and a spoken query to a crowd of mTurk workers who

[1]http://collabedit.com/

then assist by identifying objects or writing in the image [14]^{e,i}. PlateMate has a similar modality, whereby anyone can upload a picture of a plate of food, and, through a series of mTurk tasks, the content of the plate is ascertained with respect to calories, fat, carbohydrates, and protein [109]^{e,i}. Soylent, "a word processor with a crowd inside," crowdsources common word processing tasks such as summarizing and proof-reading [12]^{i}. In fact, work on Soylent was part of the inspiration for performing this literature review via crowdsourcing.

For some mTurk tasks latency is a primary concern. Obviously latency can be high for systems which incorporate humans into the loop, and, while for some domains high latency is merely an inconvenience, in others it is unacceptable. VizWiz, for example, requires a quick response if an image description is going to be useful to the blind individual who submitted it. Research into real-time crowdsourcing is still in its infancy, but techniques do exist to speed up reaction times from the crowd [11]^{c,i}. Queueing theory has even been used to optimize the cost/performance tradeoffs of keeping a crowd constantly at the ready for real-time tasks [13]^{c,i}.

Independent of the question of latency, there has been considerable research regarding control architectures for mTurk tasks. In many domains, it can be advantageous to feed the output of one task in as the input of another. The visual management of crowdsourcing workflows has been explored in CrowdWeaver [79]^{c,e,i}. TurKit allows JavaScript programs to manipulate HITs and cache the human computation during each pass through the code [92]^{i}. CrowdForge [81]^{c,i} follows the map-reduce paradigm to split and combine tasks. Other work has focused on treating the crowd as a database of human knowledge which can be accessed through declarative languages ([43]^{c,e,i}, [98]^{i}, and [113]^{e,i}.) Although the preceding examples are each quite different, they all ease the scripting of human computation tasks through various forms of crowd-control. There has even been work that takes this paradigm a step further and crowdsources the creation of the workflow itself [83]^{e}.

Quality is another dimension important to requesters. Three main approaches to automatically controlling for quality are described here, and may be implemented using the tools described above. The first is the use of a small amount of gold standard data to identify and weed out workers who are gaming the system. This approach has been successfully used in a number of early cases, e.g. [78]^{c,e,i} and [133]^{e,i}. Since obtaining expert-annotated, gold-standard data is expensive, other researchers have explored the utility of generating known answers programmatically in some domains [111]^{e}. The second approach is to deploy multiple assignments for a HIT in parallel and analyze worker agreement in the results. Much of the work in this area goes beyond simple matching and attempts to automatically identify high quality workers through agreement statistics (see [144]^{e} and [68]^{i}.) The final quality control mechanism is an iterative approach. Using this method workers incrementally improve on the output of previous workers [34]^{i}.

A number of studies have examined the effects that positive and negative monetary incentives have on quality (see [102]^{c,e,i} and [57]^{c,e,i}.) The general consensus has been that increasing the payment for a task will increase the speed at which it is done, but not necessarily the quality of the output. The same might be said of our literature review HITs. The more expensive task took four hours to complete and just eight different people contributed. The cheaper task was performed by 20 different workers over the course of three days. Skimming each of the 125 papers in each review, we determined that the cheaper task contained 75 papers that could have reasonably been included while the more expensive task contained 87. 18 papers were recovered by both mTurk tasks. Both tasks recovered interesting papers that were not found in our independent review, suggesting that mTurk can be a useful way to expand a literature search.

2.1.3 Crowd-sourcing for Speech and Language

Crowdsourcing is particularly well suited to the tasks of generating and processing natural language. Again, GWAPs are one means of procuring such data. In [138], von Ahn et al. describe a game called Verbosity which generates common-sense facts. Similar to the ESP Game, Verbosity connects two players, who must collaborate on a word game. A side-effect of the game is that data is created that is useful to the natural language understanding community. Where this type of data generation is difficult, researchers have again resorted to mTurk and other micropayment workforces. Here we review some of the recent work done in this area, reserving the topic of crowdsourcing for speech *collection* for the subsequent section.

Initial interest in the use of Amazon Mechanical Turk in the NLP community began with the work of Snow et al. [133]. They perform experiments on five natural language processing tasks: affect recognition, word similarity, recognizing textual entailment, event temporal ordering, and word sense disambiguation. They also show that quality control can be handled using worker agreement and using a few gold-standard labels. For categorical data they show that the output can be greatly improved with a bias correction algorithm based on the work of Dawid and Skene [36].

A number of review papers nicely summarize many of the subsequent achievements in the field [22, 114]. A few examples of mTurk's use in NLP are for statistical machine translation [21], named entity recognition [89], and evaluating topic modeling [26]. More recent research is beginning to combine mTurk tasks with active learning algorithms, which intelligently and automatically determine how to collect new data on-the-fly [88, 2]. In [2], an end-to-end human-in-the-loop machine translation system is built, which combines non-expert workers with an automatic machine translation system.

Speech transcription is another task ripe for crowdsourcing. Indeed, some work on web-based transcription methods dates back as far as 1997 [40], but required the transcriber to install a special audio player for the task. Transcription on mTurk began in industry with CastingWords, a service that leverages the crowd to transcribe large amounts of audio. Speech tasks took a little longer to arrive on the academic crowdsourcing scene, perhaps due to the additional technological considerations of manipulating audio on the web. Our work, detailed in Chapter 6, initially looked at labeling and transcribing short utterances. Given their brevity, directly comparing transcriptions from multiple workers was an easy form of quality control. Marge et al. explored the use of ROVER to combine longer outputs into a single transcript [99]. Novotney et al. then showed that mTurk can be used to transcribe vast amounts of training data [110].

Beyond transcription, a number of speech-related tasks have been deployed to mTurk. In [100], Marge et al. use non-expert workers to annotate speech from meetings for the purposes of extractive summarization. Evaluation of speech synthesis has also been analyzed using mTurk [146, 18]. Others have taken a similar approach to judging whether a real voice comes from a non-native speaker or not [84]. These examples all require the worker to play audio in the browser. Another interesting set of tasks can be performed when we reverse the paradigm and begin collecting audio through the browser. We describe these in the next section.

2.2 Crowdsourcing Speech Collection

If one defines the term broadly enough, the speech research community has been *crowd-sourcing* the collection of speech for many years and in many different contexts. This section describes speech collection efforts for a diverse set of applications. Explicit corpus collections have been performed for a myriad of speech-related tasks: speaker verification, language identification, large-vocabulary speech recognition, etc. In other cases, such as with widely deployed spoken dialog systems, the collection of audio is simply a byproduct of using the system.

We describe what we believe to be a representative sample of this work to give the reader a feel for the collection techniques typically used for speech. For each collection effort, we also describe the technology used to make data collection possible and give a broad overview of the protocol used during collection. Speech collection can either occur on-site or remotely. The remote methods of data collection applicable to crowd-sourcing include collection over a standard telephone, through the web, or via smartphones.

2.2.1 Speech Corpora

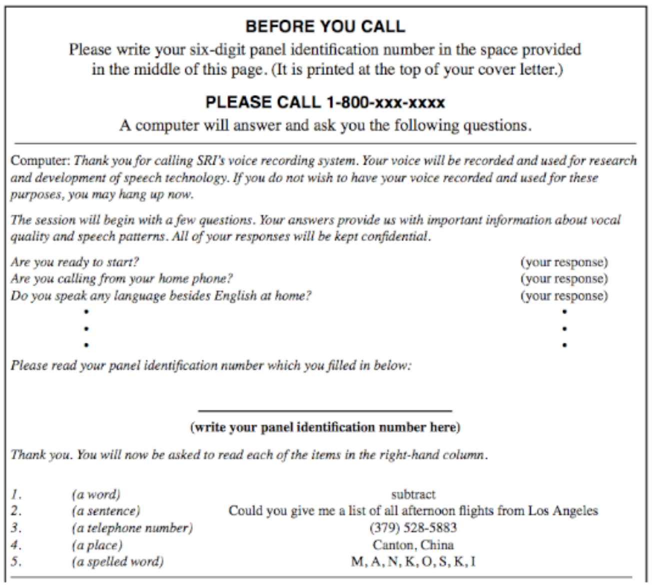
We begin with speech corpus collection, which is a task of central importance in speech research. These corpora provide a standard for comparison with related research in addition to their utility for training various speech-related algorithms. Their very collection, however, is a feat in and of itself. A catalog containing many influential speech corpora is available through the Linguistic Data Consortium (LDC).

For certain purposes it is desirable to collect audio in the lab using multiple microphones under ideal conditions. The *TIMIT* corpus of read speech sponsored by DARPA was one such effort [42]. A collaboration between SRI, Texas Instruments (TI), and MIT, *TIMIT* was designed to contain phonetically varied read speech. Speech was collected from 630 English speakers using two microphones, one close-talking and the other a free-field microphone. Recording was carried out at Texas Instruments and, after some post-processing, 16kHz audio was shipped to MIT for phonetic transcription [152]. To this day, *TIMIT* remains the most popular corpus carried by the LDC.

In subsequent decades, the general trend in speech data collection efforts has been towards collecting data remotely, rather than requiring the speakers to be on site. This method comes with advantages and disadvantages. First, it can be more difficult, for instance, to ensure channel uniformity and a clean acoustic environment across all of the speakers. Second, remote data collection brings new technological challenges, since the recording software must be set up to handle simultaneous speakers. With such an infrastructure in place, however, it is easier for the collectors to connect with a larger number of participants and ensure greater diversity thereof.

Texas Instruments launched a wave of automated corpus collection with its Voice Across America project (VAA) [145]. In order to interface with the telephone network, the protocol required customized hardware and software to connect a PC to a T1 line. Typical of telephone speech, the audio is transmitted and recorded with an 8-bit μ-law encoding scheme. Speech from a normal telephone call could then be recorded to disk. The initial corpus collected contained 3700 speakers producing 50,000 utterances by reading prompted sentences given to them in a unique mailed letter. This collection technique was quickly brought to a larger scale.

The original *SWITCHBOARD* corpus followed in the footsteps of VAA with a similar,



BEFORE YOU CALL

Please write your six-digit panel identification number in the space provided in the middle of this page. (It is printed at the top of your cover letter.)

PLEASE CALL 1-800-xxx-xxxx

A computer will answer and ask you the following questions.

Computer: *Thank you for calling SRI's voice recording system. Your voice will be recorded and used for research and development of speech technology. If you do not wish to have your voice recorded and used for these purposes, you may hang up now.*

The session will begin with a few questions. Your answers provide us with important information about vocal quality and speech patterns. All of your responses will be kept confidential.

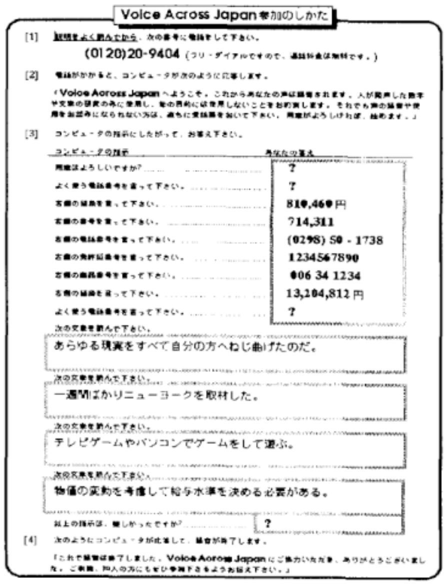
Are you ready to start?	(your response)
Are you calling from your home phone?	(your response)
Do you speak any language besides English at home?	(your response)
•	•
•	•
•	•

Please read your panel identification number which you filled in below:

(write your panel identification number here)

Thank you. You will now be asked to read each of the items in the right-hand column.

1.	*(a word)*	subtract
2.	*(a sentence)*	Could you give me a list of all afternoon flights from Los Angeles
3.	*(a telephone number)*	(379) 528-5883
4.	*(a place)*	Canton, China
5.	*(a spelled word)*	M, A, N, K, O, S, K, I



Voice Across Japan 参加のしかた

[1] 説明をよく読んでから、次の番号に電話をして下さい。

(0120)20-9404 (フリーダイアルですので、通話料金は無料です。)

[2] 電話がかかると、コンピュータが次のように応答します。

「Voice Across Japan へようこそ。これからあなたの声は録音されます。人が発声した数字や文章の研究のみに使用し、他の目的には使用しないことをお約束します。それでも声の録音や使用をお望みにならない方は、直ちに受話器を置いて下さい。用意がよろしければ、始めます。」

[3] コンピュータの指示にしたがって、お答え下さい。

コンピュータの指示	あなたの答え
用意はよろしいですか?	?
よく使う電話番号を言って下さい。	?
右側の金額を言って下さい。	810,460円
右側の番号を言って下さい。	714,311
右側の電話番号を言って下さい。	(0298) 50 - 1738
右側の免許証番号を言って下さい。	1234567890
右側の郵便番号を言って下さい。	006 34 1234
右側の金額を言って下さい。	13,204,812円
よく使う電話番号を言って下さい。	?

次の文章を読んで下さい。

あらゆる現実をすべて自分の方へねじ曲げたのだ。

次の文章を読んで下さい。

一週間ばかりニューヨークを取材した。

次の文章を読んで下さい。

テレビゲームやパソコンでゲームをして遊ぶ。

次の文章を読んで下さい。

物価の変動を考慮して給与水準を決める必要がある。

以上の指示は、難しかったですか? ?

[4] 次のようにコンピュータが応答して、録音が終了します。

「これで録音は終了しました。Voice Across Japan にご協力いただき、ありがとうございました。ご家族、知人の方にもぜひ参加下さるようお伝え下さい。」

Figure 2-3: These are sample prompts from the American and Japanese contribution to the *PolyPhone* project. Countries were tasked with funding the collection of their own speech corpus. Many employed market-research firms to identify potential participants and mailed them prompts to be read aloud over the phone.

entirely automated collection protocol [48]. Since this corpus was meant to contain spontaneous speech, however, when a subject called in, a database of participants was queried and an outgoing call was made in an attempt to find a conversational partner. A topic of conversation was suggested, and the participants were given the power to signal when recording should begin. 2,430 conversations were collected, which, on average, lasted around six minutes each, resulting in over 240 hours of audio. The participants self-rated the naturalness of their conversations on a Likert scale where (1) was very natural and (5) indicated conversation that felt forced. The average rating was 1.48, indicating that the collection methodology was successful in capturing natural, spontaneous speech.

In the mid-90s, countries from across the world were submitting their contributions to a set of corpora collected under the *PolyPhone* project [9, 82, 142]. These projects also used PCs outfitted to collect audio over standard phone lines. In the *Macrophone* project in the U.S., prompt sheets such as the one in Figure 2-3, were mailed to participants. Over the course of a six week period, the project was able to collect around 200,000 utterances from 5,000 speakers. In Japan, a corresponding project later collected 8,866 speakers and validated 122,570 utterances, discarding those that contained noise or hesitations.

Once the research community overcame the technological hurdle of how to automate the collection of large speech corpora, there remained the question of how to recruit participants affordably. Many collection efforts, such as for the NYNEX *PhoneBook*, found success using outside market-research firms to gather participants. The *PhoneBook* corpus now contains over 1,300 speakers reading around 75 isolated words each [117]. The LDC's *CALLHOME* corpus [23], applied an innovative technique to the recruitment problem. For this collection, participants were given the ability to make free long-distance phone calls in exchange for having their voices recorded. When the *Fisher* corpus was collected [35], a more typical internet marketing strategy was employed. In this case, however, after participants were registered, the system actively called them to request participation. This system-initiated protocol has been continued more recently in the Mixer corpora for speaker recognition [31].

It is clear from these example corpora, that the speech community has been crowdsourcing for quite some time. One novelty, however, is that we have begun to automate the verification process in addition to the collection itself. A group from Nokia, for example, describe the *CrowdTranslator*, for which corpora in a target language are collected using prompts over mobile phones for low resource languages [90]. Verification was performed by comparing answers to redundant prompts given to workers within individual sessions.

2.2.2 Spoken language systems

While in some cases corpus collection is performed as an independent step from the speech systems built in the target domain, with many spoken language systems there is a chicken-and-egg problem in which data are needed to construct a system, but the system is also needed to provide proper context for data collection. With such applications, a bootstrapping method is often applied to iteratively edge closer to natural, in-domain data.

The Air Travel Information System (ATIS) domain exemplifies this paradigm well. Initially, a wizard-of-oz style collection procedure was used, whereby subjects brought into the lab were led to believe that they were speaking with a computer, when in fact a human transcriber was handling their flight-related queries. [61]. Sessions were collected at a slow pace of one per day for a period of about eight weeks, yielding 41 sessions containing 1041 utterances. Later, the collection effort was distributed to multiple sites, including AT&T, BBN, CMU, MIT, and SRI [64]. By 1994, most sites were automating their collection procedure using the spoken dialog systems built for the domain [33].

Around the same time, in Europe, similar efforts were in place [115]. There was also interest in another travel domain. A system developed in Germany allowed members of the public to call in to get train time-table information [4]. Described in this work are a number of pitfalls associated with making such systems widely available. First, there was a struggle to receive enough calls for development purposes before releasing to the general public. Second, technical difficulties with the telephone interface only allowed a single caller through at a time. Finally, the calls themselves varied enormously in almost every aspect. Some users would ask non-sensical questions of the system, background noise was a common phenomenon, and the variation in natural language, even for a response as simple as an affirmation, came in a variety of forms (e.g. "yes", "great", "sure", "perfect", "okay", etc).

In subsequent years, an increasing number of institutions began to tackle the task of making spoken language systems widely accessible. The DARPA *Communicator* project renewed interest in the air travel domain [140]. With nine groups participating, this time a common architecture for dialogue system development was used and a telephone interface was deployed for short-term data collection [129]. Examples of longer-term deployments can be seen in AT&T's *How may I help you* customer care dialog system [49], MIT's Jupiter weather information system [153], and CMU's Let's Go Bus information system [120].

The complexity involved in releasing a spoken language system into the wild is difficult to overstate. Maintaining a constantly running suite of software that incorporates cutting-edge research is a tricky balance. Still, the benefits to deploying systems early, and iterating on development are clear. Figure 2-4 shows the performance of the Jupiter weather information system over a number of development cycles and release dates. Adjusting the axis, it is clear that the word error rate decreases logarithmically with increasing amounts of data, reinforcing the old adage: "There's no data like more data."

2.2.3 User-configured recording environments

In the last decade, researchers have begun to explore a new possibility for speech collection via light-weight client-configurable recording tools. Whether through the web or on a mobile device, these user-driven solutions come with a new set of advantages and disadvantages. Whereas the phone provides a somewhat consistent channel over which the voice is transmitted, collecting speech directly from a user's device requires individuals to record from the equipment they have at their disposal, whether it is a built-in microphone or a cheap head-set microphone. Misconfigured microphones can also be a source of variability. An inexperienced user may not be able to recognize clipping in the audio, let alone find the controls to turn down the recording volume.

Still, the advantages of recording audio in this fashion are many. First and foremost, the tools necessary to perform collection are easier to deploy and load-balance than those required for phone-based collection. An early example of web-based speech data collection

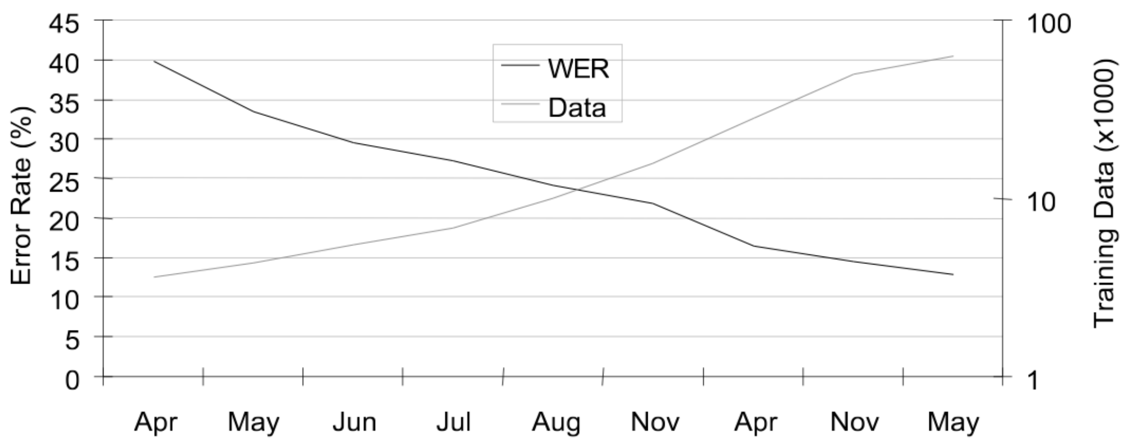


Figure 2-4: The plot above depicts two years of development early in the life of the Jupiter weather information dialogue system. As more training data were collected, new recognizers were released, causing a steady improvement in performance. Note the non-linear x-axis corresponding to the release dates of updated systems.

comes from Germany, where a team gathered read speech from adolescents across the country using a downloadable audio recording tool called SpeechRecorder [41]. SPICE [125] and the WAMI Toolkit [52] were two early examples that also used Java to record audio directly from a web-interface and upload it to a server behind-the-scenes. WAMI was even able to simulate streaming via the chunking protocol already available in HTTP 1.1. More advanced streaming, with data compression, can be performed through a Flash Media Server.

Other platforms rely on mobile applications (apps) to collect audio. Lane et al., for example, developed a suite of tools specifically for data collection through crowd-sourcing venues such as Amazon Mechanical Turk [86]. AT&T developed a platform for speech mashups for mobile devices, [38]. The development kit contained a native client for the iPhone and a plug-in for Safari that allowed speech applications to connect to a speech recognizer running remotely.

By attaching a remote speech recognizer to the client, users can interact with full-fledged spoken dialog systems from their personal computers or mobile devices. In 2007, *CityBrowser* was made publicly available using the WAMI Toolkit and email lists were used to recruit subjects for a user study [54]. An iPhone app was later developed for the WAMI Toolkit, and *Flight Browser*, the latest incarnation of the ATIS and Communicator projects, was given a mobile interface.

While mobile interfaces can be attractive, they can require a considerable amount of developer time and energy. This is compounded by the fact that the popular mobile phone operating systems of today are entirely incompatible with one another, making it difficult to minimize the effort of reaching a large audience through smartphones. The approach we have adopted at MIT is to build a generic webkit-based app capable of streaming speech to a server for iOS and Android. We then implement prototypes in JavaScript and HTML and run them on a desktop browser, or in one of these speech-enabled mobile browsers. This gives us the luxury of a write-once-and-run-anywhere deployment protocol.

The use of web technologies in crowd-sourcing for speech is not a new phenomenon. In the mid 90s, a team from MIT combined their telephone-based collection protocol with a web prompting interface to collect read speech [66]. This phone-web hybrid strategy has continued today in other groups with full-fledged spoken dialogue systems. In [72, 45], workers were given a number to call and a task to follow via the Amazon Mechanical Turk web interface.

The rise of Web 2.0 technologies allows for more dynamic interaction, however, when the speech recording software is embedded directly within the web page. YourSpeech from the Microsoft Language Development Center offers both a Quiz game and a text-to-speech (TTS) generator to incentivize users to participate. The TTS generator prompts users for speech and produces a concatenative speech synthesizer once enough data are received [44]. In [106], our flight information system was evaluated and over 1,000 dialogue sessions were collected by deploying a unified graphical user interface to Amazon Mechanical Turk, combining speech input with visual feedback. Another group subsequently used mTurk to evaluate a somewhat simpler interface for language learning [121]. For an overview of some of the speech acquisition tasks performed through modern-day crowdsourcing, see [114].

2.3 Automatic Speech Recognition

The trend in spoken language systems, particularly within the recognizer itself, has been towards ensuring that its components are stochastic, and thus can be trained using large datasets. Having described the literature showing that crowdsourcing may be a good source of data, we focus this section on the models that we are trying to improve and the types of data required to improve them. This thesis concentrates on three such models all found within a speech recognizer itself: the language, lexical, and acoustic models. This section describes the overall architecture of a speech recognizer, how these models fit into this framework, and their basic training or adaptation procedures.

2.3.1 The Fundamentals

The fundamental goal of an automatic speech recognizer is to determine the most likely string of words contained in an utterance that has been mapped to a sequence of acoustic observations. A probabilistic formulation might look like this:

$$W^* = \arg\max_W P(W|A) = \arg\max_W P(A, W)$$

where W represents a sequence of words, and A represents a series of acoustic observations. We might decompose our representation further into a sequence of sub-word units U, making the independence assumption that the acoustics are independent of the words given this sequence:

$$P(A, W) = \sum_U P(A|U)P(U|W)P(W)$$

We have now recovered the three central components of a speech recognizer. $P(A|U)$ is often referred to as the acoustic model. $P(U|W)$ represents a lexicon, which maps words to their pronunciations in units such as phones or phonemes. Finally, $P(W)$ is referred to as the language model and captures statistics over word sequences.

This thesis makes use of the SUMMIT speech recognizer [47]. One defining characteristic of this recognizer is that it is segment-based rather than frame based. The typical model topology used in speech recognition is a Hidden Markov Model (HMM), which operates on feature vectors in fixed intervals [119]. SUMMIT, however, performs a preprocessing step that searches for significant changes in the features and lays down landmarks over which the search is eventually performed. To account for this in our equations we say that A is actually a series of observations O over a particular segmentation S of the utterance. To keep operations on the recognizer tractable we make the assumption that, for a given sequence of words, the total likelihood of the underlying unobserved sequences is roughly equivalent to their maximum likelihood. This type of approximation is often called a *Viterbi* approximation, after the inventor of the dynamic programming algorithm often used by recognizers to find the most likely sequence of words during decoding.

$$\begin{aligned} W^* &= \arg\max_W \sum_{S,U} P(O|S,U)P(S|U)P(U|W)P(W) \\ &\approx \arg\max_{S,U,W} P(O|S,U)P(S|U)P(U|W)P(W) \end{aligned}$$

$P(S|U)$ is the model topology. It may be used to model the duration of sub-word units, perhaps through transition probabilities between states. This mathematical framework allows us to represent both HMM-recognizers and segment-based recognizers simultaneously. Indeed, above the acoustic model, we represent all of our distributions using the same generic framework: finite state transducers (FSTs) [63].

Roughly speaking, a weighted FST is a graph with an initial state, a final state, and where each edge contains an input, an output, and a weight. Precise definitions and algorithms that operate on FSTs can be found in [107]. It will suffice to note that an FST can represent a distribution $P(X|Y)$ by containing a path with inputs for each sequence $Y = \{y_1 \dots y_m\}$ and outputs for a sequence $X = \{x_1 \dots x_m\}$ such that the product of the weights along the path is $P(X|Y)$. Note that in practice, we often put negative log probabilities on the arcs, for numerical stability, making a summation the operation of choice to aggregate probabilities. An FST can also represent a distribution $P(Z)$ by ensuring that the output and input on each edge are identical. Operations on probability distributions translate directly to operations on FSTs. One of the most useful operations is composition, denoted by $\circ$. We might represent the product $P(U|W)P(W)$ with the composition of a lexicon and a grammar, $L \circ G$.

In the remainder of this section we look at the individual components of the speech recognizer. For each component we describe common ways in which it is modeled and describe the basics of how it is trained.

2.3.2 The Language Model

A language model represents the *a priori* probability of a sequence of words. $P(W)$ can be parameterized in a number of ways. The most common approaches are arguably through a context-free grammar (CFG) or an N-gram.

A CFG is a formal grammar that contains a set of production rules which define the sequences of *terminals* that are contained in the language. The terminals of a grammar for recognition are the words that can be recognized. To be context-free, the left-hand-side of every production rule must only contain a single *non-terminal*. In this sense, a non-terminal is just a name for a rule. Non-terminals can appear on the right-hand-side of a production rule, but to define a string in the language, all such non-terminals must be expanded out into terminals. The Java Speech Grammar Format[2] (JSGF) is a specification for a CFG, with some syntactic sugar to enable us to write the CFG compactly.

```
#JSGF V1.0;
public <top> = <greet>*;
```

[2]http://java.sun.com/products/java-media/speech/forDevelopers/JSGF/

```
<greet> = Hello (<who> | <what>);
<who> = Newman;
<what> = World;
```

The CFG shown above is unweighted, but could easily be supplemented with probabilities for each rule. The $\star$ means that the non-terminal can be repeated zero or more times. The | signifies a logicial *or*, indicating either `<who>` or `<what>` may be used. This extra syntax does not change the expressive power of a CFG. It should be noted that a CFG is actually *more* expressive than an FST, but we often approximate them as FSTs for use in a recognizer [116].

CFGs are most often written by hand and left unweighted. Although statistics over rules can be learned, N-grams provide an alternative approach which can also be represented in FST-form. An N-gram can be expressed mathematically as follows:

$$P(W) = P(w_1, \dots, w_K) = \prod_{i=1}^{K} P(w_i|w_1 \dots w_{i-1}) = \prod_{i=1}^{K} P(w_i|w_{i-(N+1)} \dots w_{i-1})$$

The N-gram makes the assumption that a word is dependent only on the N previous words. At the word level, trigrams are a popular size of N-grams, but certain recognizers have been known to use 4-grams. The statistics can be learned directly from a corpus of text. Preferably, the text would match the recognition domain as closely as possible. Since spoken language can be agrammatical and contain disfluencies, the ideal data are transcribed speech in the target domain.

Certain distinct words in a language share similar contexts, and can be grouped together into classes. Suppose, for example, that a recognizer was built for spoken language access to a restaurant review database. Two queries that might be given to the system are *"Is the french food at Cuchi Cuchi good?"* and the phrase *"Is the chinese food at Royal East good?"* Without classes, the statistics over which words might precede *good* are fairly fragmented. With classes, however, both queries might reduce to *Is the CUISINE food at RESTAURANT good?* A standard N-gram can be built over transcripts of this form, and the individual classes can internally be a uniformly weighted list of entries or may have statistics weighting each entry in the list. In many cases, classes are determined automatically based on the statistics of the corpus [17].

Support for classes in SUMMIT is even more general, in that the recognizer allows for the inclusion of dynamic classes that can be changed at run-time. The restaurant application described above, for example, would be able to load new restaurants into the recognizer on-the-fly. Beyond simple weighted word-lists, arbitrary FSTs can be spliced into the search

(1)	n ih r ax s td	t ao r n ey df ow
(2)	n ih axr s	t er n ey dx ow
(3)	n ih r ax s tcl	t ao r n ey dcl d ow

Table 2.1: Pronunciations for the phrase “nearest tornado”, exemplifying the difference between phonemes and phones.

space. In Chapter 5, we make use of this feature to dynamically update the recognizer for a cinema voice search domain.

2.3.3 The Lexicon

The lexicon, $P(U|W)$, is a mapping from words to pronunciations. The units that make up the sequence U are those over which the acoustic model is built. In the simplest case, these might be basic speech units called *phones*. More often, though, we place context-dependencies on these sub-word units so that our acoustic models may be built over diphones or triphones.

Since the conversion from a sequence of phones to a sequence if diphones is deterministic, the recognizer need only store a mapping from words to phone sequences. Sometimes an additional layer of indirection is added to account for phonological variation [59]. In SUMMIT, phonological rewrite rules are implemented to provide a mapping from semantically contrastive units called phonemes to the acoustically contrastive units called phones, which are more numerous. Take, for example, the words *nearest tornado*, in Table 2.2.

The phoneme representation is shown in line (1) and two different phone sequences follow in lines (2) and (3). Notice the difference in consonant closures and even vowel substitutions. When spoken fluently, as in line (2), the t in *nearest* is not pronounced. When a slight pause is introduced between the two words, as in line (3), both t-closures might be present.

The phonological rules can also be represented as an FST. Wrapping the various levels of representation into one recognition framework using FST composition can be done as follows:

$$R = C \circ P \circ L \circ G$$

The pronunciations in a lexicon can be weighted with probabilities, but it is common for researchers to leave lexicons unweighted. When a lexicon is weighted, we have

$$P(U|W) = P(B|W) = \prod_{i=1}^{K} P(\mathbf{b}_i|\mathbf{w}_i)$$

$\mathbf{w}$	=	c	o	u	p		l	e
$\mathbf{b}$	=	k	ah		p	ax	l	
	=	k		ah	p	ax	l	
$\mathbf{g}_1$	=	c/k	o/ah	u/ϵ	p/p	ϵ/ax	l/l	e/ϵ
$\mathbf{g}_2$	=	c/k	o/ϵ	u/ah	p/p	ϵ/ax	l/l	e/ϵ

Table 2.2: Two example alignments for the word couple and its phonemic pronunciation.

where $\mathbf{b}_i$ is a sequence of phonetic units representing the pronunciation of word, $\mathbf{w}_i$.

As mentioned in the *language model* section above, our recognizer can load new words on-the-fly. This means that we must be able to add pronunciations to the lexicon for previously unseen words. This is typically accomplished with a letter-to-sound (L2S) model. Modeling the mapping between letters (graphemes) and phonetic units has been the subject of much research [27, 16, 74, 128, 97, 8, 87, 70]. Some of this work is rule-based, [74], while others learn sub-word units automatically, [16, 126], and still others perform hybrid of the two [128].

This thesis makes use of the recent research of Bisani and Ney to learn a mapping from graphemes to phones or phonemes [16]. The basic sub-word unit in this framework is thus called a *graphone* or *graphoneme*, or more generally, a joint multigram. Each graphone, g, consists of zero or more graphemes mapped to zero or more phonetic units. Mapping the empty string, ϵ, to another ϵ is disallowed. In this work, we restrict our attention to *singular* graphone(eme)s, for which there is at most one letter mapping to at most one phonetic unit. A particular spelling and pronunciation can therefore be represented as a sequence of joint multigrams, $\mathbf{g}$. It is important to note that a particular spelling and pronunciation may be segmented in more than one way. Table 2.2, for example, provides two sequences for a single pronuciation for the word *couple*.

For a given sequence of graphemes, $\mathbf{w}$, the goal now is to model a joint probability with a pronunciation, $\mathbf{b}$, as follows:

$$P(\mathbf{w}, \mathbf{b}) = \sum_{\mathbf{g} \in S(\mathbf{w},\mathbf{b})} P(\mathbf{g}) \approx \max_{\mathbf{g} \in S(\mathbf{w},\mathbf{b})} P(\mathbf{g}) \tag{2.1}$$

Here again, we are making the Viterbi approximation by making the assumption that the most likely sequence is about equal to the sum over all sequences that segment a particular word/pronunciation pair.

The task now becomes a problem of modeling $P(\mathbf{g})$. Fortunately, we already have the tools, since we can create a language model over joint multigram units which has this very form. The details of the training procedure for this model are found in [16]. Given an existing lexicon containing mappings $\mathbf{w} \rightarrow \mathbf{b}$, they use Expectation Maximization (EM), [37], to simultaneously discover graphones and their N-gram weights using Expectation Maximization. This language model can also be put into FST form, with letters on the input and phonetic units on the output.

Aside from the joint multigram training procedure, which requires a large lexicon of expertly generated pronunciations, we have not described how one might go about automatically improving a lexicon. In particular, we would like to find a way for non-experts to contribute to the training or adaptation procedure. We achieve this in Chapter 5, where we show how to generate better pronunciations using spoken examples, which can come from a crowd of non-experts.

2.3.4 The Acoustic Model

Most treatments of acoustic modeling describe this task as finding a parameterization of $P(A|W)$ [65, 71]. Sine the layers between the language model and acoustic model are typically deterministic, or at least unweighted, they are not a source of uncertainty that needs to be explicitly modeled. Even a lexicon with multiple pronunciations for one word is typically left unweighted in modern recognizers. In this work, however, we prefer to consider the lexicon as yet another place to model uncertainty, and therefore let the acoustic model represent $P(A|U)$. That is, the acoustic model represents the probability of a set of acoustic observations given a sequence of phonetic units.

As previously described, once a model topology is chosen, the task reduces to modeling the probability of a sequence of observations $O = o_1 \ldots o_T$ given the segmentation S and units U, which may be context-dependent phonetic units. Here the details differ slightly between a segment-based recognizer and an HMM-based recognizer, especially when one builds a model over individual segments rather than simply their landmarks. We refer the reader to [47] for an explanation of the extra normalization requirements of the segment models, and describe a set of acoustic models over the boundaries. The boundary models are essentially diphones with some additional labels representing the possibility of a boundary landing in the middle of a phone. We assume, given a segmentation and a sequence of phones, that the observations are independent of one another.

$$P(o_1 \ldots o_T|U, S) = \prod_{i=1}^{T} P(o_i|u_{s_i})$$

Now we need to model the probability of a single observation vector given a particular acoustic unit (e.g. a diphone). This can be done with a mixture of Gaussian distributions.

Each Gaussian mixture model (GMM) can be parameterized for a particular unit, u_k, by a set of means, $\{\mu_j^k\}$, covariance matrices, $\{\Sigma_j^k\}$, and weights ,$\{\alpha_j^k\}$, where $\sum_j \alpha_j^k = 1$. The following equations outline the basic formulation of a single acoustic model for a unit u_k.

$$P(o_i|u_k) = \sum_{j=1}^{N} \alpha_j^k P(o_i|\mu_j^k, \Sigma_j^k)$$

A normal distribution describes each component of the mixture model. Thus, for component j, we have:

$$P(o_i|\mu_j^k, \Sigma_j^k) = \frac{1}{(2\pi)^{\frac{d}{2}}|\Sigma_j^k|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(o_i - \mu_j^k)^T(\Sigma_j^k)^{-1}(o_i - \mu_j^k)\right)$$

The feature space of a speech recognizer can be derived from the signal in a number of ways. The most common approaches are via Mel-frequency or perceptual linear predictive (PLP) cepstral coeffcients [62]. Averages and derivatives of these basic building blocks can form a high-dimensional feature vector. Principal component analysis is then performed to reduce the dimensionality and whiten the feature space. Information regarding the details of core acoustic modeling concepts – GMMs, MFCCs and more – can be found in the latest edition of *Spoken Language Processing*, [71].

In the fully supervised case, an acoustic model would ideally have gold-standard segmentations and phonetic labels for a large corpus of utterances. This is extremely expensive to obtain, so researchers most often rely on data that are transcribed at the word-level. The phonetic labels and the segmentation are then estimated for a large corpus by forcing an alignment of words to the acoustic signal using an initial set of models, perhaps trained on a much smaller set of hand-aligned utterances. When beginning with a new domain, e.g. training acoustic models for mobile phones, one can start with an out-of-domain model to generate alignments. These alignments can then be used to train an in-domain model.

Sometimes there is not enough data in a new domain to train a full set of new acoustic models. Since we would still like to leverage this new data, there has been a lot of research regarding the *adaption* of a set of previously trained acoustic models to a new domain while avoiding data sparsity issues. One approach is called maximum *a posteriori* (MAP) adaptation [94]. In MAP adaptation the parameters themselves, $\Phi^k = \{\alpha^k, \mu_j^k, \Sigma_j^k\}$, are assumed to be a random variable drawn from a *prior* distribution $P(\Phi)$. Given a new data set of observations and alignments, we compute the MAP adaptation parameters with:

$$\Phi_{MAP} = \arg\max_\Phi P(\Phi|X) = \arg\max_\Phi P(X|\Phi)P(\Phi)$$

It's common to approximate MAP adaptation with model interpolation, whereby the parameters from both models are used, and the probabilities from each are interpolated using an empirically determined weight. One advantage of this approximation is that it does not require the choosing of a prior.

2.4 Summary

This chapter has provided background material and related work in three areas relevant to this thesis. We have summarized some of the relevant literature regarding crowdsourcing, and we have described an important subclass of crowdsourcing called *human computation* that removes the expert from the higher level process and uses members of the crowd much like method calls in an ordinary program. We then give a short history of speech data collection, and note that such undertakings, especially for spoken language systems, have required extraordinary effort. Finally, we have given a brief overview of speech recognition. In particular, we have described three stochastic models, the language model, the lexicon, and the acoustic model. These models will become the focus of Chapters 4, 5, and 6 respectively.

Chapter 3

Collecting Speech from the Crowd

Before delving into the crowd-supervised training of the components of a speech recognizer, we devote this chapter to the crowdsourced collection and transcription of speech data. These tasks are typically expensive and time-consuming in and of themselves. In particular, we explore the use of Amazon Mechanical Turk to significantly reduce the cost and increase the speed of speech data collection. More generally, we argue that Voice over IP (VoIP) and cloud computing are poised to greatly reduce impediments to research on spoken language interfaces in many domains.

As shown in the previous chapter, acquiring in-domain training data for spoken language systems is central to their development. Unfortunately, this gives rise to a classic chicken-and-egg problem. A working system is required to collect in-domain data; however, this very data is needed to train the underlying models before the system can be implemented effectively. Over the years, we have learned to bootstrap our acoustic and language models from existing systems. This is typically a slow process that can take years of data collection and iterative refinement, a point well-illustrated by Figure 2-4, which depicts a log-linear relationship between the amount of data collected and the improvements in terms of word error rate.

Data collection has always been a major issue for researchers developing conversational spoken dialogue systems. The ATIS project, described in the previous chapter, collected approximately 25,000 utterances in the travel domain in around two years, with contributions coming from multiple sites [64]. In 2000 and 2001, with the help of additional participants, multi-site data collection resumed in the travel domain under the *Communicator* project. This led to the collection of just over 100,000 utterances spoken to automatic or semi-automatic flight-reservations systems [139]. A similar amount of data was collected by MIT through *Jupiter* [153], a publicly deployed weather information system. In general, however, not every domain can receive the sort of attention that wide-spread deployment or multisite data collection affords. Since the collection of large amounts of data for arbitrary domains is prohibitively expensive, researchers often resort to conducting small user studies with high management overhead. This prevents data-driven techniques from reaching their full potential.

The premise of this chapter is that crowdsourcing can address some of these needs. Researchers in the natural language processing community have already begun to harness mTurk for data collection [73]. The speech community, however, has been somewhat slower to capitalize on this new paradigm. While speech researchers have used mTurk largely for transcription [110, 99], we believe it is natural to turn our attention to the *collection* of in-domain data for spoken language systems. From a technical standpoint, this endeavor is somewhat more complex. Amazon Mechanical Turk does not provide an interface for web-based audio collection; furthermore, while incorporating audio playback into a website is relatively straightforward, few tools exist for recording audio from web pages. Appendix A describes the state-of-the-art technology capable of audio collection over the web.

For this work, we have used our open-source Web Accessible Multimodal Interfaces (WAMI) Toolkit, which provides a Javascript API for speech-enabling a web-site [52]. A similar technology from AT&T is also in development [38]. The WAMI toolkit was originally tested by soliciting remote subjects to solve scenarios within the *CityBrowser* multimodal dialogue system [51], and rewarding them substantially with gift certificates. While this method was successful, we feel that Amazon Mechanical Turk potentially offers a much more economic solution and a much broader user base. With this tool, it is now feasible to integrate these speech recognition services with Web 2.0 interfaces deployed directly to Amazon Mechanical Turk. Appendix D provides a tutorial that walks the reader through the deployment of a simple speech collection task to mTurk using these open-source tools.

This chapter details two experiments that make use of our open source WAMI Toolkit to collect corpora in two different domains which cumulatively constitute over 113 hours of speech. The first corpus contains 100,000 utterances of read speech, and was collected by asking workers to record street addresses in the United States. For the second task, we collected conversations with *Flight Browser*, a multimodal spoken dialogue system. The *Flight Browser* corpus contains 10,651 utterances composing 1,113 individual dialogue sessions from 101 distinct users. The aggregate time spent collecting the data for both corpora was just under two weeks. For the prompted speech collection task, our servers were logging audio from workers at rates faster than real-time. We describe the process of collection and transcription of these corpora while providing an analysis of the advantages and limitations of this data collection method.

3.1 Read Speech

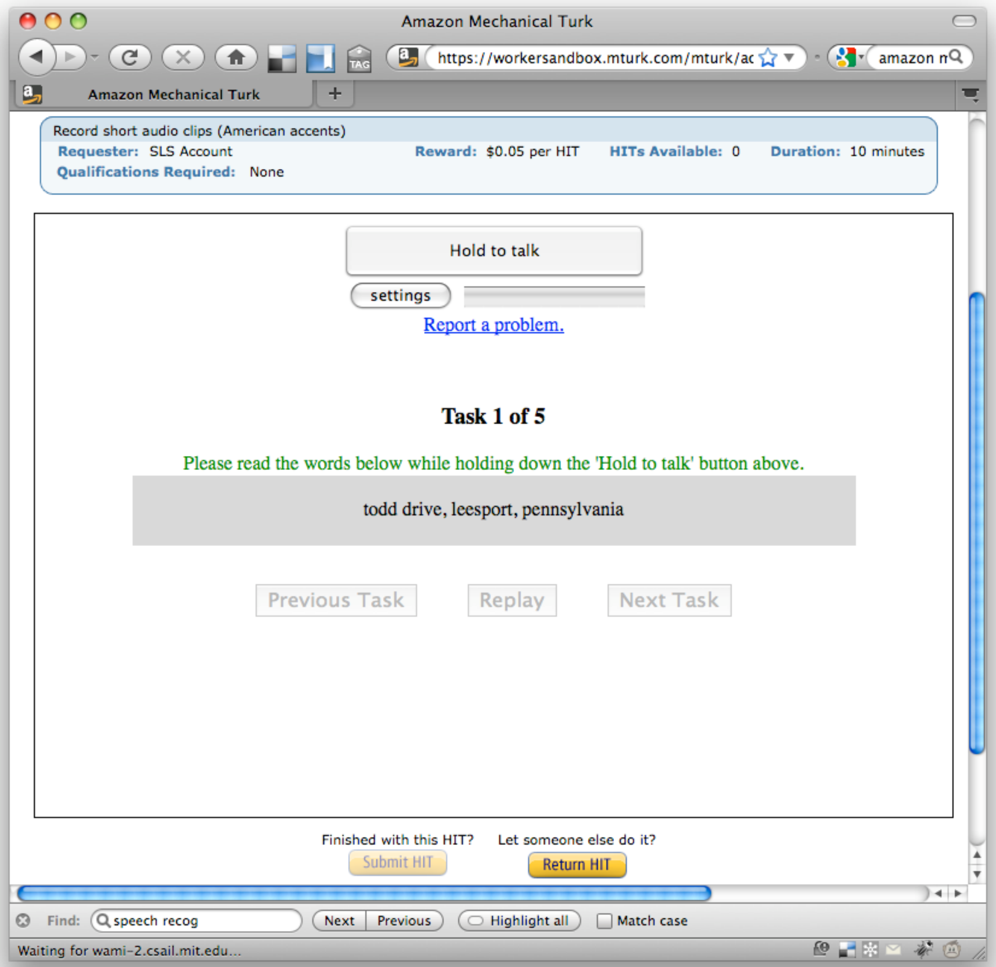
This section explores the use of Amazon Mechanical Turk to collect read speech containing spoken addresses. The Spoken Language Systems group has, for some time, been interested in systems where points-of-interest are spoken by the user, often using an address [55]. Over the years, we have collected data in this domain by bringing people into the lab or through ad-hoc web interfaces, but have never undertaken a marketing campaign to ensure heavy usage. Given the millions of possible addresses that a user might speak, it is discouraging that we had collected so little data. In this pilot experiment, we distribute a speech-enabled web interface using Amazon Mechanical Turk, and elicit a total of 103 hours of speech from 298 users. This simple task demonstrates the feasibility of large scale speech data collection through mTurk.

3.1.1 Collection Procedure

Recall that units of work on Amazon Mechanical Turk are called Human Intelligence Tasks or HITs. A requester can build HITs and deploy them to mTurk using a web interface, command-line tools, or another of the many APIs made available to the public. While each task is assigned a price, which an individual worker might receive as payment, requesters reserve the right to deny payment for work that is unsatisfactory.

The addresses in our reading task are taken from the TIGER 2000 database provided by the U.S. Census Bureau. Each address is a triplet: (*road, city, state*). There are over six million such triples in the TIGER database. To ensure coverage of the 273,305 unique words contained in these addresses, we chose a single address to correspond to each word. 100,000 such triples formed our pilot experiment; mTurk workers were paid one U.S. cent to read each prompt. Figure 3-1 shows an example HIT. After the worker has recorded an address, they are required to listen to a playback of that utterance before moving on, to help mitigate problems with microphones or the acoustic environment.

Since we are employing anonymous, non-expert workers, there is little incentive to produce high quality utterances, and a worker may even try to game the system. We propose two distinct ways to validate worker data. The first is to have humans validate the data manually. Given the success of mTurk for transcription tasks in previous work, we could theoretically pay *other* workers to listen to the cloud-collected speech and determine whether the expected words were indeed spoken. At this stage, redundancy through vot-



Record short audio clips (American accents)
Requester: SLS Account Reward: $0.05 per HIT HITs Available: 0 Duration: 10 minutes
Qualifications Required: None

Task 1 of 5

Please read the words below while holding down the 'Hold to talk' button above.

todd drive, leesport, pennsylvania

Finished with this HIT? Let someone else do it?

Waiting for wami-2.csail.mit.edu...

Figure 3-1: A sample Human Intelligence Task (HIT) for collecting spoken addresses through Amazon Mechanical Turk.

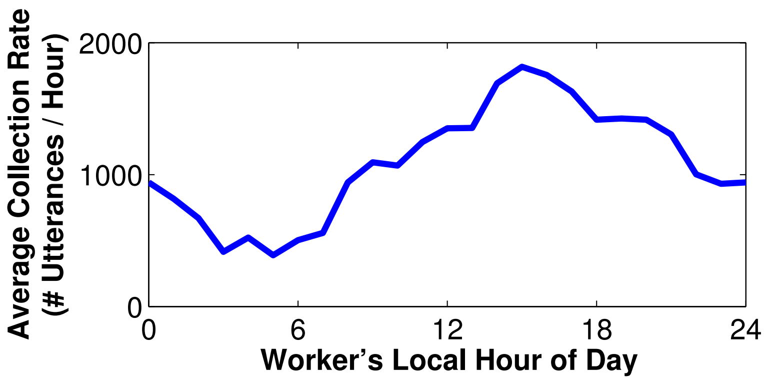


Figure 3-2: Data collection rate as a function of the worker's local time of day. Note that different time zones even out the actual collection rate.

ing could be used to verify the speech verification. A less expensive approach, which we explore in this section, is to integrate a speech recognizer into the data-collection process itself.

Since the VoIP interface we employ is used by our dialogue systems, we have the ability to incorporate the recognizer in real time. Thus, we can block workers who do not satisfy our expectations immediately. For the pilot experiment, however, we decided not to block any workers. Running the recognizer in a second pass allows us to examine the raw data collected through mTurk and experiment with different methods of blocking unsuitable work, which might be deployed in future database collection efforts.

3.1.2 Corpus Overview

Our reading tasks were posted to mTurk on a Wednesday afternoon. Within 77 hours, 298 workers had collectively read all 100,000 prompts, yielding a total of 103 hours of audio. Figure 3-2 depicts the average number of utterances collected per hour plotted according to the worker's *local* time of day. Workers tended to talk with our system during their afternoon; however, the varying time zones tend to smooth out the collection rate with respect to the load on our servers.

The majority of our data, 68.6%, was collected from workers within the United States. India, the second largest contributor to our corpus, represented 19.6% of our data. While some non-native speakers produced high quality utterances, others had nearly unintelligible accents. This, as well as the fact that the acoustic environment varied greatly from speaker to speaker, make the MIT Address corpus particularly challenging for speech recognition.

To determine the properties of our corpus without listening to all 103 hours of speech, two researchers independently sampled and annotated 10 utterances from each worker.

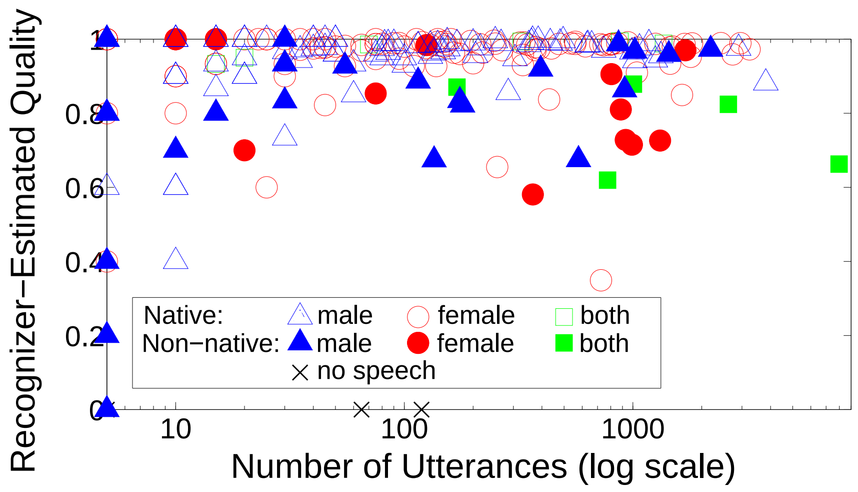


Figure 3-3: Individual workers plotted according to the recognizer estimated quality of their work, and the number of utterances they contributed to our corpus. Note that some Amazon accounts were used by multiple individuals. Those that were detected as having both male and female speakers were labeled “both” in this plot.

Speakers were marked as male or female and native or non-native. Anomalies in each utterance, such as unintelligible accents, mispronounced words, cut-off speech, and background noise, were marked as present or absent. We then extrapolate statistics for the overall corpus based on the number of utterances contributed by a given worker. From this, we have estimated that 74% of our data is cleanly read speech.

This result raises the question of how to effectively manage the quality of speech collected from the cloud. Here we explore an automatic method which incorporates our speech recognizer into the validation process. In particular, we run the recognizer that we have built for the address domain over each utterance collected. We then assign a quality estimate, q, to each worker by computing the fraction of recognition hypotheses that contain the U.S. *state* expected given the prompt. Figure 3-3 shows the recognizer-estimated quality of each worker, plotted against the number of utterances that worker contributed. Notice that a single worker may actually be two or more different people using the same Amazon account.

mTurk provides requesters with the ability to block workers who do not perform adequate work. Using our automatic method of quality estimation, we simulate the effects of blocking users according to the quality threshold q. It is clear from Figure 3-4 that, while the data collection rate might have slowed, requiring a high q effectively filters out workers who contribute anomalous utterances. Figure 3-5 depicts how the corpus properties change when we set $q = .95$. While not unexpected, it is nice to see that egregiously irregular utterances are effectively filtered out.

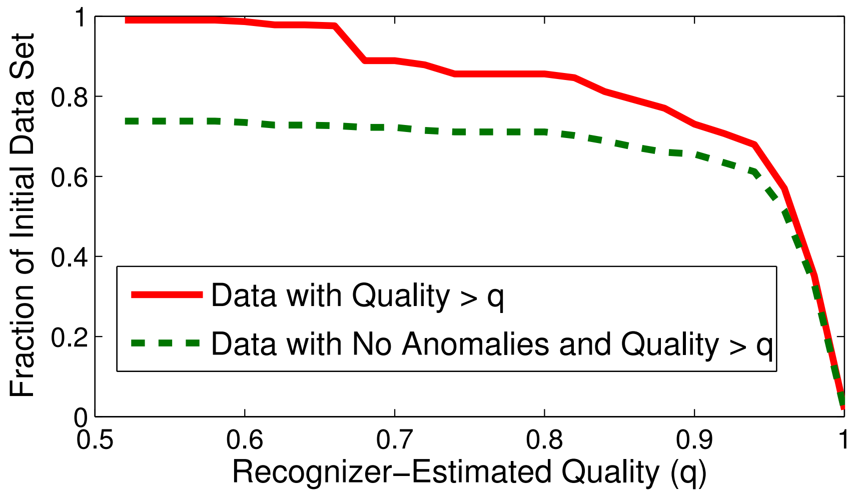


Figure 3-4: The quantity and quality of recognizer-filtered sub-corpora of spoken address data. By filtering out users whose estimated quality does not meet a certain threshold, we can simulate the effect of using the recognizer to automatically block workers.

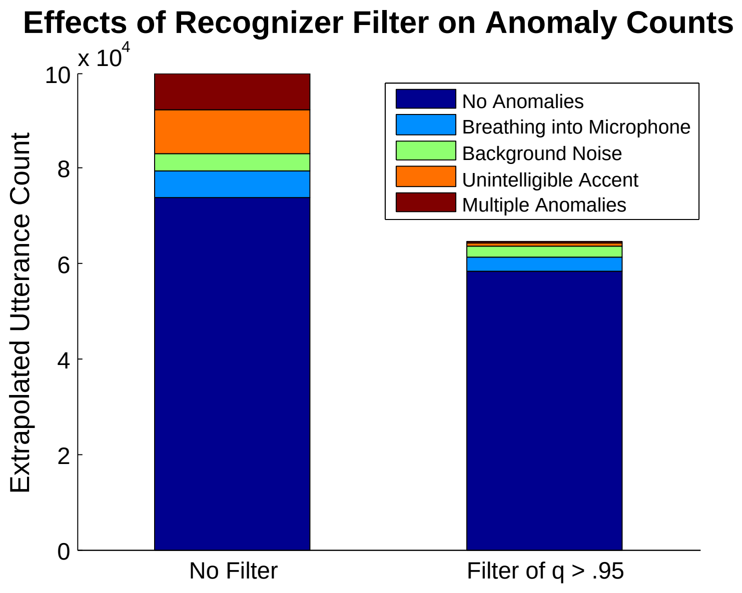


Figure 3-5: Breakdown of anomalies present in the corpus as a whole and the sub-corpus where workers have a high quality estimate, $q >= .95$. The recognizer-filtered sub-corpus still retains 65% of the original speech data. Though not explicitly shown here, we found that non-native speakers were still able to contribute to this sub-corpus: 5% of the filtered data with no anomalies came from non-native speakers.

While we do not experiment with using this data for retraining a recognizer in this chapter, one might be concerned that applying a recognition-based filter would have adverse effects when using this data to adapt an acoustic model. After all, if the recognizer already provides a correct partial transcript, retraining on this utterance might be unnecessarily reinforcing its existing models in certain portions of the acoustic space. In Chapter 6, we provide some experiments in a different domain that shows this is not necessarily the case.

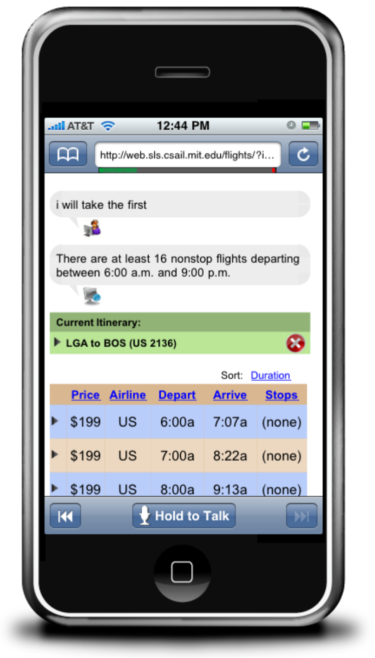
3.2 Multimodal Dialogue Interactions

Our experience with the address corpus inspired us to deploy a fully functional multimodal spoken dialogue system to mTurk. The Spoken Language Systems group has long been interested in *multimodal* interfaces in a variety of domains [127, 153]. Since the very same toolkit used in the prompted speech experiments acts as the web front-end to most of our spoken dialogue systems, it is relatively straightforward to collect a corpus for our *Flight-Browser* dialogue system. WAMI embeds an audio recorder into a web page to stream speech to MIT's servers for processing. Recognition results are then sent either to another server-side module for processing, or straight back to the client via Asynchronous Javascript and XML (AJAX).

In this section, we explore a range of price points for web tasks deployed to mTurk that ask the worker to book a flight according to a given scenario. The scenarios themselves were also generated by mTurk workers. Finally, once the data had been collected, we manually posted it back on mTurk for transcription and use the transcripts to evaluate the word error rate of the system. Had we automated these last few steps, we might have called this a human computation algorithm for spoken language system assessment. As is, we refer to this as crowdsourced spoken language system evaluation.

3.2.1 System Design

Flight Browser was derived from the telephone-based Mercury system, originally developed under the DARPA Communicator project [127]. Mercury's design was based on a *mixed-initiative* model for dialogue interaction. When flights are found, the system describes verbally the set of database tuples returned in a conversational manner. It prompts for relevant missing information at each point in the dialogue, but there are no constraints on what the user can say next. Thus, the full space of the natural language understanding system is available at all times.



i will take the first

There are at least 16 nonstop flights departing between 6:00 a.m. and 9:00 p.m.

Current Itinerary:

LGA to BOS (US 2136)

Sort: Duration

	Price	Airline	Depart	Arrive	Stops
▸	$199	US	6:00a	7:07a	(none)
▸	$199	US	7:00a	8:22a	(none)
▸	$199	US	8:00a	9:13a	(none)

Figure 3-6: *Flight Browser* interface loaded in the iPhone's WAMI browser. The same interface can be accessed from a desktop using any modern web browser. *Flight Browser* responds to spoken queries with a list of flights, the "drop-down" details of which can be accessed by clicking on the triangles to the right.

Using the WAMI Toolkit, we adapted Mercury to a multimodal web-interface we call *Flight Browser*. The dialogue interaction was modified, mainly by reducing the system's output verbosity, to reflect the newly available visual itinerary and flight list display. A live database of flights is used for the system. About 600 major cities are supported worldwide, with a bias towards U.S. cities. The interface was designed to fit the size constraints of a mobile phone, and multimodal support, such as clicking to sort or book flights, was added. Figure 3-6 shows *Flight Browser* in a WAMI app built specifically for the iPhone.

3.2.2 Scenario Creation

When designing a user study, many spoken dialogue system researchers struggle with the question of how to elicit interesting data from users without biasing the language that they use to produce it. Some have tried to present scenarios in tabular form, while others prefer to introduce extra language, hoping that the user will only pick up on the important details of a scenario rather than the language in which it is framed. Continuing the theme of crowd-sourcing research tasks, we take an alternative approach.

To generate scenarios, we created a mTurk task which asked workers what they would expect from a flight reservation system. They were explicitly told that we were trying

1. You need to find the cheapest flight from Maryland to Tampa, Florida. Find a cheap flight out of your choice of Philadelphia, Dulles or Baltimore airports.
2. You won a prize to visit Disneyland but have to provide your own airfare. You are going the week of Valentine's Day and you need 2 tickets from Seattle. You only have $500 to spend on tickets.
3. Destination: London, England Departs: January 15th, 2010 anytime in the morning Returns: March 1st, 2010 anytime after 3:00pm Price: Anything under $1500 To make things interesting I want as many layovers as possible!
4. You would like to take a vacation in Puerto Rico for two weeks. The departure and arrival dates must be on a Saturday.
5. You are a cartoonish crime boss in New York City, but Batman has caught on to you and you need to skip town for a while. You decide to head for Memphis, a city not normally known for costumed villainy. Set up a one-way flight with no layovers first thing tomorrow morning; cost is no object.

Table 3.1: Example scenarios collected for *Flight Browser* data collection. Quality and creativity ranged greatly. Overall, however, we found this to be a useful way of avoiding integrating our own biases into the data collection tasks.

to build a conversational system that could handle certain queries about flights, and we provided them with a few example scenarios that our system can handle. Their job was then to construct a set of new scenarios, each starting with the word "You..." and continuing to describe "your" desired itinerary. We paid $0.03 per scenario, and within a few hours 72 distinct workers had given us 273 scenarios, examples of which are shown in Table 3.2.2.

Not all of these 273 scenarios were suitable for a user study. Some workers did not fully follow the directions. Other crowd-sourced scenarios had dates that were in the past by the time we deployed our system. For the most part, however, the scenarios generated were far more creative and varied than anything we could have come up with ourselves in such a short amount of time. Although it was clear that some tasks would cause our system trouble, we did not explicitly exclude such scenarios from our study. For example, our system does not have *Disneyland* in its vocabulary, let alone a mapping from the landmark to the nearest airport. Ultimately, we chose 100 scenarios to form the basis of the data collection procedure described in the next section.

We view the scenario collection procedure described above as a step towards constructing user studies that are relatively unbiased with respect to system language and capabilities. One could envision formalizing a framework for soliciting relevant scenarios for evaluating spoken dialogue systems from non-expert workers.

3.2.3 Data Collection

Although the interface shown in Figure 3-6 is optimized for a mobile device, the WAMI Toolkit allows us to access it from modern desktop browsers as well. The following paragraphs describe how we were able to collect over 1,000 dialogue sessions averaging less than $0.20 apiece in under 10 days of deployment on Amazon Mechanical Turk.

HIT Design

The design of a HIT is of paramount importance with respect to the quality of the data we collected using Amazon Mechanical Turk. Novice workers, unused to interacting with a spoken language interface, present a challenge to system development in general, and the mTurk-workers are no exception. Fortunately, mTurk can be used as an opportunity to iteratively improve the interface, using worker interactions to guide design decisions.

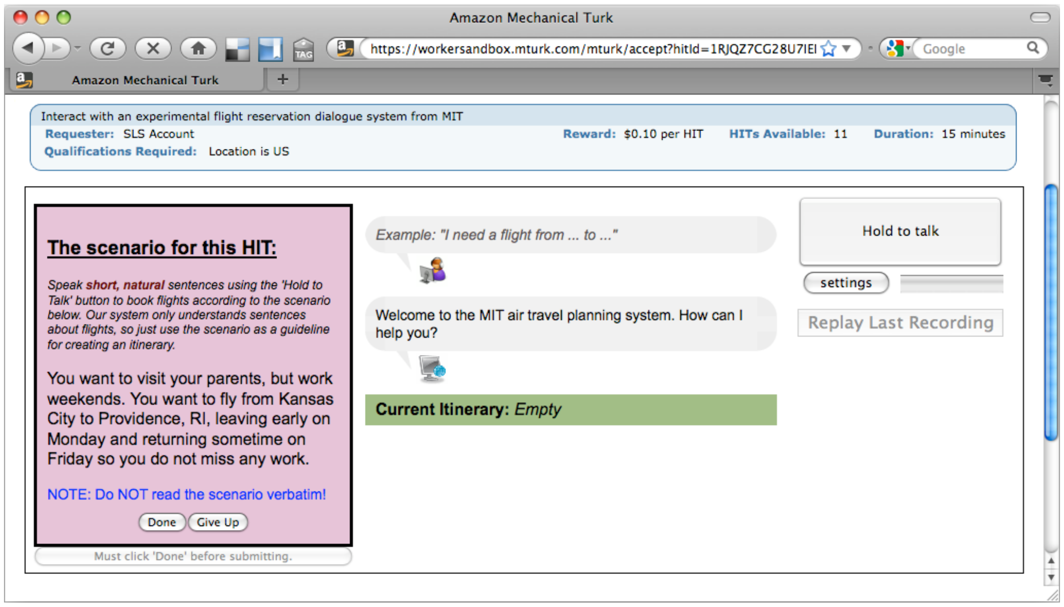
To optimize the design of our system and the HIT, we deployed short-lived mTurk tasks and followed them up with improvements based on the interactions collected. Since the entire interaction is logged on our servers, we also have the ability to *replay* each session from start to finish, and can watch and listen to the sequence of dialogue turns taking place in a browser. By replaying sessions from an early version of our interface, we discovered that many workers were not aware that they could click on a flight to view the details. This inspired the addition of the arrows on the left hand side, to indicate the potential for drop-down details.

Although initially we had hoped to minimize the instructions on screen, we found that, without guidance, a number of mTurk-workers just read the scenario aloud. Even after providing them with a short example of something they could say, a few workers were still confused, so we added an explicit note instructing them to avoid repeating the scenario verbatim. After a few iterations of redeploying and retuning the dialogue and scenario user interfaces, we eventually converged on the HIT design shown in Figure 3-7.

In order to complete a HIT successfully, a worker was required to book at least one flight (although we did not check that it matched the scenario); otherwise they were asked to "give up." Whether the task was completed or not, the worker had the option of providing written feedback about their experience on each scenario before submitting.

Extended Deployment

With the design stage complete, we decided to leave our HIT on mTurk for an extended period of time to collect a large amount of data. Beginning on a Tuesday, we deployed *Flight Browser* to mTurk and paid workers $0.20 for each scenario. We restricted the



Interact with an experimental flight reservation dialogue system from MIT
Requester: SLS Account **Reward:** $0.10 per HIT **HITs Available:** 11 **Duration:** 15 minutes
Qualifications Required: Location is US

The scenario for this HIT:

Speak ***short, natural*** *sentences using the 'Hold to Talk' button to book flights according to the scenario below. Our system only understands sentences about flights, so just use the scenario as a guideline for creating an itinerary.*

You want to visit your parents, but work weekends. You want to fly from Kansas City to Providence, RI, leaving early on Monday and returning sometime on Friday so you do not miss any work.

NOTE: Do NOT read the scenario verbatim!

Must click 'Done' before submitting.

Example: "I need a flight from ... to ..."

Welcome to the MIT air travel planning system. How can I help you?

Current Itinerary: *Empty*

Figure 3-7: A screenshot of the *Flight Browser* interface loaded into the mTurk HIT as it would appear to a worker. A scenario on the left indicates the nature of the worker's task. The itinerary is constructed in the center as spoken queries are made with the "Hold to talk" button on the right.

deployment to workers who had Amazon accounts in the United States. Each worker was limited to submitting sessions corresponding to the 100 scenarios described previously. In just under four days from our initial deployment, we had collected 876 dialogues from 63 distinct users, totaling 9,372 audio files. An example dialogue collected is shown in Figure 3-8.

Curious about how price affected the rate of collection, we deployed the same task for $0.10 around a month later. This task was started on a Thursday and left running for 6 days. Though clearly there was less interest in the HIT, we were still able to collect 2,595 utterances over 237 dialogues from 43 distinct workers. It should be noted that we made no special effort to exclude workers from the earlier task from participating in the $0.10 HIT a month later.

Figure 3-9 shows histograms for each price point of sessions collected from individual workers, as well as the number of tasks they marked "finished" and "give up". As shown in the plots, five workers participated in the $0.10 task despite being paid twice as much the month before. It is interesting to note that they were still willing to participate despite earning half as much. In fact, three of the top four contributors to the second round of data collection were repeat visitors. This is consistent with our general observation that many workers from the U.S. do these tasks for fun, and that the small sums of money involved

S:	[...] How can I help you?
U:	i need a flight from kansas city to providence rhode island rhode island
S:	[...] What date will you be traveling?
U:	monday morning
S:	[...] One moment please.
S:	[...] There are at least 27 flights departing between 5:30 a.m. and 12:39 a.m.
U:	i will take the five thirty
S:	[...] What date will you be returning on?
U:	[noise] friday afternoon
S:	[...] One moment please.
S:	[...] There are at least 20 flights departing between 12:23 p.m. and 6:45 p.m.
U:	i will take the twelve twenty three
S:	Can you provide a departure or arrival time? There are at least 6 Delta flights from Providence to Kansas City on Friday February fifth.
U:	arrive at five oh four
S:	[...] How about a flight arriving at 5:04 p.m. connecting in Detroit? Would that work? [...]
U:	yes
S:	Delta flight 6695 connecting to Delta flight 7377 in Detroit has been added to your itinerary. [...] Is there something else I can do for you?
U:	no
S:	Have a nice day.

Figure 3-8: A sample mTurk-collected interaction. While the recognition results are given in full, parts of the system responses have been omitted for brevity. Also not shown are the updates to the visual display which show the user query results.

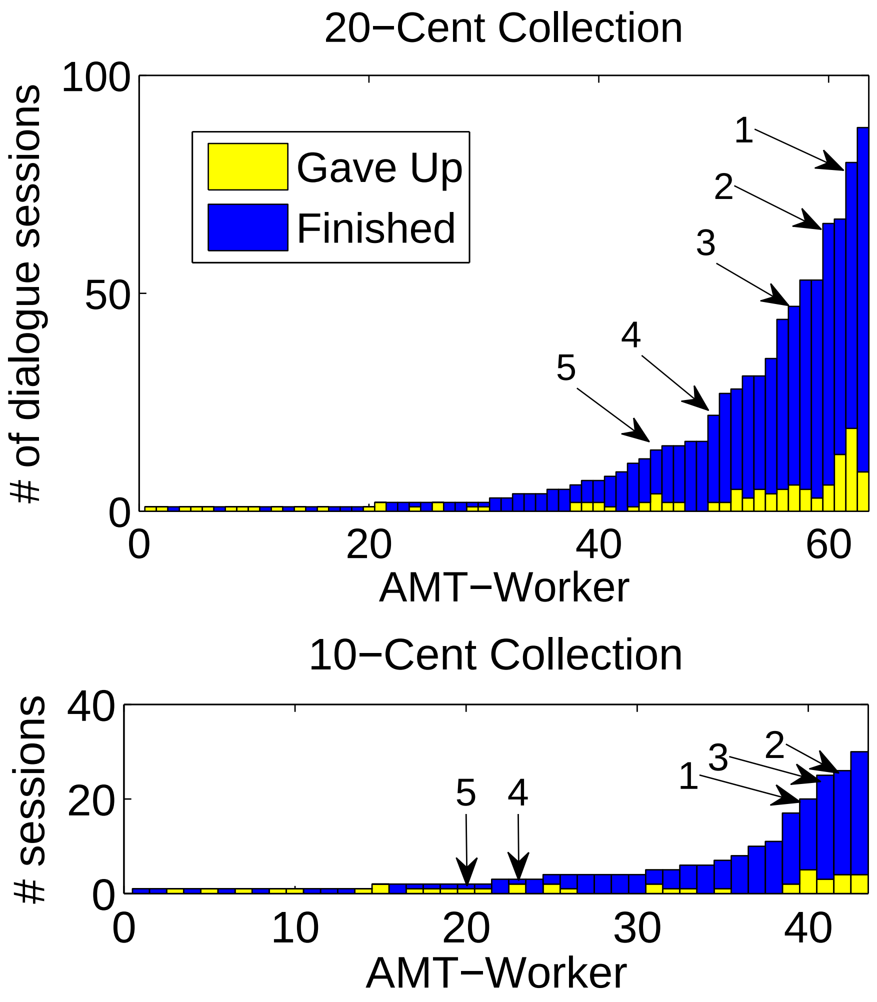


Figure 3-9: A breakdown of the data collection efforts by worker. For each price point, the workers are sorted in ascending order of the number of dialogues they contributed to the corpus. Numbers 1-5 identify the five workers who participated in both data collection efforts.

are viewed as an added bonus.

In both deployments a non-trivial number of audio files were recognized as noise or silence. This phenomenon has been observed previously when utterances come from more realistic sources [1]. Listening to these in context, it became apparent that some users required time to familiarize themselves with the recording software. We decided to ignore the 1,316 files associated with empty recognition results, leaving 10,651 utterances for analysis. Table 3.2 summarizes statistics from both deployments.

	\$0.20 HIT	\$0.10 HIT
# Sessions	876	237
# Distinct Workers	63	43
# Utterances	8,232	2,419
Avg. # Utts. / Session	9.5	10.2
% Sessions Gave Up	14.7	17.3

Table 3.2: Corpus Statistics for \$0.10 and \$0.20 mTurk HITs. For the former, this works out to 2.1¢ per utterance; for the latter, it is less than 1¢ per utterance. Note that when workers are paid more, they typically have a higher tolerance for difficult sessions, as indicated by the % of sessions given up at each price point.

3.2.4 Data Transcription

To transcribe our newly collected data, we once again turn to the Amazon Mechanical Turk cloud service. Previous work has explored the use of mTurk for transcription to generate highly accurate orthographies. We explore this area further, and show how seeding the transcription interface with recognizer hypotheses enables an automatic detection method for "bad" transcripts.

Figure 3-10 depicts a flowchart of our transcription procedure. We deployed our entire corpus to mTurk in a \$0.05 HIT, which asked workers to listen to utterances and correct recognizer hypotheses. Each HIT contains a bundle of 10 utterances for transcription. Once a set of candidate transcripts is complete, we automatically filter transcripts that are likely to be erroneous before moving on to a voting stage where transcripts are combined given the candidates they have accumulated so far. The process was iterated until 99.6% of our data were accepted by our voting scheme.

We use two filters to remove poor transcript candidates from the pool before voting. First, since the average number of words per HIT is around 45, the likelihood that *none* of them need to be corrected is relatively low. This allows us to detect lazy workers by comparing the submitted transcripts with the original hypotheses. We found that 76% of our non-expert transcribers edited at least one word in over 90% of their hits. We assumed that the remaining workers were producing unreliable transcripts, and therefore discarded their transcripts from further consideration. Second, we assume that a transcript *needs* to be edited if more than two workers have made changes. In this case, we filter out transcripts which match the hypothesis, even if they came from otherwise diligent workers.

The question of how to obtain accurate transcripts from non-expert workers has been addressed by [99], who employ the ROVER voting scheme to combine transcripts. Indeed, a number of transcript combination techniques could be explored. In this work, we take a simple majority vote on *good* transcripts, which have passed the filtering stage. Before

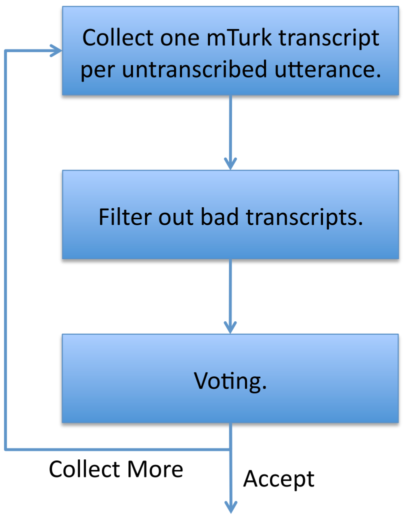

Collect one mTurk transcript per untranscribed utterance.
Filter out bad transcripts.
Voting.
Collect More
Accept

Figure 3-10: Flowchart detailing the transcription procedure. For a given utterance, one transcript is collected at a time. Based on the editing habits of each worker, bad transcripts are filtered out. Provided enough good transcripts remain, punctuation and capitalization is removed and voting then proceeds. If there is not yet a majority vote, another transcript is collected. If five good transcripts are collected without a majority vote, we begin to accept a plurality.

# Good Transcripts Required (G) % Corpus Transcribed (T)					
G	2	3	5	6	7+
T	84.4	95.2	96.3	98.4	99.6

Table 3.3: *Flight Browser* transcription statistics. Shown here is the % that we were able to transcribe with a simple majority voting procedure given a certain number of "good" transcripts.

voting, punctuation and capitalization are removed. If a majority vote cannot be found, another transcript is requested. If still no majority vote can be found among five good transcripts, we begin to accept a plurality vote. We found that 95.2% of our data only needed 3 good transcriptions to pass a simple majority vote. Table 3.3 indicates the amount of data we were able to transcribe for a given number of good transcripts.

The total cost of the transcription HIT was $207.62 or roughly 2¢ per utterance. Fifty three audio files did not have an accepted transcript even after collecting 15 transcripts. We listened to this audio and discovered anomalies such as foreign language speech, singing, or garbled noise that caused mTurk workers to start guessing at the transcription.

In order to assess the quality of our mTurk-transcribed utterances, we had two expert transcribers perform the same HIT for 1,000 utterances randomly selected from the corpus. We compared the orthographies of our two experts and found sentence-level exact agree-

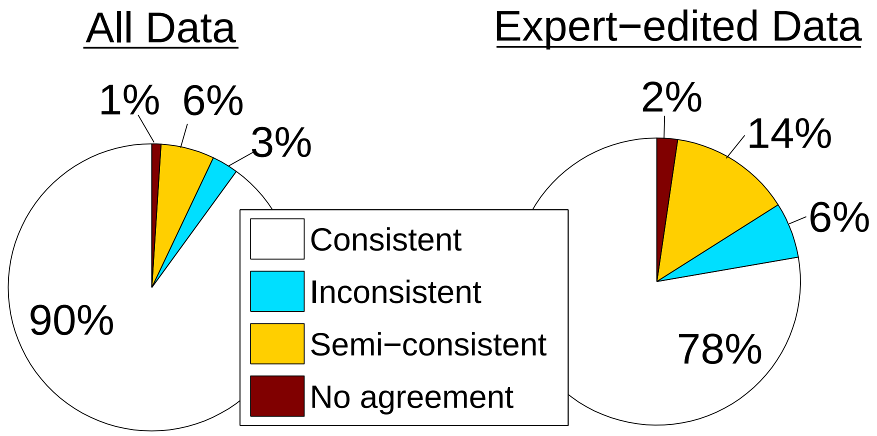


Figure 3-11: These charts indicate whether the mTurk transcripts were consistent, semi-consistent, or inconsistent with two expert transcribers. The semi-consistent case arises when the experts disagreed, and the mTurk transcript matched one of their transcripts.

ment to be 93.1%. The mTurk-transcripts had 93.2% agreement with the first expert and 93.1% agreement with the second, indicating that our mTurk-derived transcripts were of very high quality.

Figure 3-11 shows a detailed breakdown of agreement, depicting the consistency of the mTurk transcripts with those of our experts. For example, of all the data edited by at least one expert, only 6% of the mTurk-transcripts were inconsistent with an expert-agreed-upon transcript. Where the experts disagree, mTurk-labels often match one of the two, indicating that the inconsistencies in mTurk transcripts are often reasonable. For example, “I want a flight to” and “I want to fly to” was a common disagreement.

Lastly, we also asked workers to annotate each utterance with the speaker’s gender. Again, taking a simple vote allows us to determine that a majority of our corpus (69.6%) consists of male speech.

3.2.5 Data Analysis

Using the mTurk-transcribed utterances, we can deduce that the word error rate of the mTurk-collected speech was 18.1%. We note, however, that, due to the fact that the dialogue system task was not compulsory, this error rate may be artificially low, since workers who found *Flight Browser* frustrating were free to abandon the job. Figure 3-12 shows the WER for each worker plotted against the number of sessions they contributed. It’s clear that workers who experienced high error rates rarely contributed more than a few sessions,

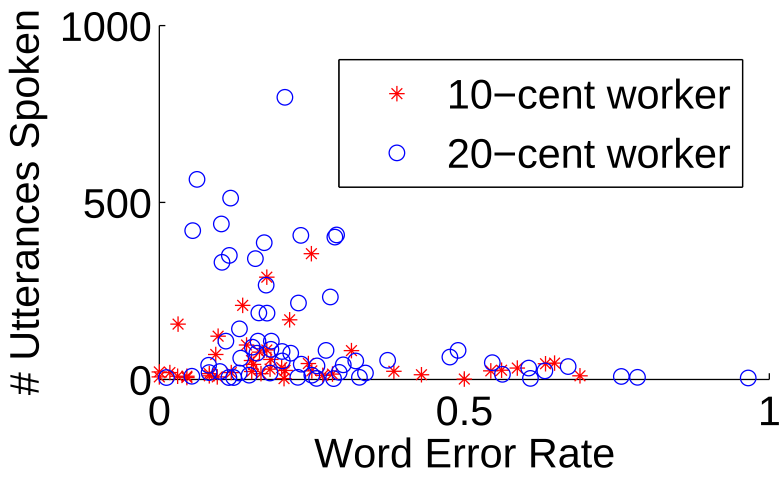


Figure 3-12: The number of *Flight Browser* sessions contributed by each worker is plotted against the WER experienced by that worker. Note that the 10¢ workers were less willing to persist in the face of recognition errors.

likely due to frustration. To provide a fairer estimate of system performance across users, we take the average WER over all the *speakers* in our corpus and revise our estimate of WER to 24.4%.

Figure 3-12 also highlights an interesting phenomenon with respect to system usability. It appears, that workers were willing to interact with the system so long as their WER was less than 30%, while workers who experienced higher WERs were not likely to contribute more than a few sessions. We imagine this threshold may also be a function of price, but do not explore the matter further here.

Upon replaying a number of sessions, we were quite happy with the types of interactions collected. Some dialogue sessions exposed weaknesses in our system that we intend to correct in future development. The workers were given the opportunity to provide feedback, and many gave us valuable comments, compliments and criticisms, a few of which are shown in Table 3.4.

To analyze the linguistic properties of the corpus quantitatively, we decided to compare the recognition hypotheses contained in the worker interactions with those of our internal database. From March of 2009 to March of 2010, *Flight Browser* has been under active development by 5 members of our lab. Every utterance spoken to *Flight Browser* during this time window has been logged in a database. User studies have been conducted in the laboratory and demos have been presented to interested parties. The largest segment of this audio, however, comes from developers, who speak to the system for development and debugging. In total, 9,023 utterances were recorded, and these comprise our internal

1. There was no real way to go back when it misunderstood the day I wanted to return. It should have a go back function or command.
2. Fine with cities but really needs to get dates down better.
3. The system just cannot understand me saying “Tulsa”.
4. Was very happy to be able to say two weeks later and not have to give a return date. System was not able to search for lowest fare during a two week window.
5. I think the HIT would be better if we had a more specific date to use instead of making them up. Thank you, your HITs are very interesting.

Table 3.4: Feedback on the *Flight Browser* mTurk HIT.

	Internal	mTurk-Collected
# Utts.	9,023	8,232
# Hyp Tokens	49,917	36,390
# Unique Words	740	758
# Unique Bigrams	4,157	4,171
# Unique Trigrams	6,870	7,165
Avg. Utt. Length	4.8	4.4

Table 3.5: Comparison between internal corpus and one collected from mTurk. Note that in light of the fact that the utterances in the mTurk collected set are both fewer in number and shorter, the comparable number of N-grams would suggest that this data is to some degree more complex.

database. We summarize a number of statistics common to the internal and the 20¢ mTurk-collected corpora in Table 3.5.

While the average recognizer hypothesis is longer and the number of words is greater in our internal data, the overall language in the cloud-collected corpus appears to be more complex. This is striking because our data collection was domain-limited to 100 scenarios, while the developers were unrestricted in what they could say. These results suggest that, because system experts know how to speak with the system, they can communicate in longer phrases; however, they do not formulate new queries as creatively or with as much variety as mTurk workers.

3.3 Summary

In this chapter, we have demonstrated the utility of the Amazon Mechanical Turk service in a number of speech data collection tasks. We have explored the practicality of deploying a simple read-aloud task to mTurk, and demonstrated that it is possible to collect large amounts of in-domain speech data very quickly and relatively cheaply. By extending this approach to spontaneous speech solicitation within a multimodal dialogue system, we have provided spoken language systems researchers with a method of collecting in-domain speech data.

Central to this work has been designing tasks for non-expert workers that are easily verifiable. We have shown how the recognizer can be used as a tool to loosely constrain both transcription and collection tasks, allowing us to filter out low quality data. When taken to the limit, much of the drudge work associated with spoken-dialogue system research can be easily outsourced to the crowd in the cloud. In the remainder of this thesis, we will use this technique to iterate between the tasks of scenario generation, data collection, transcription, and even retraining, to automatically improve system performance with minimal expert guidance.

Chapter 4

Crowd-supervised Language Modeling

Typically, data collection, transcription, language model generation, and deployment are separate phases of creating a spoken language interface. An unfortunate consequence of this is that the recognizer usually remains a static element of systems often deployed in dynamic environments. By providing an API for human intelligence, Amazon Mechanical Turk changes the way system developers can construct spoken language systems. In this chapter, we describe an architecture that automates and connects these four phases, effectively allowing the developer to *grow* a spoken language interface through human computation. In particular, we show that a *human-in-the-loop* programming paradigm, in which workers transcribe utterances behind the scenes, can alleviate the need for expert guidance in language model construction. We demonstrate the utility of these *organic* language models in a voice-search interface for photographs.

Recall from the previous chapter that spoken language interface development is subject to the classic chicken and egg problem: training data are needed to build a system; however, to collect in-domain training data, one first needs the system. To combat this reality for language models, researchers are forced to play a delicate game of seeding the speech recognizer using data collected from a text-only version of the system or generated programmatically using templates [19], perhaps with the aid of some out-of-domain data [30]. This chapter will show that crowdsourcing platforms such as mTurk are poised to fundamentally change the development process of such systems by either eliminating or automating these awkward initial steps of building spoken language interfaces.

To the casual user mTurk is a convenient mechanism for distributing tasks via the web to an anonymous crowd of non-expert workers. The work in the previous chapter was conducted using the command-line-interface, which makes it even easier to manage large numbers of assignments. The true power of Amazon Mechanical Turk, however, can only be harnessed by manipulating tasks programmatically through an API. Only then is it possible to construct and combine tasks on-the-fly, allowing developers to create true *human-in-the-loop* algorithms. In Chapter 2, we described how researchers have begun to use this API to create sophisticated workflows for a set of mTurk tasks to construct complex *human computation* processes, such as performing "impossible" database queries where humans answer otherwise incomputable sub-problems of the task [43].

In this chapter, we use a Java API to perform the iterative work necessary to produce transcripts of short utterances. As previously noted, TurKit is a toolkit well-suited to iterative tasks. For example, in [92], Little et al. describe using TurKit to iteratively improve upon the interpretation of sloppy hand-writing. In Chapter 5, we explore the use of TurKit to iteratively improve speech transcripts in a similar fashion; however, in this chapter we rely on a Java API for mTurk directly.

While we report on latency, we leave its optimization to future work. Complex tasks can be completed relatively quickly on mTurk. Soylent [12], for which Bernstein et. al. describe crowd-sourcing fairly involved word-processing tasks, was shown to have wait times of less than 20 minutes. Spoken language systems can be deployed to mTurk for days, and we show in this chapter that gains can still be analyzed even if the improvements to the models lag behind by an hour or two. Chapter 2 described more research regarding optimizing latency and the possibility of interacting with real-time crowds.

The architecture described in this chapter employs *human-in-the-loop* programming to facilitate spoken language interface creation using automatically collected and transcribed utterances. In particular, we focus on the task of automatically growing a language model on-the-fly using in-domain speech data without expert guidance. In this chapter, we consider the relatively unconstrained domain of photo annotation and search. We show that photo query recall improves dramatically over the lifetime of a deployed system which builds its vocabulary and language model from scratch by automatically coordinating a crowd of mTurk workers to provide training data. This small set of experiments demonstrates the feasibility of crowd-supervised language modeling.

4.1 WAMI's Javascript API

We noted in Chapter 2 that moving from speech annotation to elicitation is a daunting endeavor due to technological hurdles. Incorporating a speech component into a task is problematic due to the additional server-side complexity. Tools such as those described in [52] and [86], both of which have been used to collect read speech, alleviate some of the impediments to speech elicitation. Taking this to the next degree, we use the open source WAMI Toolkit to build fully interactive multimodal interfaces deployable to Ama-

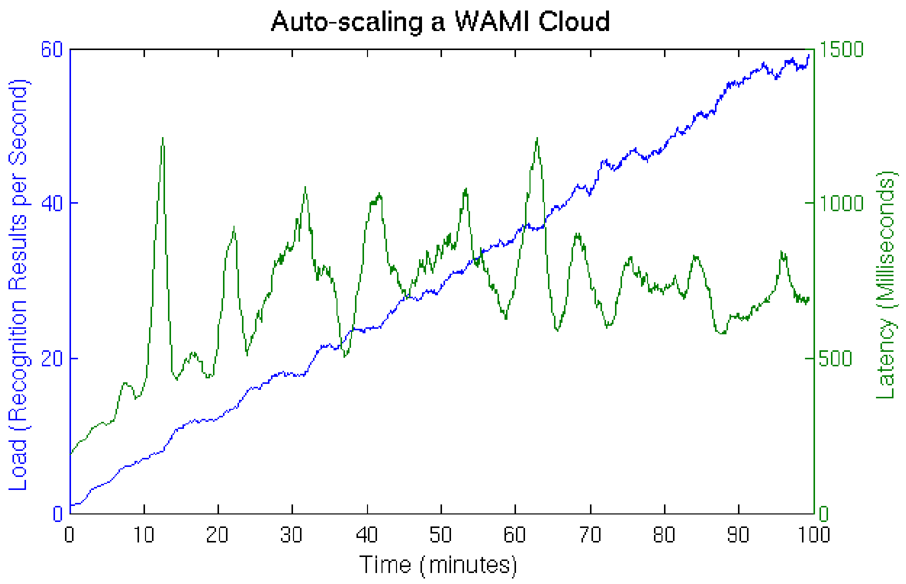


Figure 4-1: WAMI deployed to Amazon EC2. By replaying actual WAMI sessions at greater and greater frequency, we simulated a linearly increasing load. This plot highlights WAMI's auto-scaling capabilities. Despite the increasing load, latency is kept to a constant by starting new virtual instances on Amazon's cloud architecture. After 100 minutes, the load has reached 400 simultaneous simulated sessions. The average latency, e.g. the amount of time it takes to receive a recognition result, is still well under one second.

zon Mechanical Turk. In this section we discuss, in more detail, some of the capabilities of the WAMI Toolkit and explain how we have extended it to incorporate features which allow us to create and study organic spoken language systems.

WAMI is a complex bundle of server-side and client-side machinery based on an Apache Tomcat, PostgreSQL stack. WAMI's core functionality is to provide all the necessary plumbing to get audio from the client side, typically a web browser, to a recognizer running server-side. The recognition results must then find their way back to the client. We also add logging code to capture events and audio from user interactions. WAMI scales gracefully with an arbitrary number of users, and care was taken in each component to ensure efficiency and thread-safety.

We have even taken steps to stress-test WAMI under a significant load while it was deployed on an auto-scaling Amazon EC2 cloud. In Figure 4-1, we run a simulation of an increasing number of WAMI sessions over a period of 100 minutes. Load is added linearly over the duration of the experiment. We make use of Amazon's auto-scaling features to add new machines when the load on a particular machine reaches a certain threshold. At this stage, a new machine fires up and takes over some of the responsibility. Each peak in Figure 4-1 is indicative of a WAMI instance starting up and being added to the load

```
<script src="http://wami.csail.mit.edu/wami-2.0.js" />

<script>
var myWamiApp = new Wami.App( ... );

var grammar = {
    grammar : "...";
    language : "en-us";
    type : "jsgf";
};

myWamiApp.setGrammar(grammar);
</script>
```

Figure 4-2: The WAMI Javascript API. This sample code hints at the relative ease with which one can incorporate speech recognition into a web page using the WAMI API.

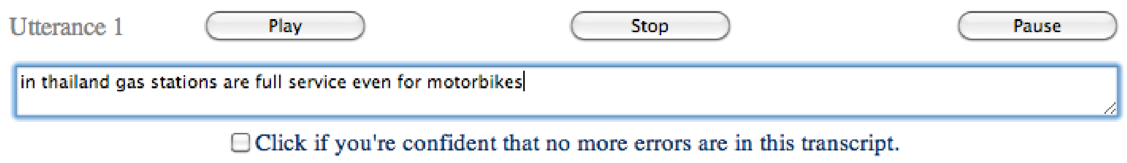
balancer. By the end of this experiment, 406 simultaneous users, simulated from previously logged sessions, were interacting with this WAMI cluster. The latency of the recognition results remained largely under one second.

Setting aside the configuration of a load-balanced WAMI cluster, implementing even the relatively basic features of WAMI requires a myriad of technologies, presenting a large barrier-to-entry for anyone wishing to incorporate speech features into a website. For this reason we have not only open-sourced the WAMI project, but we also host a third-party-accessible version of WAMI[1] which exposes the easy-to-use Javascript API shown in Figure 4-2. The Javascript API allows web developers with no knowledge of speech science to incorporate basic recognition into their sites with just a few lines of client-side code. In Chapter 6, we describe Quizlet.com, a flashcard website which has used WAMI to speech-enable two educational games.

At a minimum, a third party developer using WAMI needs to specify a language, a grammar, and the type of language model that the grammar represents. The publicly deployed API supports two main types of grammars: `jsgf` and `corpus`. The `jsgf` language model is specified using the Java Speech Grammar Format described in Section 2.3.2. One advantage of these grammars is that embedded semantic tags can be parsed and decoded from the recognizer results. Still, a feature commonly requested was the ability to loosen the language constraints of our recognition service.

A `corpus` argument is also an accepted `type` in the `grammar` object passed through to the recognizer. The text associated with the grammar is then assumed to be a set of sentences in the domain. These sentences are then compiled into a trigram language model on-the-fly, which is then used in the recognizer for that session. Since it is not advisable

[1]http://wami.csail.mit.edu



in thailand gas stations are full service even for motorbikes

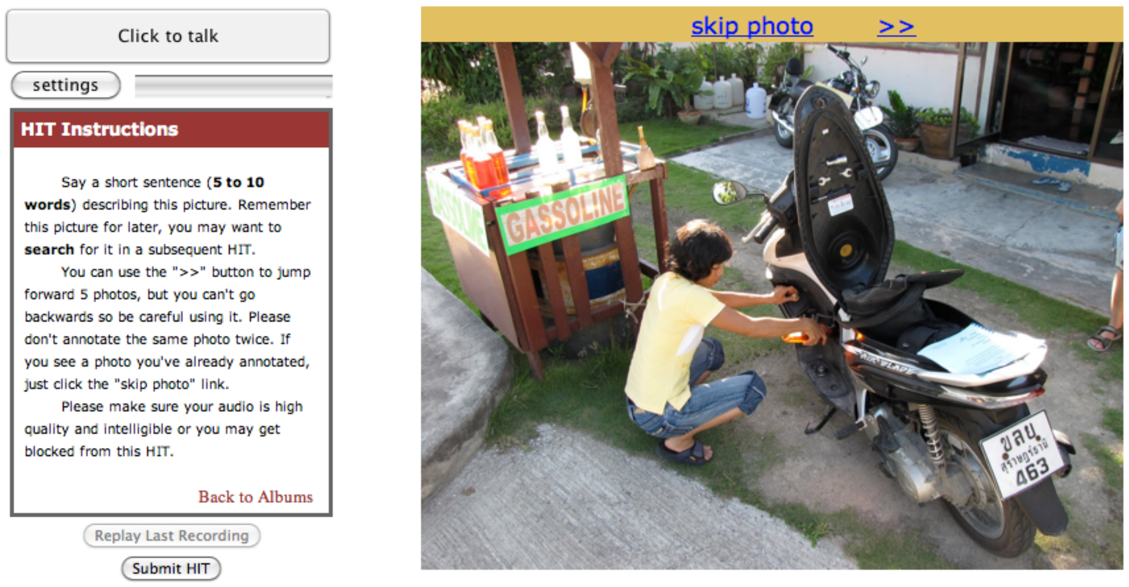
Click if you're confident that no more errors are in this transcript.

Figure 4-3: In the iterative transcription task above, batches of five utterances are combined into a 5¢ HIT. The first worker fills in a blank transcript which subsequent workers correct. A final worker approves the transcript if there are no more changes to be made.

to pass large amounts of data from the client, a `cached` type of grammar has also been implemented. In particular, we allow a third party developer to upload a corpus once, and provide them with an ID by which to reference it in their Javascript.

For our internal use, we have extend the trigram compilation framework to add another language model type, `organic` language models, which is *grown* on-the-fly. Such a language model can be defined from multiple transcription sources, which are abstracted away into a table in the logging database. Currently, a background thread in WAMI is configured to check for updates to each language model defined in the database every ten minutes. If new transcripts are found they are seamlessly sent to the recognizer for LM recompilation. One could envision an adaptive language model whose complexity (e.g., N-gram size) is altered based on the amount of data or even test set performance. In this chapter, however, attention is restricted to the simple case where a set of transcribed utterances is compiled into a continually updating trigram language model.

While the framework is agnostic to the way in which the transcripts find their way into the database, Amazon Mechanical Turk is an obvious choice. Figure 4-3 shows part of a transcription HIT which can be deployed to mTurk. Relying on a single worker is inadvisable when a certain level of accuracy is required of the transcripts. For this reason we have used mTurk's Java API to implement an iterative HIT. The first worker is given a blank area in which to type the transcript. A subsequent worker will then determine if there are any changes to be made. The final worker checks a box indicating that the transcript is error-free. This continues to a maximum of five workers per utterance. More often than not, this process stops early when a worker verifies a transcript without making edits. Although, this procedure is simpler and arguably more error-prone than the filter-based procedure proposed in Chapter 3, here we use the data for training rather than testing purposes. When used for training a speech recognizer, it has been shown that money is better spent collecting more transcripts rather than perfecting those already collected [110].



Say a short sentence (**5 to 10 words**) describing this picture. Remember this picture for later, you may want to **search** for it in a subsequent HIT.

You can use the ">>" button to jump forward 5 photos, but you can't go backwards so be careful using it. Please don't annotate the same photo twice. If you see a photo you've already annotated, just click the "skip photo" link.

Please make sure your audio is high quality and intelligible or you may get blocked from this HIT.

Figure 4-4: Photo annotation HIT. The instructions to the left detail the annotation portion of this HIT. After 10 photos annotated in with speech, however, a photo-search HIT would appear with slightly different instructions. The search HIT required the user to search through the previous 10 photos using voice queries.

4.2 Photo Search: A Case Study

We decided on photo-annotation and search as the domain in which to perform our preliminary experiments on our generic framework for organic language models. In addition to spoken annotation, we allow the user to draw with the mouse. Gesture collection, however, is not an area of focus in this thesis.

The photo user interface described in the following subsections was written entirely in Javascript and required no domain-specific server-side code. On the back-end, the SUMMIT landmark-based recognizer is configured with a large hand-crafted lexicon and acoustic models trained on telephone speech. For words not in the lexicon, a letter-to-sound module generates pronunciations on-the-fly.

4.2.1 Photo Annotation

We devised a web interface where an individual can log into Picasa, Flickr, or Facebook to access their personal photos. For those with privacy concerns, we also created a version that uses random public images from Flickr. In either case, voice annotations are streamed to a recognizer configured from the client side to use the organic language model.

Since our photo annotation system is new, it has no cultivated user base, but we can exploit mTurk to create a motivated user base through monetary award. To this end, we devised the photo-annotation HIT depicted in Figure 4-4, which encourages users to annotate photos for 10¢ a piece. This is relatively generous as mTurk tasks go, but our reasoning was that a certain privacy barrier must be broken when people talk about their own photos. A similar task where workers could annotate public photos was also deployed for 5¢.

An organic language model, initially containing only a single word, "*nothing*", was grown over the life-time of the HIT by transcribing the collected speech behind-the-scenes via a separate but simultaneous iterative transcription HIT. As the language model matures, the hypotheses eventually become useful for photo search.

4.2.2 Photo Search

To measure the utility of recognition results from photo annotations, we designed a voice search component to our photo user interface. Since positing a novel speech-based photo-retrieval algorithm is beyond the scope of this initial prototype, we took a simple approach to the search problem. Instead of using the same recognizer configuration, a special context free grammar for photo search was constructed and compiled on-the-fly using code similar to the sample found in Figure 4-2. The recognition hypotheses from a set of voice annotations are stored to generate a bag of words for each photo considered in the search: `<photo-i> = (word-1 | word-2 | ... | word-N)*` A few carrier phrases, such as `search for` and `find the`, were added to the grammar leading into these word-loops. Finally, semantic tags, e.g. `[photo=i]`, were embedded into each word-loop, to allow us to easily determine which photo answered the hypothesized query.

We inserted our search interface into the aforementioned annotation HIT in the following manner. After every set of ten photos, instructions were presented which asked the users to think back over the photos that they had just annotated and create a voice search. A user might, for instance, say *"search for the photograph of my motorbike in thailand"* in hopes that the computer would display the photo shown in Figure 4-4. Upon displaying the hypothesized image, the computer asks the user to confirm whether the correct photo was displayed. We required the user to make three search queries before continuing with the annotation of more photos. Thus, provided the worker does not abandon the task early, each set of ten photo annotations is accompanied by three search queries.

4.2.3 Experimental Results

The two versions of our photo annotation HIT, running with either public or personal photos, were each deployed to Amazon Mechanical Turk for two days. Using a few mTurk con-

Number of...	workers	utts.	searches
personal	27	995	105
public	35	1099	117

Table 4.1: Statistics for the two deployed photo annotation HITs. Note that according to the task design, three searches should be performed every 10 annotation HITs. Given that the ration is closer to one search every 10 annotations, it is clear that many of the workers left the task early.

figuration settings, we restricted the task to US workers and required them to have an acceptance rating of over 75% for their previous assignments. Still, since each user has their own microphone, speaking style, and subject matter, we decided to constrain the contributions of an individual worker over time, limiting each person to at most six searches in one hour. Once a worker went over his or her limit, a warning appeared telling the user to come back after a couple hours if they wanted to do more. Lastly, we encouraged users to keep utterances relatively short, and explicitly warned them if the recognizer detected over 25 words.

Once we had finished testing our user-interface, we deployed our final HITs and set the organic language model in motion. Table 4.1 displays some statistics regarding the two runs, each of which lasted 48 hours. It is interesting to see, for instance, that the additional 5¢ was enough to encourage workers to log into their personal photo accounts to complete the HIT. It is also clear that not everyone who starts working on the HIT makes it far enough to complete a search task.

The search error rate can be approximately determined by the workers' feedback on the correctness of the photo returned from the periodic search queries. While workers are not guaranteed to tell the truth, given that we restricted ourselves to the most trustworthy workers, and that they have no knowledge of our experimental interests, this assumption seems reasonable.

The average utterance length for the public photo annotation was 7.2 words, whereas for personal photos it was 6.6. The average delay between an utterance being logged in our system and a transcription being assigned to it via Amazon Mechanical Turk was 87 minutes. The high latency is due to a number of factors. Transcription HITs are bundled into groups of five, and so the first necessarily awaits the fifth before deployment. Furthermore, the background processes poll at fixed ten-minute intervals. This, along with the iterative nature of the HIT, requires a single utterance to go through a series of steps before being fully processed. Fortunately, the delay is still short enough to support studying system dynamics.

To determine the performance effects of the language model growth, the search error

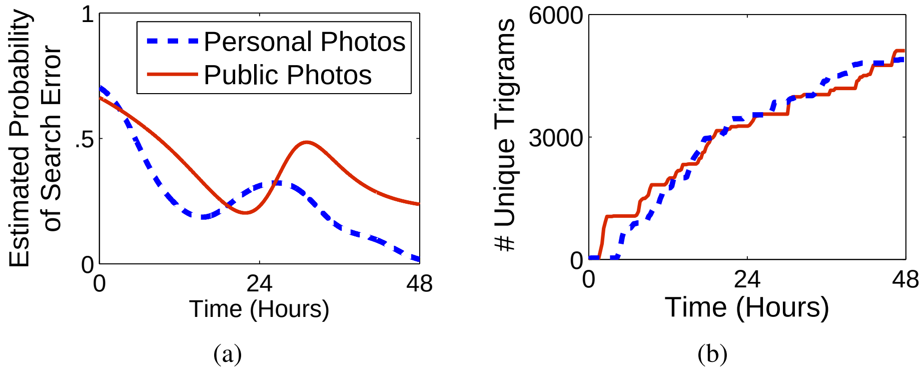


Figure 4-5: On the left, the probability of a search error occurring as a function of time is plotted for the two photo annotation tasks. This plot was created by taking the worker feedback over time and performing by Parzen-Rosenblatt density estimation. On the right, the number of unique trigrams in the organic language model is shown as it grows.

indicated by the worker is treated as a binary response variable dependent on time. Logistic regression, [56], was used to determine that the slope of the estimated probability of a search error over time is negative for both the public and personal photo tasks, with $p < .01$. We can also visualize the trend using kernel density estimation, as shown in Figure 4-5(a). The estimated probability of a search error as a function of time is given by Parzen-Rosenblatt density estimation. Figure 4-5(b) shows the growth of the language model in terms of the number of unique trigrams. Despite similar growth, the search error rate trends downward, and is lower for personal photos. We believe that the personal photos were easier to remember, and the search queries were better matched to the annotations. Both plots exhibit considerable fluctuation, which we believe is due to variation across different users.

It is clear from the plots that the organic language models are operating as expected, improving the system behind the scenes with no expert input. The difference was dramatic enough that one worker emailed us with the comment: *"I just wanted to say that I have noticed that your system seems to be getting better at recalling specific pictures."*

4.3 Summary

In this chapter, we have shown the feasibility of deploying an organic spoken language system to Amazon Mechanical Turk. We have explored growing trigram models using mTurk for a spoken language interface, and shown that improvements can be achieved without expert guidance. The next chapter, while focusing largely on the lexicon, will confirm the utility of this approach by describing an experiment in which a class-based

N-gram is retrained using turk-transcribed data.

The technology we have used in this chapter is, with the exception of the speech recognizer itself, open source. External open source recognizers, such as Sphinx form CMU [141], can be incorporated into WAMI with relative ease. Combining these tools, anyone can create a WAMI-enabled web page with just a few lines of JavaScript code. Those who wish to host their own web-based speech recognition APIs can download the latest version of the WAMI toolkit. We have also recently upgraded our audio recorder to use Flash. In some cases, merely capturing audio meets all the requirements of the task at hand. We therefore now host this project separately, as explained more fully in the appendices.

Chapter 5

Crowd-supervised Lexicon Learning

We now turn our attention to the lexicon. Often, it seems, the uncertainty modeled in the modern day speech recognizer is relegated to either the language model or, perhaps especially, to the acoustic model. Although in most recognizers it is straightforward to give pronunciations probabilities, lexicons are not usually stochastic. In this chapter, we will show that it might be time to reconsider this common-wisdom. We describe experiments on a *pronunciation mixture model* (PMM) that hint at the utility of incorporating pronunciation weights into the lexicon. Once we frame the lexicon probabilistically, we open it to the possibility of crowd-supervised training. We explore this route by first showing that we can achieve expert-level pronunciations by collecting inexpensive spoken examples from the crowd. We then wrap this new technique into the human computation paradigm to create another organic spoken language interface.

The system in our human computation experiments is called *Movie Browser*. The cinema voice-search domain that *Movie Browser* covers is replete with difficult names and words that are potentially out-of-vocabulary. Although the PMM, as formulated in the next section, requires both a spelling and spoken examples, both are easily obtained through crowdsourcing. With crowdsourced transcripts, we can also perform language model retraining. Tying these crowdsupervised training techniques together into a single system, this chapter nicely highlights many of the contributions made in this thesis so far.

5.1 The Pronunciation Mixture Model

In Section 2.3.3, we outline the state-of-the-art research on the topic of grapheme to phoneme conversion. In those examples, however, we did not cover work that makes use of supplementary acoustic examples. Since the goal of this section will be to introduce a framework that uses spoken examples, we discuss this research here [96, 29, 7, 91, 135].

In [7], a decision tree was used to find the pronunciation $\mathbf{b}^*$ given a word or grapheme sequence $\mathbf{w}$ and a spoken example $\mathbf{A}$. Another approach, described in [135], is to use a phonetic decoder to generate a list of possible pronunciations, then assign weights to each using a Minimum-Classification-Error criterion.

Arguably the work most similar to our own, however, is that of Li et al. [91]. They adapt graphone models using acoustic data and apply the learned pronunciations to a voice dialing task. Starting with a training set $(\mathbf{A}_i, \mathbf{w}_i)$ of acoustic examples $\mathbf{A}$ and their transcripts $\mathbf{w}$, this work maximizes the log-likelihood of their data:

$$\sum_{i=1}^{M} \log\ P(\mathbf{A}_i, \mathbf{w}; \theta) = \sum_{i=1}^{M} \log \sum_{\mathbf{b} \in \mathcal{B}} P(\mathbf{A}_i|\mathbf{w}) P(\mathbf{w}, \mathbf{b}; \theta)$$

Note that $\mathbf{b}$ represents both a single pronunciation and a sequence of phonetic units, $\mathbf{b} = b_1, b_2 \ldots b_{|\mathbf{b}|}$. In theory, the summation is over all possible pronunciations denoted by the set $\mathcal{B}$. The log-likelihood is maximized using Expectation Maximization (EM) to adjust θ, which, in this case, represents the graphone N-gram parameters. In addition to exploring the maximum likelihood approach, they experiment with discriminative training and show that it produces better results. Our work uses a more general framework which, while similar to their maximum likelihood approach, is not inherently graphone specific.

We motivate our pronunciation generation framework with an example. Suppose that we had two spoken utterances, $\mathbf{A}_1$ and $\mathbf{A}_2$, from which to learn a pronunciation of the word $\mathbf{w}_0$. One approach might be to find the single most likely pronunciation, $\mathbf{b}^*$, given both examples $\mathbf{A}_1$ and $\mathbf{A}_2$. This might be reasonable for many words, but suppose the word was "either", which has two common pronunciations. It probably does not make sense to have both utterances vying for a single "canonical" pronunciation, $\mathbf{b}^*$, if $\mathbf{A}_1$ pronounces the word *iy dh er* and $\mathbf{A}_2$ pronounces it *ay dh er*. Instead, in our model, both utterances are effectively allowed to distribute soft votes to a mixture of possible pronunciations.

In [5], we tackle the simplified task of learning word pronunciations from isolated-word speech. We describe some of the experiments from this work later in this section, with a particular focus on the results from a set of crowdsourced experiments. In [6], we extend the model to handle continuous speech. Since the isolated-word case is a simplification of the continuous speech model, in the remainder of this section, we provide the mathematical underpinnings for only the continuous case.

Learning pronunciations from continuous speech is a more difficult endeavor than the isolated word case due to coarticulation across word boundaries; however, the potential payoff is large since continuous speech is both easy to collect and fits well within the domains of many ASR tasks. The formulation itself is also somewhat more involved in the continuous case. Recall from Section 2.3.3 that the general ASR problem is a search for the most likely string of words $\mathbf{W}^* = \mathbf{w}_1^*, \cdots, \mathbf{w}_k^*$ given an utterance $\mathbf{A}$.

$$\mathbf{W}^* = \arg\max_{\mathbf{W}} P(\mathbf{W}|\mathbf{A}) = \arg\max_{\mathbf{W}} \sum_{\mathbf{B}\in\mathcal{B}_{\#}} P(\mathbf{W},\mathbf{B}|\mathbf{A})$$

Here we have explicitly modeled word boundaries. $\mathbf{B}$ is a sequence of word pronunciations: $\mathbf{B} = \mathbf{b}_1\#\mathbf{b}_2\#\cdots\#\mathbf{b}_k$ in the set of all possible word-delimited sequence $\mathcal{B}_{\#}$. We can decompose the equation above as follows:

$$\mathbf{W}^* = \arg\max_{\mathbf{W}} \sum_{\mathbf{B}\in\mathcal{B}_{\#}} P(\mathbf{A}|\mathbf{W},\mathbf{B})P(\mathbf{B}|\mathbf{W})P(\mathbf{W})$$

where $P(\mathbf{W})$ is the language model, $P(\mathbf{B}|\mathbf{W})$ can be computed using a stochastic lexicon and $P(\mathbf{A}|\mathbf{W},\mathbf{B})$ reduces to using the acoustic model to compute $P(\mathbf{A}|\mathbf{B})$. Since the speech recognizer employs the standard Viterbi approximations during decoding, our fundamental equation of speech recognition becomes:

$$\mathbf{W}^* = \arg\max_{\mathbf{W},\mathbf{B}} P(\mathbf{A}|\mathbf{B})P(\mathbf{B}|\mathbf{W})P(\mathbf{W}) \tag{5.1}$$

The goal now is to learn the appropriate weights for $P(\mathbf{B}|\mathbf{W})$. Our training data is comprised of M utterances and their transcripts $\{\mathbf{A}_i, \mathbf{W}_i\}$ where $\mathbf{W}_i = \mathbf{w}_1^i, \cdots, \mathbf{w}_{k_i}^i$ but the word boundary locations are unknown. We parameterize the log-likelihood as follows:

$$\sum_{i=1}^{M} \log P(\mathbf{A}_i, \mathbf{W}_i; \theta) = \sum_{i=1}^{M} \log \sum_{\mathbf{B}\in\mathcal{B}_{\#}} P(\mathbf{A}_i, \mathbf{B}, \mathbf{W}_i; \theta)$$

We now make an independence assumption that is typical of most lexicons. In particular, we assume that the words are modeled independently. While it is clear that pronunciations of adjacent words in continuous speech affect one another, a properly trained acoustic model will often recover from these types of coarticulation phenomena:

$$P(\mathbf{A}_i, \mathbf{B}, \mathbf{W}_i; \theta) = P(\mathbf{A}_i|\mathbf{B}) \prod_{j=1}^{k_i} P(\mathbf{w}_j^i, \mathbf{b}_j; \theta)$$

We parameterize the log-likelihood of our data with $\theta_{\mathbf{w}_j^i,\mathbf{b}_j} = P(\mathbf{w}_j^i, \mathbf{b}_j; \theta)$:

$$\sum_{i=1}^{M} \log P(\mathbf{A}_i, \mathbf{W}_i; \theta) = \sum_{i=1}^{M} \log \sum_{\mathbf{B}\in\mathcal{B}_{\#}} P(\mathbf{A}_i|\mathbf{B}) \prod_{j=1}^{k_i} \theta_{\mathbf{w}_j^i,\mathbf{b}_j} \tag{5.2}$$

The parameters, θ, are initialized to our graphoneme N-gram model scores. Note that this is the only graphone-specific step in the PMM framework. This initialization could easily be replaced by an alternative framework for defining distributions over graphemes and phonemes. Once the parameters are initialized, multiple EM iterations can be run.

E-step:

$$\overline{\mathrm{M}}_\theta[\mathbf{w}, \mathbf{p}] = \sum_{i=1}^{M} \sum_{\mathbf{B} \in \mathcal{B}_\#} P(\mathbf{B}|\mathbf{A}_i, \mathbf{W}_i; \theta) \mathrm{M}[\mathbf{p}, \mathbf{w}, \mathbf{W}_i, \mathbf{B}]$$

M-step:

$$\theta^*_{\mathbf{w},\mathbf{p}} = \frac{\overline{\mathrm{M}}_\theta[\mathbf{w}, \mathbf{p}]}{\sum_{\mathbf{w}',\mathbf{p}' \in V \times \mathcal{B}} \overline{\mathrm{M}}_\theta[\mathbf{w}', \mathbf{p}']}$$

where $\mathrm{M}[\mathbf{p}, \mathbf{w}, \mathbf{W}_i, \mathbf{B}]$ is the number of times word $\mathbf{w}$ appears in $\mathbf{W}_i$ aligned with the pronunciation $\mathbf{p}$. That is, $\mathrm{M}[\mathbf{p}, \mathbf{w}, \mathbf{W}_i, \mathbf{B}] = |\{j : \mathbf{b}_j = \mathbf{p} \text{ and } \mathbf{w}^i_j = \mathbf{w}\}|$.

In practice, we generate FSTs for each word by restricting the graphone language model to a particular word's grapheme sequence. Concatenating these FSTs together with word-boundary delimiters according to the sequence of words in a particular transcript yields our initial representation of $P(\mathbf{B}|\mathbf{W})$ in Equation 5.1. It is here that we implicitly make the restriction that a sequence of pronunciations $\mathbf{B}$ must have the same number of word boundaries as there are in $\mathbf{W}$ to have a non-zero probability. In subsequent iterations, the term $P(\mathbf{B}|\mathbf{W}; \theta)$ can then be computed as:

$$P(\mathbf{B}|\mathbf{W}; \theta) = \prod_{j=1}^{k} \frac{\theta_{\mathbf{w}_j,\mathbf{b}_j}}{\sum_{\mathbf{p} \in \mathcal{B}} \theta_{\mathbf{w}_j,\mathbf{p}}} \tag{5.3}$$

Notice that each term in the product above is a normalized weight for a pronunciation. In this way, the weights learned by the PMM can be used directly in a stochastic lexicon for decoding. We now describe some of our initial experiments using this framework.

5.2 *Phonebook* Experiments

The continuous speech PMM above can be applied to isolated word speech as well. In this section, we experiment with the NYNEX PhoneBook corpus [117], described in section 2.2.1. The corpus contains phonetically-rich, isolated-word speech, collected entirely over the telephone. A database consisting of 93,667 utterances, totaling 23 hours of speech, is comprised of 1,358 individual native English speakers saying up to 75 words each. Each word was spoken on average by about 12 speakers, making this an ideal corpus for experimenting with our pronunciation mixture model.

For our baseline experiments, we selected utterances from our corpus that corresponded to 2,000 words chosen randomly from the set of words that were spoken by at least 13 distinct speakers. For each word, we held out two utterances, one from a male speaker and the other from a female speaker, to construct a 4,000 utterance test set. This left 22,000 utterances from which to learn pronunciations using the PMM.

We use SUMMIT to experiment with the pronunciation mixture model. The observation space is computed using MFCC averages over varying durations around the hypothesized landmarks described in Section 2.3.4. In particular, a 14-dimensional MFCC-based feature set is computed for eight telescoping windows around a landmark and stacked to form the 112-dimensional feature vector. Principal components analysis then whitens the feature space and reduces the dimensions to the first 50 principal components. The acoustic models are diagonal Gaussian mixtures with up to 75 components trained on telephone speech.

The expert lexicon, which we use for both our baseline experiments and to train the graphone model, is based on the publicly available *PronLex* [76] dictionary. It has been expanded to contain around 150,000 lexical entries, including most of the words in the PhoneBook corpus. We therefore simulate an out-of-vocabulary scenario by removing the 2,000 experimental words from this lexicon. We then use an edit distance criterion to prune similarly spelled words. Following the work of [143], we train a 5-gram graphone language model.

We begin by describing two baseline experiments. First we build a lexicon for the 2000 test-words according to Equation 2.1. That is, we generate pronunciations using only the graphone L2S model and no acoustic information. We then use this lexicon to decode our 4,000 test utterances, and find a WER of 16.7%. Next, we use the expert-lexicon to decode the test-set, yielding a WER of 12.4%. These two experiments serve as our baselines against which we can evaluate the PMM.

The pronunciation mixture model experiments were conducted with the aid of a small development set of 1,500 previously discarded PhoneBook utterances. By generating lexicons after each EM iteration and testing on this development set, we determined that stopping after two iterations of EM was optimal. For evaluation, we created two PMM-based lexicons using varying amounts of data. The first lexicon is unweighted, and was created by choosing the top-weighted pronunciation for each word from a PMM-generated list and then discarding its weight. The second lexicon was stochastic and was created by incorpo-

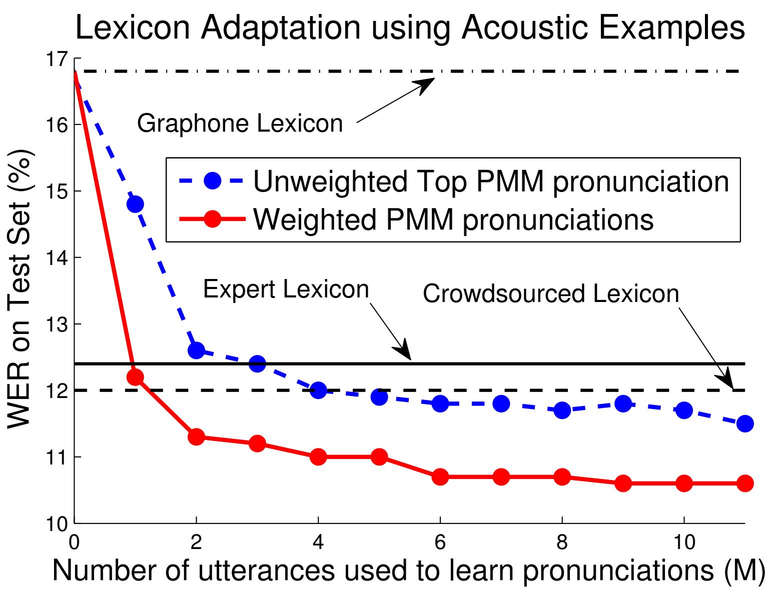


Figure 5-1: Word Error Rate (WER) on the PhoneBook corpus as a function of the number of example utterances used to adapt the underlying lexicon. We show results for using the PMM, the Graphoneme model and the Expert Dictionary

rating multiple pronunciations with their normalized PMM weights. Figure 5-1 shows the WERs for each of these types of lexicons. We have varied the number of utterances that each word is trained on to show how the PMM behaves with different amounts of data.

The first comparison we make places the pronunciation mixture model on a level playing field with the expert lexicon. The unweighted PMM, which contains a single pronunciation per word, is a close match to the expert lexicon, which averaged 1.08 entries per word and is also unweighted. Figure 5-1 shows that by the third utterance, the unweighted lexicon containing just the top PMM pronunciations is already competitive with the experts. As more utterances are added the performance improves even further.

A clear advantage of a *stochastic* lexicon, on the other hand, is the ability to learn pronunciation variation. Thus, rather than clinging to the notion of a canonical pronunciation for a given word, for our second comparison, we examine the stochastic PMM, which included the top 100 pronunciations along with their weights, learned according to Equation 5.3. Since most of the weight ends up concentrated toward the top five pronunciations, including a large number of possibly erroneous variants does not hurt performance. As seen in Figure 5-1, The stochastic lexicon achieves a WER of 10.7% by the 11th training utterance: an absolute improvement of 1.7% over the experts. McNemar's test shows this difference to be significant with $p = .016$.

The final experiment, of particular relevance to this chapter, regards the generation of pronunciations from crowdsourced speech. It is our belief that in many domains, and for many languages, acquiring acoustic data from non-experts will be cheaper than asking an expert to provide a pronunciation. In Chapter 3, we were able to collect over 100 hours of read speech, in less than four days. Here, we use the same technique to collect another 10 example utterances for each of the 2,000 words in our experiment at a cost of \$0.01 per utterance.

While using the PhoneBook data for training simulated ideal conditions wherein the training data matched the test data and the acoustic model used, we now show that *unmatched* speech collected cheaply from Amazon Mechanical Turk can be used to generate high-quality pronunciations in a fully automatic fashion. Since these distributed workers make use of their own equipment and record in whatever acoustic environment they happen to be in, there are many sources of variation in the data. It is not uncommon, when collecting audio from this service, to hear a television in the background or clipping from an ill-adjusted microphone.

Whereas in Chapter 3 we took care to filter the collected speech to obtain high-quality sub-corpora, we took no such precautions when collecting these example utterances. Thus, in addition to other sources of mismatch between the data and our acoustic model, these noisy data pose a challenge to even a recognizer built on expert pronunciations. Running a recognizer configured with the expert lexicon over these 20,000 utterances yields a WER of 50.1%. Some of the responsibility for this high error rate lies with the mismatched acoustic models, since we are no longer working with telephone speech. Still, as shown in Chapter 3, these data are also likely to contain a high degree of noise. In fact, since we make no guarantees that the worker even read the prompt, the true error rate is unknown.

It might seem, then, that using these data to generate pronunciations is a waste of effort. Recall, however, that the PMM is effectively a soft-voting scheme. As such, a single noisy utterance might not have adverse effects if its contribution can be outweighed by other examples. We perform a simple experiment to test this hypothesis. Using all 10 utterances for each word from the mTurk-collected data, we run our pronunciation mixture model to build a single crowdsourced stochastic lexicon. This lexicon, when used to decode the test set, exhibits a WER of 12.0%. As Figure 5-1 indicates, this crowd-sourced lexicon is on par with the experts.

In this section, we have shown how the PMM can be used to improve upon today's state-of-the-art L2S models using acoustic examples. We have also shown that the model is robust to noise. These results pave a trail for data-driven lexicon learning. In the next section, we integrate both the basic graphone L2S framework and the PMM technique into a running spoken language system. We show how the L2S can generate pronunciations for unknown words on-the-fly, and how the PMM can refine these pronunciations with help from the crowd. This, together with the language model learning approaches described in the previous chapter, combine into an organic spoken language system that grows and adapts multiple components while simultaneously servicing an active userbase.

5.3 Organic Lexicon Learning with *Movie Browser*

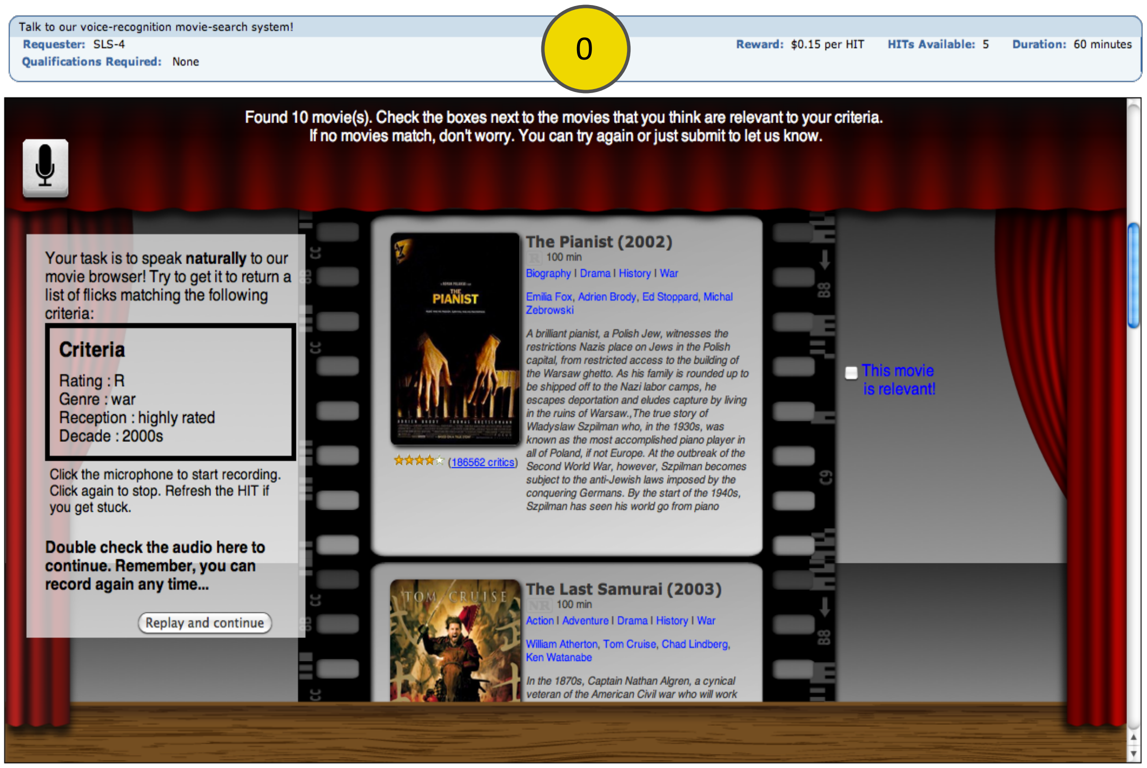
In this section, we explore the use of human computation to improve the lexicon of a spoken language system using the algorithms introduced above. We conduct our experiments in a cinema voice-search domain using the *Movie Browser* spoken language system. The system itself allows users to make speech queries to search the Internet Movie Database (IMDB). Our training framework, however, is domain independent. Unlike the work in the previous chapter, here, we enlist the help of TurKit [92], to link together transcription, semantic tagging, and speech collection tasks in a fully automatic fashion.

Using this assembly line of human intelligence tasks (HITs), we perform retraining operations, such as rebuilding the class-based language model that underpins the recognizer's search space. The main focus of this work, however, is on the lexicon. As new movies come out and new actors become popular, it is easy to imagine that the lexical domain of the *Movie Browser* might shift. Due to the inherent difficulty with the pronunciation (and, consequently, recognition) of named entities, we may wish to improve upon the state-of-the-art letter-to-sound models using a PMM.

In the remainder of this section, we describe the *Movie Browser* system, and detail the manner in which we update its lexicon and language model using a fully-automated crowd-supervised framework. We perform experiments and show improvements of a metric based on movie search result relevance. We also examine WER improvements, and in doing so, we confirm that the pronunciation mixture model is robust to the noisy contributions of distributed workers in this continuous speech setting.

5.3.1 The *Movie Browser* System

The *Movie Browser* system, shown in Figure 5-2, provides the test-bed for our training framework. The system is intended to handle natural spoken queries such as *"What are some James Bond movies starring Sean Connery?"* The *Movie Browser* makes use of conditional random fields to parse the output of the speech recognizer into the set of semantic categories used during searches. The search index, handled by the Apache Lucene project, is initialized to contain over 12,000 movie titles from the IMDB database. More details


Talk to our voice-recognition movie-search system!
Requester: SLS-4
Qualifications Required: None
Reward: $0.15 per HIT HITs Available: 5 Duration: 60 minutes

Found 10 movie(s). Check the boxes next to the movies that you think are relevant to your criteria.
If no movies match, don't worry. You can try again or just submit to let us know.

Your task is to speak **naturally** to our movie browser! Try to get it to return a list of flicks matching the following criteria:

Criteria
Rating : R
Genre : war
Reception : highly rated
Decade : 2000s

Click the microphone to start recording. Click again to stop. Refresh the HIT if you get stuck.

Double check the audio here to continue. Remember, you can record again any time...

The Pianist (2002)
R 100 min
Biography | Drama | History | War
Emilia Fox, Adrien Brody, Ed Stoppard, Michal Zebrowski
A brilliant pianist, a Polish Jew, witnesses the restrictions Nazis place on Jews in the Polish capital, from restricted access to the building of the Warsaw ghetto. As his family is rounded up to be shipped off to the Nazi labor camps, he escapes deportation and eludes capture by living in the ruins of Warsaw.,The true story of Wladyslaw Szpilman who, in the 1930s, was known as the most accomplished piano player in all of Poland, if not Europe. At the outbreak of the Second World War, however, Szpilman becomes subject to the anti-Jewish laws imposed by the conquering Germans. By the start of the 1940s, Szpilman has seen his world go from piano
★★★★☆ (186562 critics)

This movie is relevant!

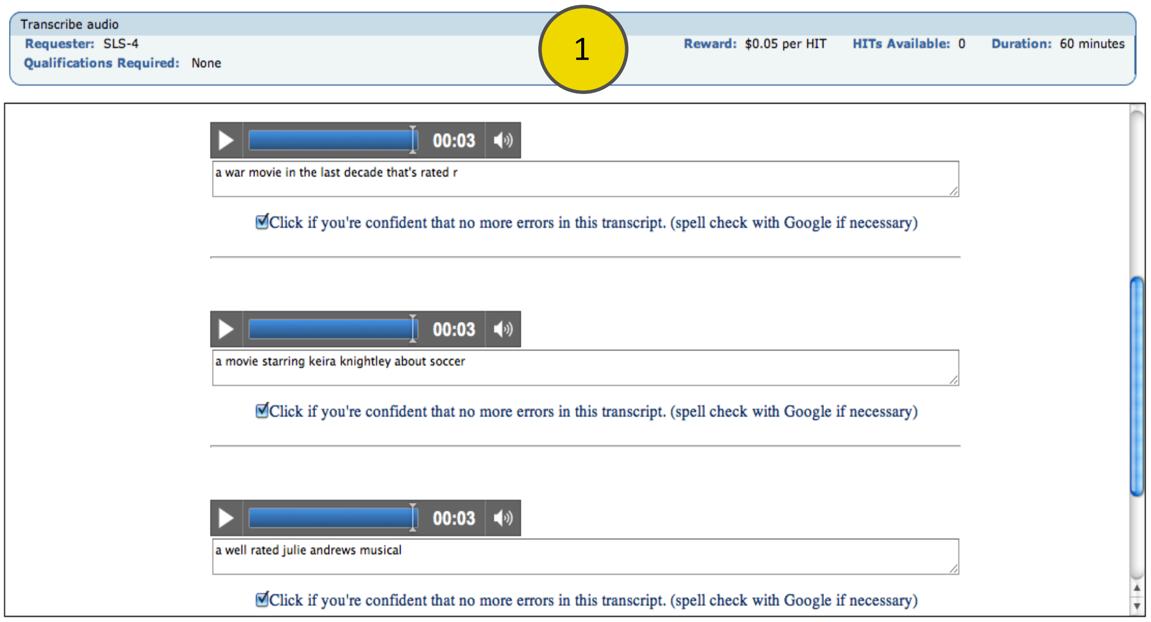
The Last Samurai (2003)
NR 100 min
Action | Adventure | Drama | History | War
William Atherton, Tom Cruise, Chad Lindberg, Ken Watanabe
In the 1870s, Captain Nathan Algren, a cynical veteran of the American Civil war who will work

Figure 5-2: *Movie Browser* is a spoken language interface for querying the IMDB database. The interface can accept natural language queries through speech or text. Since *Movie Browser* is a multi-modal web site, we can deploy it directly into mTurk. Workers can give us immediate feedback as to whether a query's response was correct through the check-boxes to the right of each movie.

about the text-based version of the *Movie Browser* are described in [93].

Here we concentrate on the automatic speech recognition component of this system. Decoding is handled by SUMMIT, which is set up in a fashion similar to the PhoneBook experiments described above. The language model of the recognizer uses a class n-gram, such as the one described in 2.3.2, to robustly model carrier phrases. Thus, the query above would appear to the LM as *"What are some CHARACTER movies starring ACTOR?"* Separate FSTs are built and dynamically inserted for each class. The ten manually chosen classes are: Actor, Genre, Director, Year, Rating (e.g., PG13), Character, Plot element, Title, Song, and Evaluation (e.g., "well-rated").

To limit its size, our recognizer does not include the full list of over 200,000 actors, directors, and movie titles in our database. Instead, we initialize the classes with lists of *popular* actors, directors, titles, etc, that were scraped from the web. The lists contained almost 3,000 movie titles and over 1,000 popular actors. They were, however, a few years old, and thus somewhat stale with respect to the domain. We hoped the recognizer would


Transcribe audio
Requester: SLS-4
Qualifications Required: None
Reward: $0.05 per HIT HITs Available: 0 Duration: 60 minutes

a war movie in the last decade that's rated r

☑ Click if you're confident that no more errors in this transcript. (spell check with Google if necessary)

a movie starring keira knightley about soccer

☑ Click if you're confident that no more errors in this transcript. (spell check with Google if necessary)

a well rated julie andrews musical

☑ Click if you're confident that no more errors in this transcript. (spell check with Google if necessary)

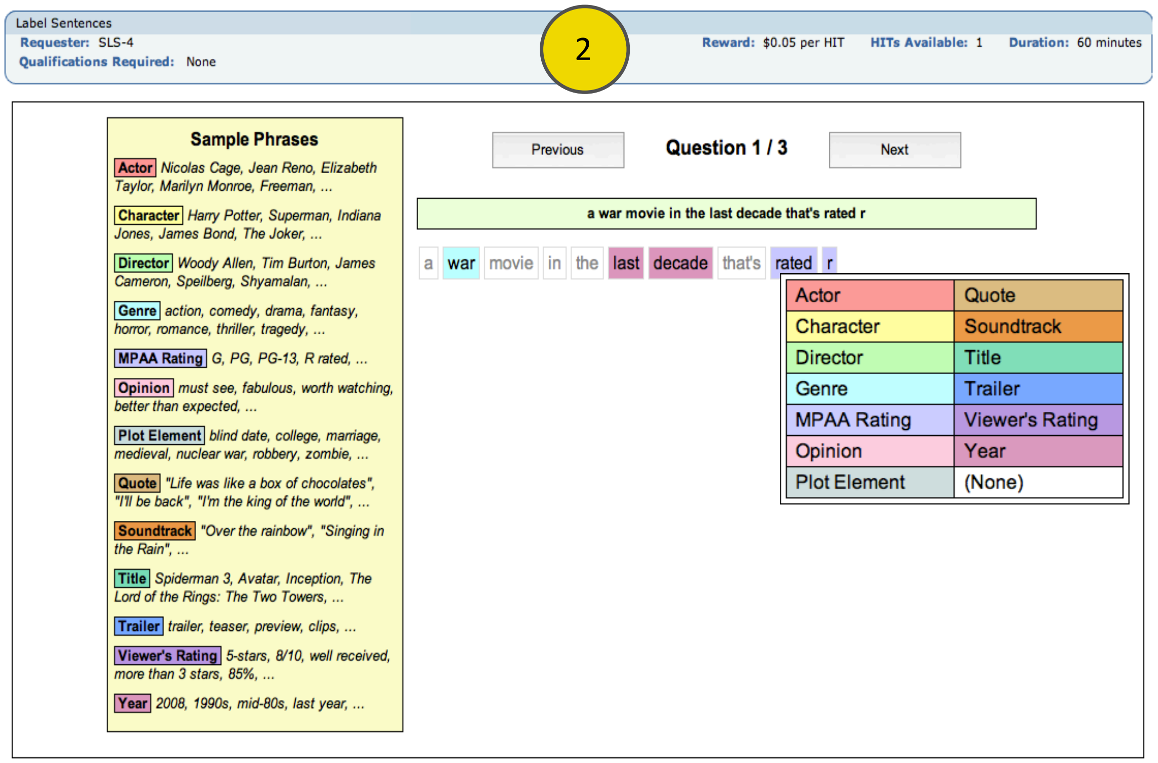
Figure 5-3: An iterative transcription HIT. Utterances are batched together; however, they are processed individually when determining whether each utterance requires another iteration before considering it transcribed. Before a particular transcript is considered final, a worker must not make edits and must also click on the verification checkbox.

learn missing lexical items on-the-fly via crowdsourcing. To initialize our lexicon, we used pronunciations from our expert dictionary if they existed and relied on a graphone-based L2S model if they did not.

5.3.2 Crowd-supervised Training Library

In this section, we introduce a library of crowdsourcing tasks designed for processing spoken language data and describe how we connect these tasks together in order to fully automate the retraining of certain components of a spoken language system. The larger vision of this work is to substantially reduce, if not entirely eliminate, the need for the expert intervention and ad-hoc boot-strapping techniques typically employed in spoken language system development. To this end, we have been developing and fine-tuning a set of Human Intelligence Tasks deployable to mTurk. Three of these HITs, transcription, segment tagging, and prompted speech collection, are described here.

A transcription HIT has been shown to be a valuable tool for obtaining orthographies of audio. Recall that combining multiple noisy transcriptions with techniques such as ROVER is known to significantly improve transcription quality [99]. Our transcription HIT, shown in Figure 5-3, deals with noisy data in a somewhat simpler manner. Each audio clip must


Label Sentences

Requester: SLS-4
Qualifications Required: None

2

Reward: $0.05 per HIT **HITs Available:** 1 **Duration:** 60 minutes

Sample Phrases

Actor *Nicolas Cage, Jean Reno, Elizabeth Taylor, Marilyn Monroe, Freeman, ...*

Character *Harry Potter, Superman, Indiana Jones, James Bond, The Joker, ...*

Director *Woody Allen, Tim Burton, James Cameron, Speilberg, Shyamalan, ...*

Genre *action, comedy, drama, fantasy, horror, romance, thriller, tragedy, ...*

MPAA Rating *G, PG, PG-13, R rated, ...*

Opinion *must see, fabulous, worth watching, better than expected, ...*

Plot Element *blind date, college, marriage, medieval, nuclear war, robbery, zombie, ...*

Quote *"Life was like a box of chocolates", "I'll be back", "I'm the king of the world", ...*

Soundtrack *"Over the rainbow", "Singing in the Rain", ...*

Title *Spiderman 3, Avatar, Inception, The Lord of the Rings: The Two Towers, ...*

Trailer *trailer, teaser, preview, clips, ...*

Viewer's Rating *5-stars, 8/10, well received, more than 3 stars, 85%, ...*

Year *2008, 1990s, mid-80s, last year, ...*

Previous **Question 1 / 3** Next

a war movie in the last decade that's rated r

a war movie in the last decade that's rated r

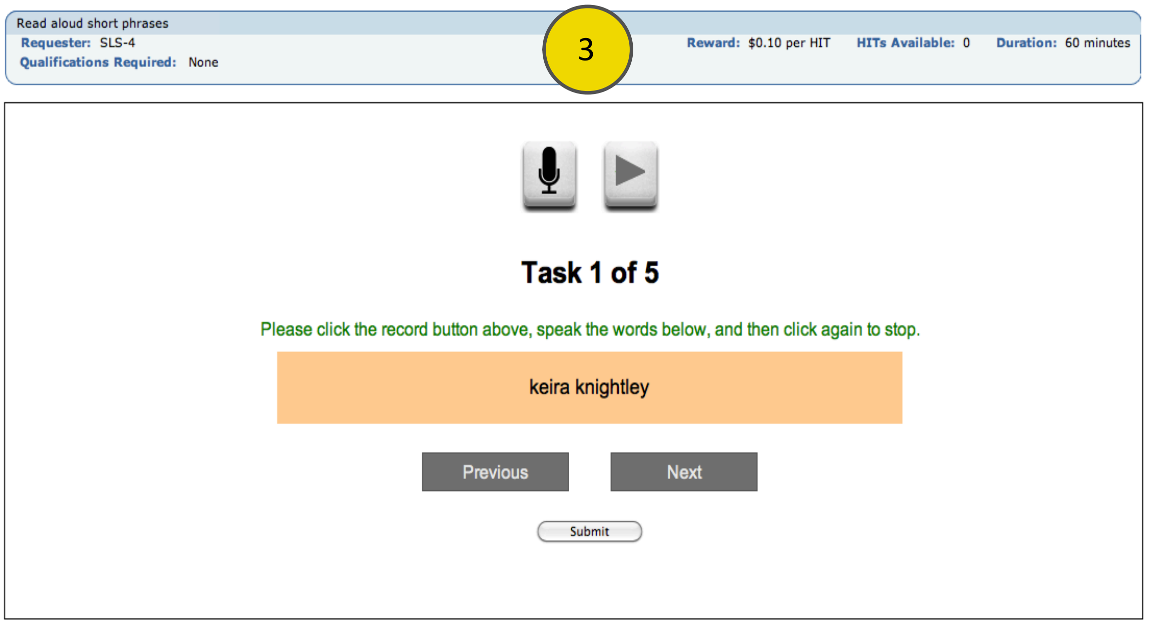
Actor	Quote
Character	Soundtrack
Director	Title
Genre	Trailer
MPAA Rating	Viewer's Rating
Opinion	Year
Plot Element	(None)

Figure 5-4: A semantic tagging HIT. In the *Movie Browser* domain, phrases can be labeled with a category. This category can be used, along with the input sentence, to train a language model's class N-gram or the semantic interpreter's conditional random field. Note that sample phrases for each category are given on the left. The additional categories in this list were not part of the study described here.

be shown to two workers, and the job of the second worker is to improve upon or verify the transcript of the first worker. While perhaps not fully addressing the problem of noise, this procedure is sufficient for the training needs of this work and can be easily extended with more iterations of improvement or approval.

Semantic labeling, whereby phrases in a sentence are tagged with a category label, is another common NLP task for which mTurk has proven useful [95]. Our semantic labeling HIT, shown in Figure 5-4, uses a click-and-drag selection mechanism to allow workers to semantically tag multiple words with a single label. In our case, the labels consist of the ten classes found in our recognizer. Although it is not difficult to imagine combining multiple outputs to reduce noise, for this HIT we took a domain-dependent approach described later.

The final domain-independent HIT is the prompted audio collection task depicted in Figure 5-5. Here, a worker simply reads aloud the prompt on the page. While it is difficult to control the properties of the speech collected, we have found that forcing workers to listen to their own speech helps to address microphone problems. As described in Chapter 3,


Read aloud short phrases
Requester: SLS-4
Qualifications Required: None
Reward: $0.10 per HIT
HITs Available: 0
Duration: 60 minutes

Task 1 of 5

Please click the record button above, speak the words below, and then click again to stop.

keira knightley

Figure 5-5: A prompted speech collection HIT. This HIT is similar to the one described in Chapter 3. It has been updated to use Flash, but the mechanics remain the same.

restricting the country to which a HIT is deployed provides rough controls over accented-speech. Furthermore, in this work, we collect eight examples of each prompt for use in a pronunciation mixture model, and as shown in the previous section this redundancy helps to ensure that a few noisy examples do not cause too many problems.

5.3.3 An Organic Training Framework

With our library complete, we now describe a domain-dependent HIT designed to collect data for the very system we are training. Figure 5-2 depicts the *Movie Browser* system in a HIT. This particular instantiation of the HIT provides the worker with a scenario which they must attempt to accomplish. Workers may be asked, for instance, to use the *Movie Browser* to find movies with ACTOR: Keanu Reeves and GENRE: Action. Once the user speaks a query, the movie search results are displayed along with feedback checkboxes which workers must use to indicate whether a result was satisfactory. Specifically, a worker checks a box next to each movie that is relevant to the query they made. Upon submission, we collect the last utterance spoken and the list of movies marked relevant.

With the system HIT in place, we have all the pieces necessary to implement the crowd-supervised training of our system. To be fully hands-free, however, we must be able to use the output of one HIT as the input of the next. To do so, we rely on TurKit [92]. This open-source project allows us to implement simple JavaScript programs to manipulate HITs

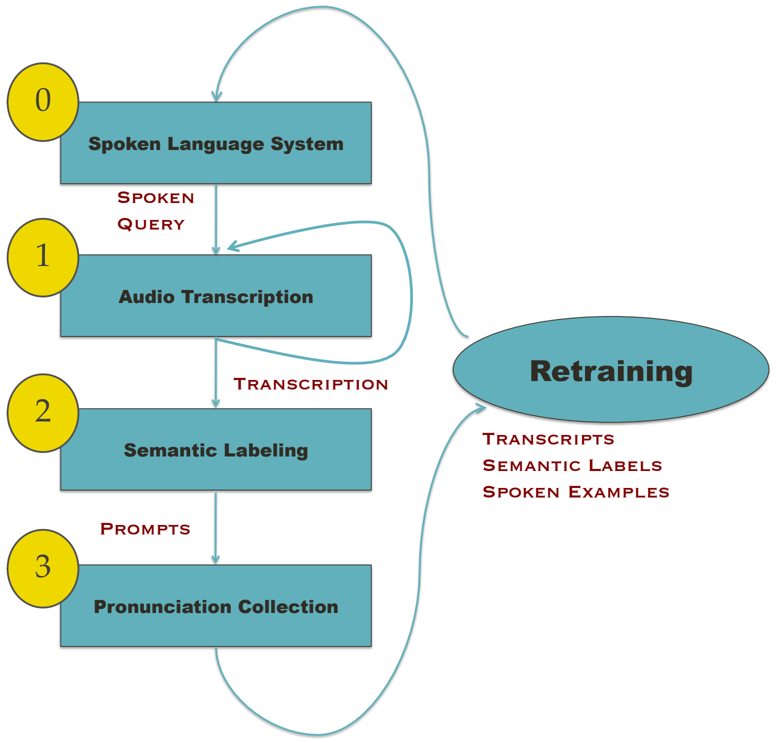


Figure 5-6: The system HIT and three domain-independent HITs compose our crowd-supervised training architecture. TurKit allows us to connect each of these HITs programmatically using simple JavaScript code. This allows us to create an entirely hands-free training procedure.

arbitrarily. The following code conveys the ease with which we can take audio collected from our system HIT and pass it along into our generic transcription task:

```
var outputs = systemHIT.outputs();
var transcribeHIT = new TranscriptionHIT();
for (var i = 0; i < outputs.length; ++i) {
    transcribeHIT.add(outputs[i].audioUrl);
}
```

We extend this code to collect data with HIT (0) and pass it through HITs (1)-(3) according to the flow chart in Figure 5-6.

Noise control is relatively straightforward to implement using TurKit. The transcription HIT has a loop in the flowchart which indicates that once the audio is transcribed by the first worker, it is then passed through the HIT a second time to be verified by a second worker before moving on to the semantic labeling stage. When the labeling is finished, we compare the transcript to the recognition results in search of semantically important words that were misrecognized. Before we generate prompt HITs using these words, however, we employ a domain-specific form of noise control. In particular, each term is run through our movie search index to ensure its validity. Once the transcribed and semantically tagged term is validated, we begin to collect spoken examples of it.

To perform repeated operations over a large number of HITs without redoing unnecessary work, TurKit relies heavily on the concept of *memoization*. We briefly explain how memoization effectively caches human computation using the code above as an example. First, each pass through the HIT extracts the output from every assignment of the system HIT. The `transcribeHIT.add` method, however, only creates a new HIT if its input, the URL of the audio, has not been used to create a previous HIT. When the new HIT is created, those inputs are recorded in a database so that the next time through, a new HIT is *not* created. Extrapolating this example to the series of HITs in the flowchart of Figure 5-6, we see that each assignment for each HIT is touched once per execution of the script, but only newly submitted work can spark the generation of a new HIT. Finally, TurKit has the ability to run a script repeatedly, causing a wave of updates to occur at each iteration.

To close the loop in the crowd-supervised training process, our system must be able to retrain and dynamically load new models on-the-fly. Given the spoken examples, we employ a pronunciation mixture model to learn new words and refine existing pronunciations in our lexicon. For the PMM used in our experiments, we run one iteration of EM using a 5-gram graphone LM over singular graphones. We normalize the learned parameters by word and use them in a stochastic lexicon. Pronunciations with probability less than 0.1 are excluded. With the transcripts and semantic labels, we train a class-based language model. Two additional obvious candidates for retraining not explored in this chapter are the acoustic model and the conditional random fields, [85], used for semantic interpretation. The former will be explored in a different domain in the next chapter, while the latter is beyond the scope of this thesis.

5.3.4 *Movie Browser* Experiments

To experiment with our crowd-supervised training framework, we decided to limit our domain to a set of scenarios. Rather than choosing our own scenarios, however, we ran a quick mTurk task for scenario collection. This task consisted of a simple HTML form, for which a worker was asked to fill out two or three of the ten classes with reasonable values, e.g. DIRECTOR: Darren Aronofsky, RATING: R. These constraints then make up the set of scenarios used in our experiments. We collected 373 scenarios and filtered them through the movie search index to ensure their presence in our database. Forty scenarios were used for development to test our training framework, and 127 scenarios were used in

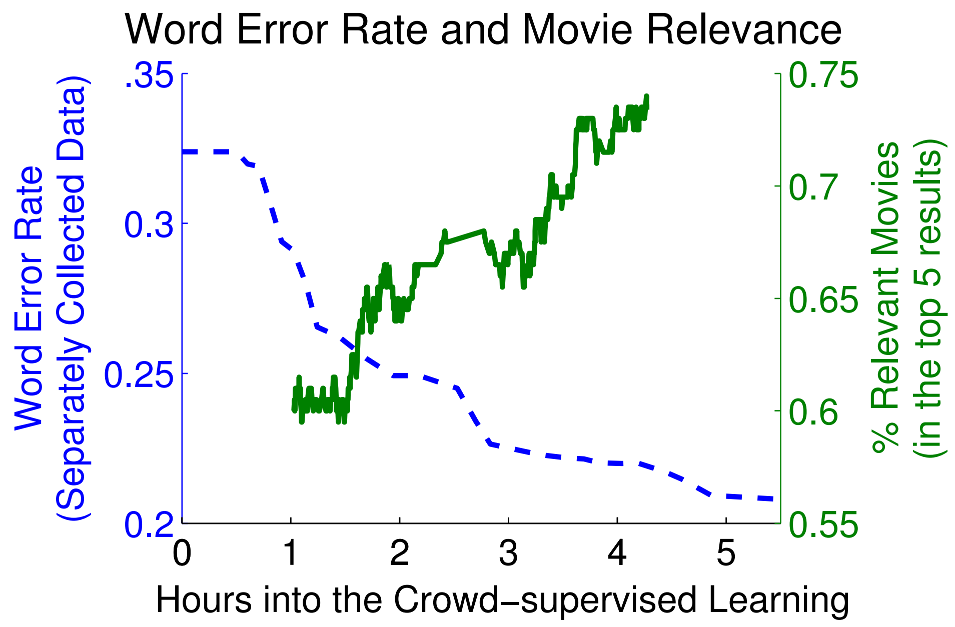


Figure 5-7: We show a windowed percentage indicating the fraction of the last 200 utterances that contain a relevant result in the top five movies returned. WER is computed on a separate dataset of 1001 utterances at various stages in the crowd-supervised learning process.

the experiments described below.

We began a small-scale crowd-supervised training experiment on a weekday evening and completed it roughly five-and-a-half hours later. All HITs were restricted to the US, and the prices of each HIT were set to optimize throughput rather than cost. Still, each utterance was collected, transcribed, and semantically labeled for less than a quarter. Over the course of deployment, 22 distinct workers contributed a total of 714 individual voice search queries to *Movie Browser* using the system HIT. Each worker was only allowed to complete a given scenario once. As workers completed the system HIT, their data progressed through the crowd-supervised training framework; about 45 minutes into the HIT, on-the-fly recognizer updates began to occur. Figure 5-7 shows that the on-the-fly training procedure has an appreciable effect on movie relevance, a metric computed from the worker's checkbox selections. We use a large window of 200 utterances to smooth out variation between users, and compute a moving average representing the fraction of the time that a relevant movie is in the top five search results. Since it is clear that the system improves, we now turn our attention to ascertaining how the gains are made.

To analyze the dynamics of the recognition, we took snapshots of the recognizer after each pass through the retraining phase. Rather than test on the same data, we decided

Category	A	G	D	R	C	T	P	all
Spoken	666	486	275	263	193	99	96	1986
Learned	74	19	27	6	27	17	16	177
Missing	27	1	10	3	20	6	10	67
Errors w/o lexicon update	23.0	20.2	22.6	22.5	64.3	49	58	29.8
Errors with L2S	5.9	20.0	7.0	18.6	18.7	23	10	13.5
Errors with L2S/PMM	**3.4**†	20.2	8.0	17.5	**13.3**∗	24	16	**12.5**∗

† PMM/L2S differences statistically significant with $p < 0.001$
∗ PMM/L2S differences statistically significant with $p < 0.05$

Table 5.1: Recognition error rates broken down by dynamic classes: *Actor, Genre, Director, Rating, Character, Title, and, Plot element*. We show the number of spoken instances, the number of pronunciations learned, an the number of pronunciations originally missing from each category. We give baseline error rates, L2S error rates and PMM error rates. The PMM error rates were arrived at after our human computation algorithm completed.

to collect and transcribe a separate dataset with the same scenarios using only HITs (0) and (1) from Figure 5-6. We made no effort to prevent workers who performed the first HIT from working on the second, and indeed 11 workers of the 38 that performed this HIT had done at least one assignment during the crowd-supervised training. Also, for this collection, workers were allowed to perform each scenario twice. The test set collection task was left running overnight, and by morning, we had collected and transcribed 1,179 utterances. Unlike the training phase, which was completely hands-free, some workers who contributed unintelligible audio were rejected. Using the final test set of 1,001 utterances, we compute a WER for each recognizer snapshot and plot it over time, which is plotted in Figure 5-7. Over the course of the crowd-supervised training, the WER of the test set drops from 32.3% down to 20.8%. The high error rates reflect both the difficulty of the domain and the noisy nature of crowd-sourced data.

The aforementioned results mix together the contributions of the crowd-supervised language model as well as the lexical components (L2S and PMM). To separate these effects, we are able to remove the training data of individual components and rerun recognition on our test set. For example, we determine the utility of retraining the language model via this post-processing experiment. In particular, we remove the crowd-supervised transcriptions from the LM training data while keeping the learned lexical items intact and recompute a WER on the test set. This causes the WER to bounce back up to 25.3%. Given that our baseline WER is 32.3%, this indicates that most of the gains are due to the improvements to the lexicon.

Closing the lexical gap between the mTurk collected scenarios and the initial recognizer lexicon is the largest source of gains. Sam Worthington, for example, the star of Avatar, was not on the original outdated list of actors. When an utterance containing this name was transcribed and labeled, the name was inevitably determined to be a misrecognition. As such, it was first validated against the search index, and then added to the list of prompts to be collected. In the meantime, the L2S model was used to generate a pronunciation. Table 5.1 shows the number of terms learned and missing from each category, as well as the number of times they were spoken in the test set. Categories that were not learned or with fewer then 10 spoken examples are omitted from the table. We see that actors make up the largest category, both in spoken examples (666) as well as individual lexical items learned (74).

Overall, our crowd-supervised framework affected 177 words in our lexicon. We examine the recognition error rate of these words directly to get a picture of how the L2S and PMM affect recognition. Table 5.1 breaks the errors down by category. We first compute the error rate of each category *before* any updates to the lexicon took place. They are relatively low due to the missing lexical items. Directors, for example, have a baseline error rate of 22.6%. After crowd-supervised training takes place, the error rate for this category drops up to 8.0%, as seen in the final row of the table, which shows the ultimate gains achieved by our framework.

The effects of the PMM can be inferred by first removing the pronunciations learned from spoken examples and replacing them with their L2S counterparts. The second-to-last row in the table of Table 5.1 depicts the accuracies of these words using only the L2S model. Where the differences between the PMM and L2S are statistically significant under McNemar's test, the PMM consistently improves over the L2S approach. In particular, recognition of actors' names was significantly improved despite having a low L2S baseline error rate of 5.9%. For example, the last name of Cary Elwes, of Princess Bride fame, was given the pronunciation `eh l w eh s` by the L2S. The PMM's top pronunciation was more accurate: `eh l w ey z`. We were surprised that the workers knew the pronunciation of this name, but listening to their speech confirmed that they did. Even if they had not, learning mispronunciations can also be beneficial.

Finally, it is nice to see that, in aggregate, the PMM yields significant improvements over the L2S approach. The error rate for the 177 learned words after crowd-supervised learning was 12.5%. This means that that our system recognizes a large portion of the content words correctly. One interesting avenue of future research may be to explore how well the semantic interpreter, which is based on conditional random fields, is able to recover from misrecognition of non-content words.

5.4 Summary

In this chapter, we have concentrated on the application of crowdsourcing and human computation to the pronunciation dictionary component of an automatic speech recognizer. We have introduced the pronunciation mixture model as a means of improving the lexicon without the aid of an expert. In two different experiments we have shown that the PMM is robust to noisy acoustic examples from the crowd.

The final set of experiments in this chapter employed an automatic crowd-supervised training framework and demonstrated its utility with respect to updating a recognizer's lexicon *and* language model. While cinema's ever-shifting lexical domain highlights the need for systems that can grow and change to accommodate previously unknown words, we believe the framework itself may be generally useful to replace the manual configuration and bootstrapping that accompanies building a new spoken language system. Again, most of the tools used in the chapter are open-source, enabling other researchers to test similar systems.

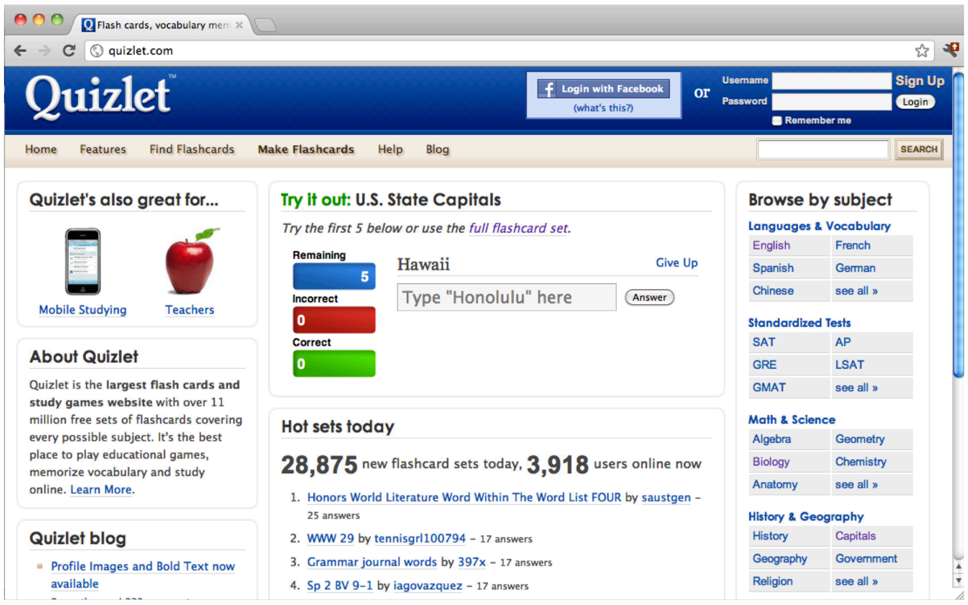
Chapter 6

Crowd-supervised Acoustic Modeling

In the preceding chapters we relied largely on Amazon Mechanical Turk to provide the crowd in our human computation tasks. We could do so in this chapter as well, eliciting speech for the purpose of retraining the acoustic model. To the best of our knowledge, this has not yet been attempted with mTurk. While we feel this is a worthy avenue of exploration, we use this chapter to address a possible shortcoming of the work we have presented so far. Specifically, we would like to move beyond the micropayment platform and present a mechanism for collecting and, in some cases, transcribing speech, *for free*.

To this end, this chapter presents two educational speech-enabled games. The first game, called *Voice Race*, collects isolated words or short phrases. The second, called *Voice Scatter*, elicits continuous speech in the form of longer sentences. Both of these games were hosted on a popular educational website for a little less than a month, during which time data were logged and gathered. We then ran a suite of experiments to show that some of these data could be accurately transcribed simply by looking at the game context in which they were collected. Using these data, we show that the acoustic model can be adapted to yield improvements otherwise unattainable in an unsupervised fashion. This protocol does not fall under our definition of human computation, since we perform these experiments in an offline fashion; however, we believe this represents a significant step towards what *could* become a fully organic spoken language interface.

The experiments in this chapter point to a broader characteristic of spoken language interfaces that, until now, has been overlooked in our study of crowd-supervised training – that is, many spoken language systems come with a crowd built-in. By definition, spoken language interfaces, when outfitted with an internet connection, have a suite of non-expert individuals who can provide valuable data. Since they are unpaid, however, we are restricted to collecting the data that are a byproduct of normal interaction with the system. To clarify this point, we examine the voice-dialing work of Li et al. [91]. Here, data are



Try the first 5 below or use the full flashcard set.

About Quizlet

Quizlet is the **largest flash cards and study games website** with over 11 million free sets of flashcards covering every possible subject. It's the best place to play educational games, memorize vocabulary and study online. Learn More.

Quizlet blog

- Profile Images and Bold Text now available

Hot sets today

28,875 new flashcard sets today, **3,918** users online now

1. Honors World Literature Word Within The Word List FOUR by saustgen – 25 answers
2. WWW 29 by tennisgrl100794 – 17 answers
3. Grammar journal words by 397x – 17 answers
4. Sp 2 BV 9-1 by iagovazquez – 17 answers

Figure 6-1: *Quizlet.com* allows users to create, share, and study their own flashcards.

collected from mobile phone users who speak names from their contact lists to dial a call. A confirmation is then given, to which the user must reply "yes" or "no". If the caller says "yes" and the call goes through, Li et al. make the assumption that the spoken name was recognized correctly. While there is some noise added in the form of false positives, they find this confirmation question is a useful means of collecting labeled training data.

The work in this chapter provides a similar approach to determining when a recognition result is correct. We do so, however, without the aid of an additional confirmation step, by using domain-specific context to validate the semantics of a recognition result. While particularly useful in the education domain, where words are known, we contend that such techniques might be applied to other spoken language interfaces to obtain similarly *self-transcribed* data. In the next few sections we detail our experiments with *Voice Race* and *Voice Scatter*, and show that the crowd's *self-supervised* retraining of the acoustic models improves recognition accuracy.

6.1 *Quizlet*

The speech-enabled games we study in this chapter are a fun way to review flashcards to help one memorize vocabulary words, scientific terms, mathematical concepts, etc. They can be played by anyone with a web browser and a microphone. *Voice Race* and *Voice Scatter* were built and deployed to *Quizlet.com*[1] with the help of its founder. The *Qui-*

[1]http://www.quizlet.com

zlet website, shown in Figure 6-1, is a place where users can create, share, and study virtual flashcards. Like real flashcards, each virtual card has two sides: typically one is used for a *term* (a word or short phrase) and the other for its *definition*. At the time we deployed our games to *Quizlet*, the site had over 420,000 registered users, who had contributed over 875,000 flashcard sets, comprised of more than 24 million individual flashcards. Now there are closer to 6 million users studying over 9 million sets with more than 300 million study terms.

Voice Race and *Voice Scatter* are based on two text-based games that had already existed on *Quizlet* for some time. The study material for each game is loaded from a single set of flashcards chosen by the user. The automatic speech recognition component was simple to incorporate into these games with the help of the publicly available WAMI Javascript API described in Section 4.1. When a game session is initialized, a simple context-free grammar is constructed using the terms in the chosen set of flashcards. The audio is streamed to our server, where speech recognition occurs and sends the results back to the Javascript event handlers in the browser.

On the server-side, The SUMMIT speech recognizer is used with a dictionary containing 145,773 words, and an automatic L2S module to generate pronunciations for those words that are found to be missing at run-time. A small context free grammar built over the terms and definitions in the chosen set serves as the language model for the recognition component of each game. While each instance of these games is a small-vocabulary recognition task over a particular set of flashcards, in aggregate the utterances we collected cover a large vocabulary, and should be useful in a variety of speech tasks.

In the spirit of other games with a purpose (GWAPs) described in Section 2.1, our games were designed to elicit data as a byproduct of game-play. Previous to this work, the only GWAP for a speech related task that we are aware of is *People Watcher* [112]. *People Watcher* elicits alternative phrasings of proper nouns, which are used to improve recognition accuracy in a directory assistance application. The use of an educational website to transcribe data was explored in [20], in which a task intended to help students learn a foreign language was deployed via a prototype website, and used by 24 students to label 10 sentences. Neither of these applications elicit new speech data, or are applied on a large scale.

Our games, on the other hand, collect audio recordings, as well as the context in which they occur. With the help of *Quizlet*, we find it easy to recruit a large number of willing subjects, which gives rise to a diversity of ages, genders, fluency, accents, noise conditions, microphones, and so forth. Furthermore, the games we design allow for the automatic transcription of a significant subset of the collected data, and for cheap transcription of the vast majority of the remainder. This means that an arbitrary amount of transcribed utterances may be collected over time at no, or slowly increasing, cost.

Our games are different from typical GWAPs in a few key respects. First, these are single-player games. Whereas typical GWAPs rely on the agreement of two humans to obtain labels, *Voice Race* instead uses the artificial intelligence of an automatic speech recognizer. Contextual constraints and narrow domain recognition tasks are paired to bootstrap the collection of transcribed corpora, which cover a larger vocabulary and a variety of noise conditions. Second, GWAPs label existing data, whereas *Voice Race* both elicits new speech data, and automatically transcribes much of it. Thus, while *Voice Race* cannot label arbitrary speech data, it can *continuously* provide new, transcribed speech without any supervision. Third, unlike GWAPs which offer only diversion, these educational games directly benefit their players by helping them to learn. Finally, users have control over their game content, which is not randomized as in many GWAPs.

6.2 *Voice Race*

In this section, we describe *Voice Race*, a speech-enabled educational game which we deployed to *Quizlet* over a 22 day period to elicit over 55,000 utterances representing 18.7 hours of speech. *Voice Race* was designed such that the transcripts for a significant subset of utterances can be automatically inferred using the contextual constraints of the game. Game context can also be used to simplify transcription to a multiple choice task, which can be performed by non-experts. We found that one third of the speech collected with *Voice Race* could be automatically transcribed with over 98% accuracy; and that an additional 49% could be labeled cheaply by Amazon Mechanical Turk workers. We demonstrate the utility of the self-labeled speech in an acoustic model adaptation task, which resulted in a reduction in the *Voice Race* utterance error rate.

In *Voice Race*, shown in Figure 6-2, definitions from a set of flashcards move across the screen from left to right. Players must say its matching term before a definition flies off the screen. Each such "hit" earns points and makes the definition disappear. If a definition is never hit, then the player is shown the correct answer and prompted to repeat it aloud. As the game progresses, the definitions move more quickly, and appear more frequently.

Because each utterance is collected in the course of playing the game, a combination of recognition N-best lists and game context can be used to automatically infer the transcripts for a significant subset of the utterances. Intuitively, when the top recognition hypothesis is known to be a correct answer, this is a strong indication that it is accurate. Using such constraints, 34% of the collected utterances were *automatically* transcribed with near per-

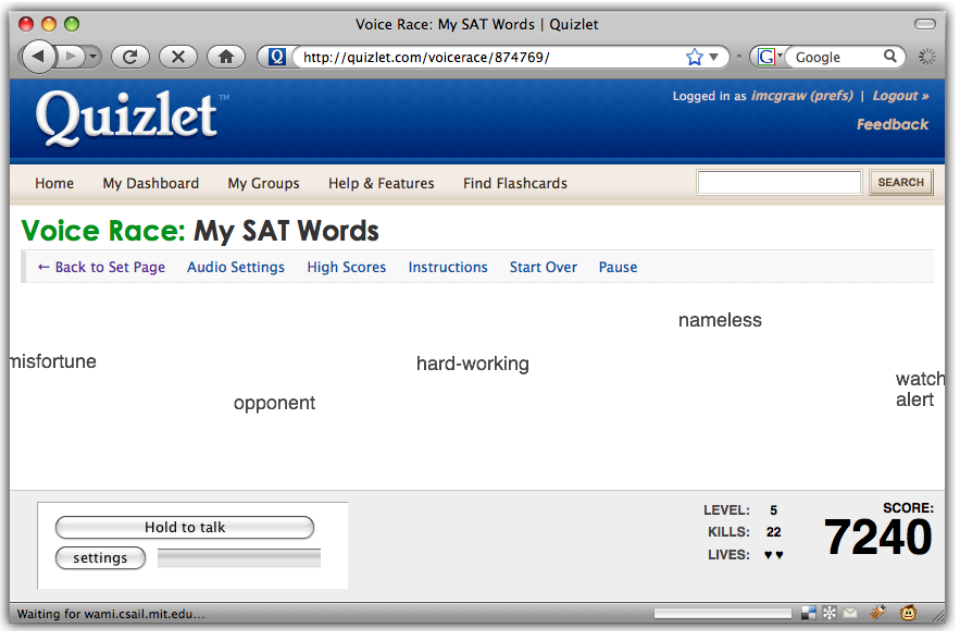


Figure 6-2: The *Voice Race* game with vocabulary flashcards. As a definition moves from left to right, the player must say the corresponding vocabulary word before it flies off the screen. Each such "hit" earns points and makes the definition disappear.

fect accuracy. For the remaining utterances, game context can also be used to simplify the task of human transcription to one of choosing among several alternative transcripts on a short list. Such a simple task is easy to complete with no training, so we used mTurk for transcription. To the best of our knowledge, when this work was performed there was no precedent in academia for using mTurk in this fashion. We found that the transcripts produced by mTurk workers were very close to expert-quality.

6.2.1 Self-Transcribed Data

We begin by precisely defining the way in which *Voice Race* is able to *self-transcribe* a portion of its data. Recall that each utterance occurs in a context where the correct answer (or answers) is known. This information, when combined with the recognition results, can be used to automatically infer the transcript for certain utterances, and greatly limit the set of likely transcripts for the rest. The subsets of interest are as follows:

Hit: In *Voice Race*, a "hit" occurs when the top speech recognition hypothesis contains the correct term associated with a definition visible on the screen. Players typically aim for the right-most definition, so such "hits" are likely to be the most reliable indicators of an accurate recognition hypothesis. Suppose, for example, that a student is learning state capitals. At any given time, only a few state names are shown on the screen. The probability of a *misrecognition* resulting in a state-capital that has a corresponding state onscreen is low. Even when terms are phonetically similar, such as the words "Austin" and "Boston", they are unlikely to co-occur in the game context, making them effectively distinguishable.

Miss: A "miss" occurs when the user has spoken, but a hit has not been detected. There is no way of knowing if a miss is due to a human error or a speech recognition error. However, when misses are due to recognition errors, the ground-truth transcript for the user's utterance is likely to be one of the correct answers. As such, when considered in aggregate, misses may be useful for automatically identifying difficult terms to recognize.

Prompted Hit/Miss: The taxonomy above applies to most *Voice Race* utterances. *Voice Race* also provides an additional category of labeled data: when a definition flies off the screen without being "hit", players are shown the correct answer and prompted to read it aloud. As such, when players are cooperative, the transcript of their utterances should be known in advance. Moreover, we run these utterances through the same small-vocabulary recognizer used for the game to notify the player of whether or not he or she was understood. These utterances can therefore again be classified as "hits" or as "misses".

6.2.2 Simplified Transcription

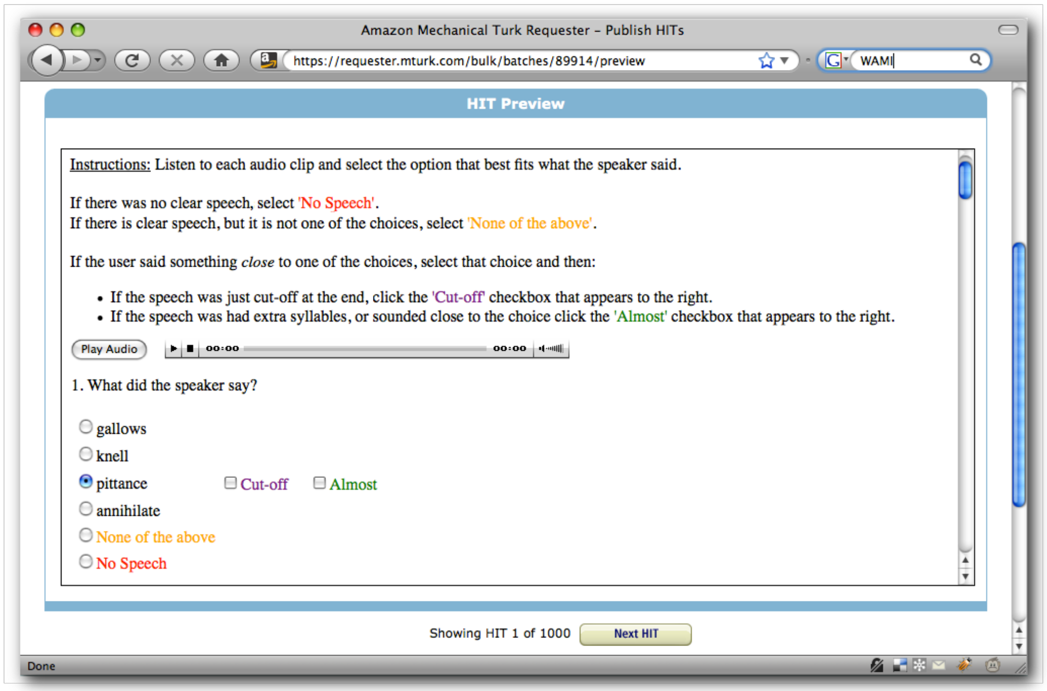
The contextual game constraints identified in the previous section, and in particular the "hits", are useful for automatically transcribing a significant portion of the data. In addition, the same constraints may be used to greatly decrease the *human* effort required to transcribe the remainder. For each utterance, the transcript is likely to be one of the correct answers, to appear on the N-best list, or both. This means that the task of human transcription for most utterances can be reduced to one of choosing a transcript from a short list of choices drawn from these two sources. Given that it requires no expertise or knowledge of the task domain to listen to a short audio clip and choose a transcript from a list, we designed a transcription task which could tap the large pool of mTurk workers.

We designed the mTurk transcription task such that workers listen to a *Voice Race* utterance and then choose from one of four likely transcripts. They can also choose "None of these"' or "Not Speech". The likely transcripts were drawn in order from the sources found in Figure 6.1, until four unique candidate transcripts were obtained to create the six answer multiple choice transcription question. An example mTurk task is shown in Figure 6-3.

After transcribers select a transcript, they can optionally label two additional attributes. *Cut-off* indicates that the speech was cut off – this happens occasionally because players release the space bar, which they must hold while speaking, before they finish. Future

1. The prompted term, if the user was asked to repeat aloud
2. The top two distinct terms in the recognition N-best list
3. The terms associated with the two right-most definitions
4. Any remaining terms on the N-best list
5. Random terms from the flashcard set

Table 6.1: The construction of the multiple-choice mTurk task selects candidate transcriptions in the order listed above.


Amazon Mechanical Turk Requester – Publish HITs
https://requester.mturk.com/bulk/batches/89914/preview
WAMI
HIT Preview
Play Audio
00:00
00:00
Showing HIT 1 of 1000
Next HIT
Done

Instructions: Listen to each audio clip and select the option that best fits what the speaker said.

If there was no clear speech, select 'No Speech'.
If there is clear speech, but it is not one of the choices, select 'None of the above'.

If the user said something *close* to one of the choices, select that choice and then:

- If the speech was just cut-off at the end, click the 'Cut-off' checkbox that appears to the right.
- If the speech was had extra syllables, or sounded close to the choice click the 'Almost' checkbox that appears to the right.

1. What did the speaker say?

gallows
knell
[✓ pittance] Cut-off Almost
annihilate
None of the above
No Speech

Figure 6-3: An mTurk task for transcribing *Voice Race* utterances.

iterations of the game will likely correct for this by recording slightly past the release of the key. Transcribers may also select *Almost* if the utterance was understandable, but contained hesitations, extra syllables, mispronunciations, etc.

6.2.3 Data Analysis

Voice Race was made available on Quizlet.com for a 22 day trial period. No announcements or advertisements were made. The two games were simply added to the list of activities available to study each (English) flashcard set. Nonetheless, as Table 6.5 shows, a total of 55,152 utterances were collected, containing 18.7 hours of speech.

Games Played	4184	Mean Words per Utt.	1.54
Utterances	55,152	Total Distinct Phrases	26,542
Total Hours of Audio	18.7	Mean Category Size	53.6

Table 6.2: Properties of *Voice Race* data collected over 22 days. Note in particular the large number of distinct words and phrases collected across the games.

5-way agreement	69.2%	Majority “None of these”	12.9%
4-way agreement	18.0%	Majority “cut-off”	12.1%
3-way agreement	9.8%	Majority “almost”	7.2%

Table 6.3: Agreement obtained for transcripts and attributes of 10,000 utterances, each labeled by five mTurk workers.

Transcription

10,000 utterances representing 173 minutes of audio were drawn from 778 *Voice Race* sessions and then submitted for transcription to Amazon Mechanical Turk (mTurk). Within 16 hours, each utterance had been labeled by 5 different mTurk workers using the simplified transcription task discussed in the previous section, at a cost of $275.

Table 6.3 shows agreement statistics for the workers. A majority agreed on one of the transcript choices for 97% of the utterances, agreeing on “None of these” only 13% of the time. Thus, the simple forced choice among 4 likely candidates (and “no speech”) yielded transcripts for 84% of the utterances.

To judge the accuracy of the produced labels, two experts each labeled 1,000 utterances randomly drawn from the set of 10,000. The interface these experts used to transcribe the data was identical to the mTurk worker’s. To assess inter-transcriber agreement we use Cohen’s Kappa statistic, [24]. The expert transcript choices showed a high level of agreement, with a Kappa score of 0.89. Each of their label sets agreed well with the majority labels produced by the mTurk workers, as measured by Kappa scores of 0.85 and 0.83.

Using the mTurk majority labels as a reference transcription, the utterance-level recognition accuracy on the set of 10,000 *Voice Race* utterances was found to be 53.2% . While accuracy is low, it’s important to note that the task is a very difficult one. The two experts noted while transcribing that (1) the vast majority of the utterances seemed to be from teenagers, (2) there was often significant background noise from televisions, music, or classrooms full of talking students, and (3) many microphones produced muffled or clipped audio. While these problems lead to imperfect speech recognition accuracy, they also lead to a richer, more interesting corpus. Moreover, usage levels suggest that accuracy was high enough for many successful games. In the next section, we show that, despite relatively poor recognition performance overall, it is nonetheless possible to use game context

Game Context:	miss	hit	prompted-miss	prompted-hit
% Correct:	13.9	86.4	12.7	97.5
% of Total Data:	43.7	43.8	8.9	3.6

Hit Context:	4-hit	3-hit	2-hit	1-hit
% Correct:	41.3	69.4	81.7	**98.5**
% of Hit Data:	1.8	3.4	9.0	**69.4**

Table 6.4: % of 10,000 mTurk-labeled utterances and self-transcription accuracy grouped by game context. Hits are further broken down in terms of the position of the item on the screen at the time the hit occurred. Statistics for the four right-most positions are shown.

to automatically obtain *near-perfect* transcriptions on a significant subset of the data.

Automatic Transcription

As previously described, because each utterance occurs in the course of playing *Voice Race*, we hypothesized that it should be possible to identify a subset of the data for which transcripts can be inferred automatically with high accuracy. In this section, we evaluate this hypothesis using as reference the transcripts agreed upon by a majority of mTurk workers. We explore the utility of *hits*, *misses*, *prompted-hits* and *prompted-misses*. Table 6.4 shows the amount of speech data collected in each category out of the 10,000 mTurk-labeled utterances.

Over 4,000 of the 10,000 utterances were *hits*, and the recognition accuracy on this data is 86.4%. In addition, *prompted-hits* yield an accuracy of 97.5%, meaning that they yield nearly perfectly transcribed data. Unfortunately, they represent less than 5% of the data.

Using game-context to filter data for accurately labeled utterances can be taken further in the case of a *hit*. Students are most likely to aim for the right-most label on the *Voice Race* screen. It stands to reason then, that *hits* of definitions which are not the right-most one are more likely to be due to a misrecognition. We call a hit that occurred while the item was in the *nth* position on the screen (from right-to-left) an *n-hit*. Recognition accuracies for $n = 1 \ldots 4$ are presented in Table 6.4.

It is exciting to note that *1-hits* constitute 30.4% of the total mTurk-labeled data, and are recognized with 98.5% accuracy. Of all 55,152 utterances collected, 18,699 – representing 5.8 hours of audio – are self-labeled in this fashion.

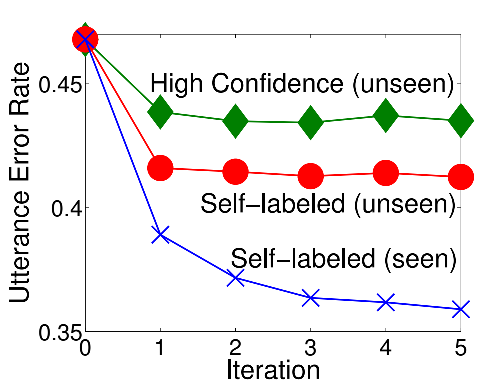
Self-Supervised Acoustic Model Adaptation

One common use of transcribed speech data is to perform acoustic model adaptation. While typically this requires human transcription, we explore using the automatically transcribed utterances to adapt the telephone acoustic models used by *Voice Race* in a fully automatic fashion. We performed MAP adaptation, described in Section 2.3.4, using "1-hits" and prompted-hits. The 10,000 utterances transcribed by mTurk served as our test set, while the remaining 45,152 utterances without human labels were used for adaptation. Using these updated acoustic models to decode the test set resulted in a decrease in utterance error rate from 46.8% to 41.2%. To show that this *self-supervised* adaptation algorithm outperforms typical unsupervised approaches, we use the confidence module of our recognizer, [60], to extract high quality utterances for a similarly sized training set. The utterance error rate for a decoder based on models trained from these utterances is 43.9%.

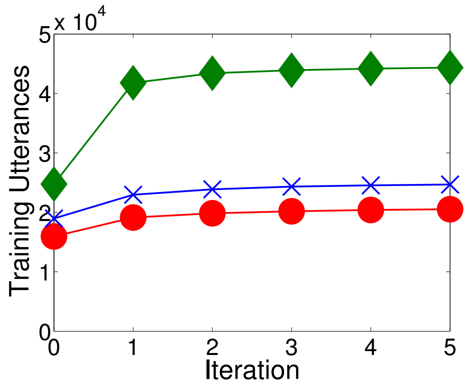
This self-supervised adaptation algorithm can also be run iteratively. Since we have saved the game context, we can actually recompute new hits and misses using the updated acoustic model. In other words, we can *re*-recognize a set of utterances and recompute the "hits", yielding a larger pool of self-labeled training data, which is then used in the following iteration. In theory, the new hits should still yield highly accurate transcripts. To test this hypothesis we iteratively trained and relabeled all 55,152 utterances until convergence. We found that 44.8% of the data could be *self*-labeled while maintaining an accuracy of 96.8% – computed on the subset corresponding to the mTurk-transcribed data.

With these promising results, we turn to analyzing the effects of the iterative approach on acoustic model adaptation. Note that, despite the self-supervised nature of our approach, training on all 55,152 utterances simulates having already *seen* the test data. We therefore perform separate experiments on the 45,152 utterance training set to simulate the case when the test data are *unseen*. We again use a confidence-based baseline approach in which, at each iteration, utterances that received high recognition confidence are selected for the new adaptation set. As Figure 6-4 shows, however, the confidence based approach is not amenable to this iterative adaptation procedure. By the second iteration, this approach uses over twice the data of the self-supervised method. However, the self-supervised method retains a 2.2% absolute improvement in error rate over the iteratively calculated high-confidence utterances through multiple iterations.

Figure 6-4 also shows the results of iteratively selecting from *all* 55,152 utterances (without using the mTurk-labels) treating the 10,000 utterance test set as *seen* data. Iteratively selecting high-confidence training utterances from all the data achieves error rates similar to those found when selecting self-labeled utterances from the original 45,152 utterances, and is omitted from the graph for clarity. Iteratively selecting *self-labeled* utterances from all of the data, however, improves performance significantly, even across iterations. The *iterative* gains are likely due to the fact that the self-adaptation set now includes utterances gathered from the same session, meaning that the speaker, acoustic environment, and



(a) Utterance Error Rate



(b) Training Set Size

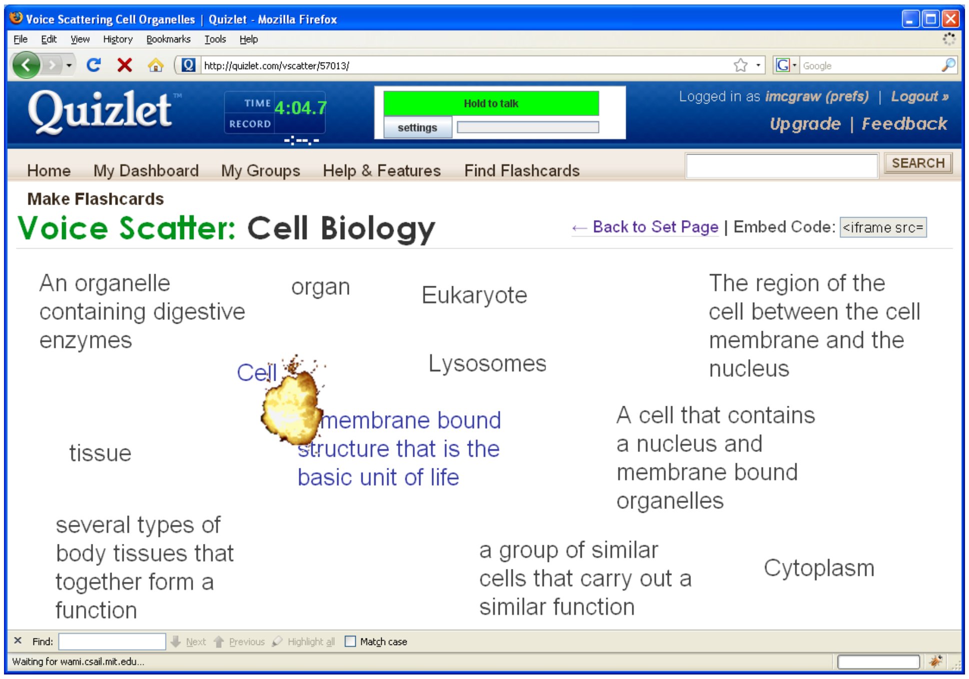
Figure 6-4: Iterative acoustic model adaptation, trained using: (1) Iteratively calculated high-confidence utterances, excluding the 10,000 mTurk-transcribed test set (i.e. the test data are unseen), (2) An iteratively calculated set of self-labeled utterances (unseen). (3) An iteratively calculated set of self-labeled utterances, including test utterances (seen).

vocabulary are the same. This hints at the potential for games like *Voice Race* to improve in a personalized, fully automatic, online fashion. The elicited utterances, however, typically contain only a single word or short phrase. To explore collecting labeled continuous speech, we now turn to *Voice Scatter*, which elicits significantly longer utterances.

6.3 *Voice Scatter*

In this section, we present *Voice Scatter*, which again relies on the resources of *Quizlet* to provide a speech-enabled educational game to a large user-base. A byproduct of its use is the collection and orthographic transcription of a significant amount of *continuous* speech. We describe experiments which made the game available on *Quizlet* for a 22 day period and resulted in the collection of 30,938 utterances, constituting 27.63 hours of speech. Each individual game uses only eight flashcards, and speech recognition was again performed using a narrow-domain CFG grammar. Despite the limited domains of each individual game, an estimated 1,193 speakers played the game with 1,275 distinct flashcard sets, so recognition hypotheses in the corpus cover 21,758 distinct words.

Unlike in the previous section, here we explore the possibility of combining the confidence-based approach with the self-transcription approach. The best technique pairs confidence scores from narrow-domain speech recognition with information from the game context about whether a hypothesis represents a correct answer. In this way, we automatically identify a sub-corpus of 39% of the data for which recognition hypotheses can be taken to be human-quality orthographic transcripts. We establish human agreement levels,


Voice Scattering Cell Organelles | Quizlet - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://quizlet.com/vscatter/57013/
Quizlet
TIME 4:04.7
RECORD -:--.-
Hold to talk
settings
Logged in as imcgraw (prefs) | Logout »
Upgrade | Feedback
Home My Dashboard My Groups Help & Features Find Flashcards
SEARCH
Make Flashcards
Voice Scatter: Cell Biology
← Back to Set Page | Embed Code: <iframe src=
organ
Eukaryote
Lysosomes
Cell
tissue
Cytoplasm
Find: Next Previous Highlight all Match case
Waiting for wami.csail.mit.edu...

An organelle containing digestive enzymes

The region of the cell between the cell membrane and the nucleus

membrane bound structure that is the basic unit of life

A cell that contains a nucleus and membrane bound organelles

several types of body tissues that together form a function

a group of similar cells that carry out a similar function

Figure 6-5: Screenshot of *Voice Scatter*.

and obtain manual transcripts of a 10,000 utterance development set, by crowdsourcing the transcription task via Amazon Mechanical Turk. When compared to a 1,000 utterance subset transcribed by experts, the crowdsourced transcripts show near expert-level agreement.

A screenshot of *Voice Scatter* is shown in Figure 6-5. Players first choose (or create) a set of flashcards to study. Then, up to eight terms and definitions are "scattered" randomly across the screen. Using a microphone and a web browser, players speak short commands to connect each term to its definition: e.g. "*match cell to a membrane bound structure that is the basic unit of life.*" Players hold the space bar, or click an on-screen hold-to-talk button, while speaking. When a term is correctly paired with its definition (a "hit"), they come together in a fiery explosion, and then disappear from the screen, as shown in Figure 6-5. When they are incorrectly paired (a "miss"), they collide and then bounce off of each other. A timer counts upward at the top of the screen, encouraging (though not requiring) players to set a speed record for the flashcard set.

Again, speech recognition was incorporated using the WAMI Javascript API and the SUMMIT speech recognizer. The following simple context free grammar is used as the speech recognizer's language model:

```
[match] <TERM> [to|with|and|equals] <DEF>
[match] <DEF>  [to|with|and|equals] <TERM>
```

where the brackets indicate optionality, and TERM and DEF are any of the terms or defini-

Games Played	4,267	Distinct Words Recognized	21,758
Utterances	30,938	Total Number of “Hits”	10,355
Hours of Audio	27.63	Recognized Words per “Hit”	8.327
Distinct Speakers[†]	1,193	Distinct Flashcard Sets	1,275

Table 6.5: Properties of *Voice Scatter* data collected over 22 days. [†]Distinct speakers are estimated as one speaker per IP address.

tions on the screen as the game begins.

6.3.1 Corpus Overview

Voice Scatter elicits utterances containing spontaneous continuous speech; however, because terms and definitions are visible on the screen, utterances – especially long ones – sometimes have the feel of being read aloud. While there is no specific requirement that players read the terms and definitions verbatim, there is a strong incentive to do so to avoid speech recognition errors. In addition, some (but certainly not all) players speak quickly because of the timer displayed during game play.

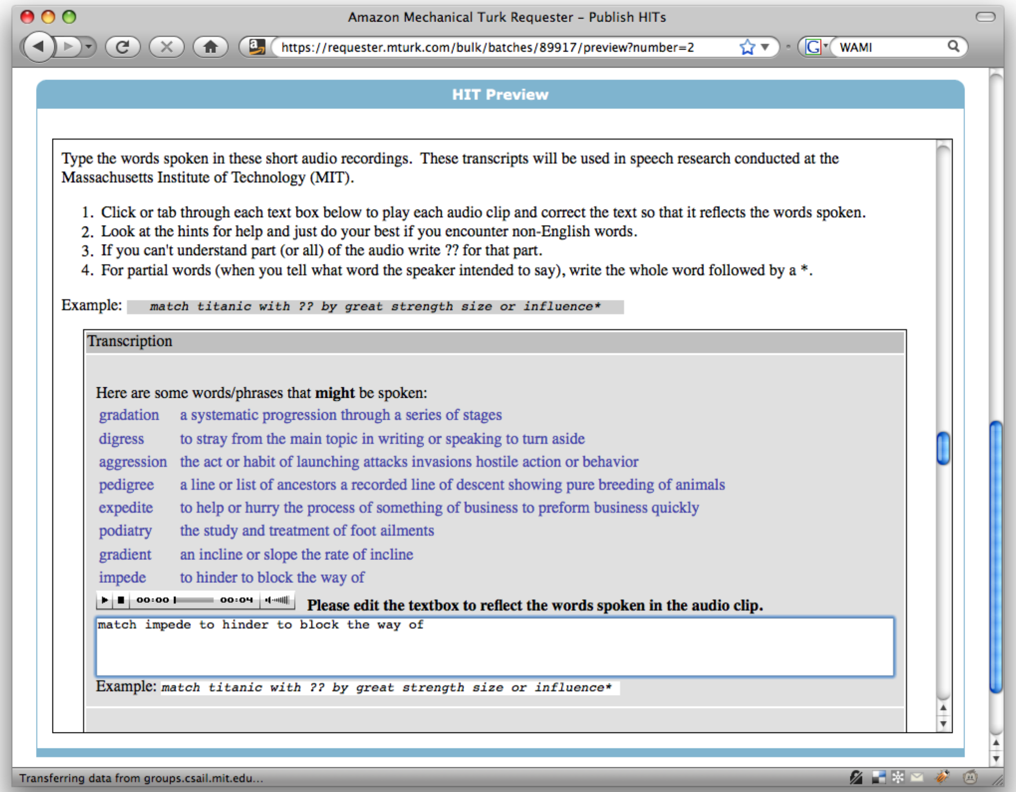
Table 6.5 gives a quantitative summary of the collected data. However, the type and variety of the data can be immediately understood by examining the sample transcripts shown in Table 6.6. As is shown, even though each individual *Voice Scatter* game is restricted to a small vocabulary, in aggregate there is a large and varied vocabulary. Moreover, by examining a random sample of utterances, we noted that almost all speakers appeared to be teenagers, and that utterances were recorded both in quiet and noisy environments. Noise typically came from televisions, music, computer noise, and people talking in the background. Finally, since players are trying to master unfamiliar material, some words are mispronounced. We observed one player, for example, who consistently mispronounced vocabulary words like “proliferate”, “unanimity”, and “steadfast”.

6.3.2 Crowdsourced Transcription

We used mTurk to orthographically transcribe 10,000 *Voice Scatter* utterances drawn from 463 random users (as determined by IP address), which totaled 11.57 hours of speech. The mTurk task is shown in Figure 6-6. Workers were given 10 utterances per page to transcribe. A text box for transcription was initialized with the speech recognizer’s top hypothesis, and workers were asked to edit it to reflect the words actually spoken. To guide the transcriber, each utterance was accompanied by a list of terms and definitions from the game associated with that utterance. Each utterance was transcribed by three different

match aimless to drifting
match robust to strong and vigorous
local area network lan
match silk road with an ancient trade route between china and europe
anything that makes an organism different from others variation
match malaise to a physical discomfort as a mild sickness or depression
match newtons first law of motion to an object at rest tends to stay at rest and a moving object tends to keep moving in a straight line until it is affected by a force
match what does friar lawrence point out to get romeo to see that life isnt so bad juliet is alive and still his wife tybalt wanted to kill romeo but romeo killed him instead the prince could have condemned him to death but he banished him instead

Table 6.6: Example transcripts drawn from the *Voice Scatter* corpus.


Amazon Mechanical Turk Requester – Publish HITs
https://requester.mturk.com/bulk/batches/89917/preview?number=2
WAMI
HIT Preview
Transcription
00:00
00:04
Transferring data from groups.csail.mit.edu...

Type the words spoken in these short audio recordings. These transcripts will be used in speech research conducted at the Massachusetts Institute of Technology (MIT).

1. Click or tab through each text box below to play each audio clip and correct the text so that it reflects the words spoken.
2. Look at the hints for help and just do your best if you encounter non-English words.
3. If you can't understand part (or all) of the audio write ?? for that part.
4. For partial words (when you tell what word the speaker intended to say), write the whole word followed by a *.

Example: `match titanic with ?? by great strength size or influence*`

Here are some words/phrases that **might** be spoken:

gradation	a systematic progression through a series of stages
digress	to stray from the main topic in writing or speaking to turn aside
aggression	the act or habit of launching attacks invasions hostile action or behavior
pedigree	a line or list of ancestors a recorded line of descent showing pure breeding of animals
expedite	to help or hurry the process of something of business to preform business quickly
podiatry	the study and treatment of foot ailments
gradient	an incline or slope the rate of incline
impede	to hinder to block the way of

Please edit the textbox to reflect the words spoken in the audio clip.

`match impede to hinder to block the way of`

Example: `match titanic with ?? by great strength size or influence*`

Figure 6-6: An mTurk task for transcribing *Voice Scatter* utterances.

workers, yielding 30,000 transcripts created by 130 workers for a total cost of \$330.

Since we have 3 transcripts for each utterance, we must combine them somehow to form a gold-standard *mTurk-transcript*. We chose the majority transcript if there was exact agreement by at least two of the workers, and selected a transcript at random if all three workers disagreed. There was majority agreement on 86.7% of utterances.

To assess the reliability of transcripts obtained in this manner, two experts each performed the same transcription task on a 1,000-utterance subset of the mTurk-transcribed data. Inter-transcriber "Word Disagreement Rate" (WDR) was computed, given N transcripts from two transcribers A and B, as follows:

$$WDR = \left(\frac{\sum_{i=1}^{N} Sub_i + Del_i + Ins_i}{\sum_{i=1}^{N} \frac{1}{2}(length_{i,A} + length_{i,B})} \right)$$

WDR is simply a symmetric version of Word Error Rate, as the denominator is the sum of the average length of each pair of compared transcripts.

The inter-expert WDR was 4.69%. The WDRs between the mTurk-transcripts and each of the two experts were 5.55% and 5.67%. Thus, it seems reasonable to treat the mTurk-transcripts as a near-expert reference orthography. In addition, the average WDR produced by pairing the three sets of transcripts produced by mTurk workers was 12.3%, indicating that obtaining multiple transcripts of each utterance is helpful when using mTurk to procure a reference.

6.3.3 Filtering for Accurate Hypotheses

Because *Voice Scatter* players often read terms and definitions verbatim, a significant portion of the utterances ought to be recognized with no, or very few, errors. In this section, we explore the usefulness of three sources of information in identifying this subset of utterances, with our goal being to select a subset of the data that can be automatically transcribed with human-like accuracy. First, we consider the utility of speech recognition confidence scores, which provide a measure of uncertainty based on acoustic and lexical features. Second, we look at information from the game context associated with each utterance. Much like in *Voice Race*, speech recognition hypotheses which produce "hits" are unlikely to occur by chance. In this case, however, a "hit" occurs when a term is correctly matched to its definition. Third, we explore the importance of using a small vocabulary, strict grammar during recognition by comparing our results to those produced by a trigram trained on all flashcards appearing in the corpus.

Figure 6-7 explores the usefulness of each of these factors in identifying high-quality

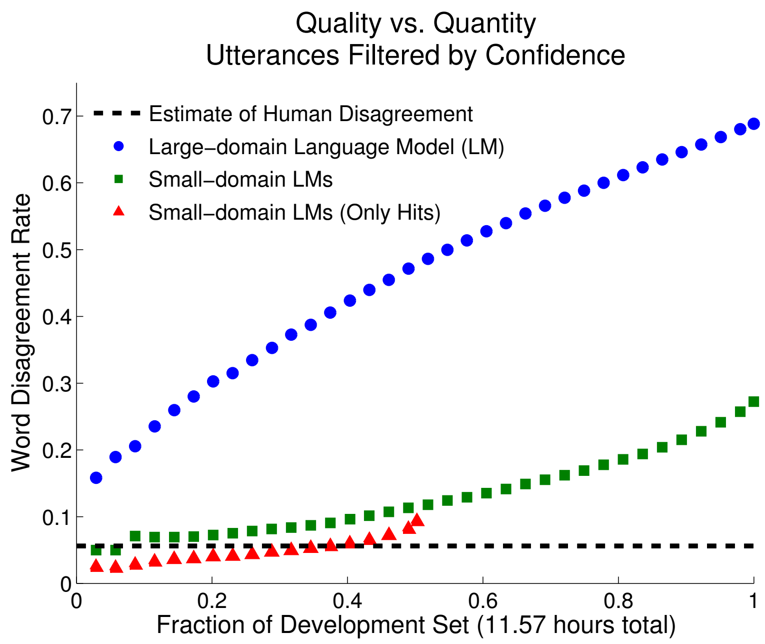


Figure 6-7: Cumulative Word Disagreement Rate (WDR) for recognition hypotheses produced using either a large domain trigram or many small-domain grammars on the 10,000 utterance mTurk-transcribed set. Cumulative subsets are created by incrementally adding hypotheses ordered by confidence score. An estimate of human WDR, calculated using the 1,000 utterance expert-transcribed subset, is shown for comparison.

subsets of the data. The curves shown are produced from three experiments performed on the 10,000 utterance mTurk-transcribed development set. First, we ordered the set of hypotheses logged from game play based on their confidence scores, as produced by the module described in [60]. We then drew utterances from the set in order from high to low confidence, and measured their cumulative Word Disagreement Rate (WDR) to produce the curve indicated with green squares. Second, we performed the same experiment, using only the 4,574 utterances which were identified as "hits" according to their recognition hypotheses. This produced the curve of red triangles. Third, to explore the effect of vocabulary and language model size, we trained a trigram on all flashcard terms and definitions which appeared in the corpus. Using this N-gram as the language model, we re-recognized each utterance to produce a new hypothesis and confidence score. We then drew hypotheses from these results in order of confidence score, to create the curve of blue circles. Finally, the dotted line shows the average WDR between the mTurk-transcripts and each expert on the 1,000 utterance expert-transcribed subset. It represents an expectation of human transcription agreement on the set.

First and foremost, it is clear from Figure 6-7 that the small-domain nature of our recognition tasks is essential. The N-gram language model had an overall WDR of 68.8% when

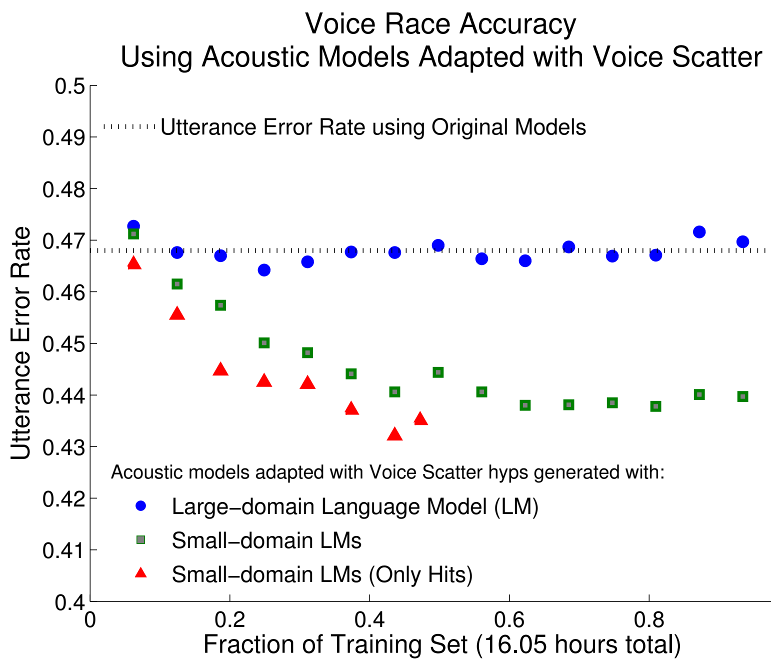


Figure 6-8: *Voice Race* utterance error rate using an acoustic model trained with incrementally more self-transcribed *Voice Scatter* utterances (sorted by confidence). The self-transcripts are generated using the original acoustic model via: a “Large-domain” n-gram, the small-domain grammars used in the online system, and the “hits” found in hypotheses generated from these small-domain grammars.

compared to the mTurk-transcripts on all 10,000 utterances, whereas the narrow domain LMs achieved a WDR of 27.2%. Moreover, using only confidence scores, it is possible to select a subset containing 15% of the original data with a near-human WDR of 7.0%.

Finally, by considering only “hits”, it is possible to select a subset containing 39% of the data at a human-like WDR of 5.6% by discarding just 78 minutes of low-confidence “hits”. Indeed, ignoring confidence scores altogether, and simply choosing all “hits”, yields 50.2% of the data at a WDR of 9.3%. It is worth noting, however, that on these filtered subsets, human transcripts are still likely to be better. For example, the average WDR between experts and the mTurk-transcripts on the 511 expert-transcribed “hits” was only 3.67%.

Self-Supervised Acoustic Model Adaptation

Here we explore using the self-transcribed *Voice Scatter* sub-corpora in the common task of acoustic model adaptation. We adapt the original acoustic model, used by both *Voice Scatter* and *Voice Race*. To show that these transcriptions are useful across domains, we explore how the quantity and quality of orthographically transcribed *Voice Scatter* data influences the effectiveness of the adapted acoustic model on the *Voice Race* recognition task.

We drew self-transcribed utterances from the 16.05 hours of data that were *not* transcribed by mTurk workers, so that we can analyze the usefulness of these transcribed data as a development set. Utterances and their self-"transcripts" were accumulated in one hour increments using each of the three filtering methods described above. After each new hour of data was added to the set, acoustic model MAP adaptation was performed using forced alignments of the self-transcripts. Each adapted acoustic model was then used by the speech recognizer to produce hypotheses for 10,000 mTurk-labeled utterances collected from *Voice Race*.

Figure 6-8 shows the utterance error rate found on the the mTurk-labeled *Voice Race* data using successively larger sets of *Voice Scatter* utterances filtered via the three methods for adaptation. First, it is clear that using errorful hypotheses produced by the N-gram language model does not result in an improvement in utterance error rate, regardless of the amount of training data used. Second, using high-confidence hypotheses of utterances recognized with a small-domain language model achieves significant gains, and appears to reach a local minimum when between 60% and 80% of the *Voice Scatter* data are used. Third, when only "hits" are used, error rates fall faster, and achieve a better local minimum, even though less than half of the total data are available.

Finally, by comparing Figures 6-7 and 6-8, we can see that the manually transcribed utterances serve as a useful development set, both to select a filtering method and set a confidence threshold at which to consider data self-transcribed. According to the development set, selecting the high-confidence "hit" data that comprises roughly 39% of the total corpus should yield a human-like WDR. Choosing a training set from utterances based on this operating point would achieve an utterance error rate in Voice Race quite close to the best local minimum shown in Figure 6-8. Moreover, in the absence of a development set, a 7.8% relative reduction in utterance error rate would have been attained simply by using all of the "hit" data.

6.4 Summary

This chapter has presented two online educational games that use speech recognition constrained by many small-domain language models to collect a rich variety of automatically orthographically transcribed continuous speech utterances. We have provided techniques that automatically identify and transcribe subsets of our data and shown that these subsets perform well as training corpora for acoustic model adaptation.

The *Quizlet* games provide a compelling example of how one can leverage WAMI to create speech-enabled websites of interest to the general public. By tapping into an existing user-base, we have shown how large amounts of speech data can be collected and transcribed at virtually no cost. In less than one month we have used these games to collect 85,938 utterances consisting of 46.33 hours of speech.

It is not difficult to imagine a wide variety of games, educational or otherwise, which fit the model exemplified by *Voice Race* and *Voice Scatter*. Unlike traditional GWAPs, which at times require somewhat contrived game-constraints to produce a label, small-domain speech recognition games may naturally fit into certain popular web sites. Educational games are particularly compelling, because they offer a situation in which players may be satisfied to choose among a small set of answers, the correct one of which is known to the computer. Such small domains help ensure accurate speech recognition and provide the opportunity to identify subsets of self-transcribed utterances.

Although this chapter has departed somewhat from the crowdsourcing approaches applied in the preceding pages, we note that the techniques explored here might be applied to the lexicon or language model components of the recognizer as well. For example, an interesting experiment that we leave for future work might be to learn new entries for the lexicon using a pronunciation mixture model trained with self-transcribed speech.

Chapter 7

Conclusions

This thesis has presented work on the application of crowd-supervised training techniques to a wide array of spoken language interfaces: photos, movies, and educational games. We prefaced this work with experiments using spoken addresses and air-travel dialogue sessions, which demonstrated that crowdsourcing techniques, combined with the WAMI toolkit, can be used to collect large quantities of speech-related data at very little cost. We then showed that it is possible to move beyond the typical batch-collection crowdsourcing strategies. We provided two examples – one in the movie domain, and the other in the photo domain – of organic spoken language systems that made use of a human computation paradigm to improve recognition performance on-the-fly. These experiments focused on the language model and the lexicon. Our acoustic model experiments demonstrated that crowdsourcing need not rely on monetary incentives to producing useful data. Using domain-specific context in educational games, we showed the benefits of *self-supervised* retraining.

7.1 Summary

In Chapter 1, we motivated our work with a vision of organic spoken language systems that grow and adapt on-the-fly. The subsequent chapters presented a suite of experiments that demonstrated the feasibility of building such crowd-supervised systems cheaply.

Chapter 2 provided the background material and related research relevant to this thesis. We began with a general discussion of crowdsourcing. At its simplest, we note that researchers use crowdsourcing platforms such as Amazon Mechanical Turk as a means of procuring or annotating data. We described other forms of collection, such as games-with-a-purpose, which can label data in certain domains for free. We then summarized decades of speech corpus collection, showed that it is typically a costly endeavor, and introduced the WAMI Toolkit as a means of collecting data over the web. Finally, we gave a short overview of three stochastic models in automatic speech recognition: the language model, the lexicon, and the acoustic models.

Chapter 3 demonstrated the utility of mTurk for collecting both prompted speech and spontaneous speech for a spoken dialogue system. The prompted speech task was designed to collect 100,000 spoken addresses at a cost of 1¢ each. Experimenting with these data, we explored the possibility of using a recognizer to filter out noisy utterances. We then turned our attention to collecting data for a spoken dialogue system in the flight reservation domain. We collected over 1,000 dialogues at two different price points and showed that, by crowdsourcing the data's transcription, we could use mTurk to evaluate a spoken language interface.

Chapter 4 examines crowd-supervised language model construction. This chapter provided the first example of an organic spoken language system for a photo annotation and search domain. We collected around 1,000 utterances through mTurk in two experiments, one with public photos and another with private photos. While the system was deployed, it automatically sent utterances back out to the crowd to be transcribed. The transcriptions were then compiled into an organically growing language model. We showed that search queries made to the systems improved over the two day period for which each system was deployed.

Chapter 5 examines crowd-supervised lexicon learning. We introduce the pronunciation mixture model as an effective strategy of turning the lexicon into a stochastic component of a speech recognizer. Using this framework we presented some initial experiments on the *PhoneBook* corpus of isolated-word speech. In particular, we drew the reader's attention to the crowdsourcing results where we showed that noisy utterances do not significantly degrade the performance of the PMM. Next we wrapped the PMM, together with a more traditional L2S approach, into another organic spoken language system – this time, in a cinema voice-search domain. The crowd-supervised retraining we employed for this system learned both the class N-gram language model *and* individual lexical entries. Using mTurk, we ran a live crowd-supervised experiment and also collected a separate 1,000 utterance test set. We showed that the recognition accuracy as well as the results returned by the system to movie queries made by the workers steadily improved over the course of the experiment. Finally, we performed an in-depth analysis of the recognition errors, showing the utility of the PMM.

Chapter 6 examines crowd-supervised acoustic model adaptation. In this case, the training is implicitly supervised by the crowd using domain-specific context, earning this technique the label of *self*-supervised acoustic model adaptation. The domains we explored were two speech-enabled educational games. Rather than use mTurk, for these experiments we were fortunate to have access to the large pre-existing user-base of *Quizlet*. Between the two games, we collected around 86,000 utterances over the course of 22 days. We presented a method that, using game context, automatically identified and transcribed over one-third of these utterances with human-level accuracy. We then showed that these data could be used for acoustic model adaptation. When compared with confidence-based approaches, our self-supervised approach was able to achieve better recognition accuracy with far less data.

This thesis has a set of appendices that detail the technical work that was required to enable speech to be captured from an ordinary web browser. Appendix A describes the current state of audio on the web, in technologies such as Java and Flash, and points to HTML5 specifications that hint at a future in which audio streaming is far easier than it is today. Appendix B walks the reader through setting up a Flash-based client-side audio recorder in a web page. Appendix C details the possible server-side configurations that might be used to collect audio from the client. Finally, appendix D is a tutorial that guides the reader through an entire audio collection task on Amazon Mechanical Turk.

7.2 Contributions

This thesis represents a significant step towards the creation of truly organic spoken language systems. In this section, we outline the contributions of this work.

First, none of the experiments in Chapters 3-6 would have been feasible without the WAMI Toolkit. This thesis required some significant upgrades to the capabilities already present in the toolkit. We added the capability of handling corpus-based language models as well as a mechanism to grow them organically. Moreover, the original Java applet was replaced with Flash, a more wide-adopted browser plugin. With WAMI, thousands of individual users were able to access our speech-enabled web sites. We have open-sourced this toolkit so that other researchers may do the same.

One significant finding of this thesis is that the lexicon can be robustly learned from the crowd using a pronunciation mixture model. This model makes use of example utterances to learn a weighted list of possible pronunciations. These weights can then be used to build a stochastic lexicon. We showed that this model was resilient to the noisy characteristics of crowdsourced data. This was demonstrated using a corpus of crowdsourced isolated-word speech as well as in the context of a spoken language system.

This thesis contains many firsts with respect to crowdsourcing speech-related tasks. To the best of our knowledge, our experiments in Chapter 6 were the first in academia to make use of mTurk for a transcription task. The speech-collection tasks in Chapter 3 were also unprecedented. While many have followed in our footsteps with respect to transcription, fewer have attempted to use mTurk for speech collection. This is likely due to the technical challenges, and our hope is that WAMI helps in this regard. Through crowdsourcing using the WAMI Toolkit, roughly 170 hours of audio was collected for this thesis.

We showed that these crowdsourced data are useful in retraining the three stochastic components of a speech recognizer. Moreover, we demonstrated a human computation framework for crowd-supervised spoken language systems. Chapters 4 and 5 go beyond typical crowd-sourcing for data-collection, and develop prototype organic speech interfaces. To accomplish this task, we developed a set of strategies for dealing with noise, a large obstacle to the hands-free retraining of these systems.

Finally, this thesis has proposed a mechanism for moving beyond the micropayment systems for speech collection. We have shown that there can be a fruitful marriage between education and speech research. Students can simultaneously provide researchers with interesting data and benefit from studying with a natural language interface. While we could have simply speech-enabled rote memorization exercises, we found that games provided interesting domain-specific contexts, which we used as mechanisms to transcribe some of the incoming speech.

These contributions provide a firm ground upon which to conduct further research. In the next two sections, we discuss both long term avenues of exploration as well as more immediate experiments that could be performed to further this work.

7.3 Discussion

As we move further into an age of constant connectivity, systems are increasingly able to take advantage of an ever-ready crowd. Whether the individual members of the crowd come from the system's internal user-base or through external sources such as Amazon Mechanical Turk, they often enable interactions that were previously impossible. This thesis has focused on the crowd-supervision of spoken language systems, which removes the retraining responsibilities from the hands of experts and relies on a scripted but symbiotic relationship between the system and the crowd. In this section, we speculate on the types of systems we might create with this technology as these research areas mature.

The experiments in this thesis suggest that a suite of tools for crowd-supervised spoken language systems have the potential to enable new types of speech interfaces. For example, the crowd-based transcription that we introduced in Chapter 3 and expanded upon in Chapters 4-6 can give a system the ability to keep track of its own word error rate. This raises the question of what the best practices are for developing speech interfaces that collect their own test sets. The case of crowd-supervised systems that are constantly improving complicate the matter, since it is well known that user behavior can change across different error conditions [130]. It may be best, for example, to throw away old data as a system matures.

Another solution to the problem of high word error rates in a fledgling spoken language system is to use the crowd to act as the system itself. Chapter 2 introduced a suite of technologies for building what some have called *crowd-powered* systems [10]. Such systems do not explicitly incorporate artificial intelligence, but instead build on the APIs of micropayment platforms, such as mTurk to perform tasks which have not yet been achieved entirely algorithmically. VizWiz, for example, employed a crowd for a speech and image recognition task to help blind individuals ask questions about their surroundings [14]. When an out-of-the-box speech recognizer was applied to the task, it was found to be unreliable. Instead, workers listened to the original audio to answer the questions posed.

One can imagine systems, however, that employ both *crowd powered* and *crowd-supervised* interaction paradigms. Suppose that the speech recognizer's confidence scores were made available to VizWiz. In such a scenario, it is conceivable that, when the recognizer is not confident about its transcription of an utterance, the audio could be sent to the crowd for transcription. Similarly, optical character recognition might be performed on images to pick out salient text. Such systems, augmented with confidence capabilities, might decide when to back off to a crowd-powered framework and when to rely on the built-in artificial intelligence. Perhaps the most satisfying detail of such an arrangement is that the responses from the crowd regarding low-confidence data can not only ensure that an accurate answer is given to the user, but can also be used as training data to improve the system in a crowd-supervised fashion.

Crowdsourcing platforms could facilitate such systems by exposing additional functionality through their APIs. There may be a need, for example, for real-time responses from the crowd. One strategy is to pay workers to wait in retainer pools where they remain constantly on call [11]. When a task is ready they are notified and asked to begin work immediately. Using the retainer strategy response times can be kept to just a few seconds. Were each requester to independently create retainer pools, however, some workers would inevitably sign up for multiple pools at a time. Since this would likely increase the average response time, it is possible that retainer pools would be better implemented at the platform level. As of yet, no crowdsourcing platform has done so.

Audio-related features comprise another set of functionality that could be incorporated into crowdsourcing platforms to ease the development of speech-related tasks. Storing audio data without having to set up a server is currently not possible, preventing those without computer experience from using mTurk as an audio source. Even for those capable of installing or creating the components necessary for client-server communication, the process of streaming audio over the web in either direction is not simple. In Appendix C, we describe an audio collection server that relies on the hosting services of the Google App Engine. Audio playback is also lacking to some degree; however, this is the responsibility of the browser vendors and will likely be corrected in the coming years with HTML5. The appendices of this thesis describe in detail the current state of web-based audio collection technology.

7.4 Future Work

While the previous section described a vision of the future of crowd-supervised systems, this section describes more immediate efforts that might be made to extend the experiments found in this thesis. Those of Chapter 3, for example, identify mTurk as a valuable platform for speech research. One of the inherent difficulties with spoken language systems research is that it is hard to compare systems, even those within the same domain. It might be interesting, however, to use mTurk as a platform for spoken dialogue system evaluation. One could imagine constructing a set of guidelines, or even an entire web-based front-end, that multiple systems in a common domain would be required to use in order to enable large-scale rigorous assessment, much like the former NIST-DARPA evaluations in the air travel domain. Before this could be achieved, however, the community would need to explore the question of how best to supplement crowd-collected dialogue system corpora with annotations, such as user satisfaction and dialogue acts [58]. If a cloud-based evaluation framework could be devised, the management overhead of an evaluation endeavor would be greatly reduced, and a potentially unlimited number of institutions could participate.

Chapters 4 through 6 examined three stochastic models for speech recognition. There are a number of ways in which we might extend our experiments with these models to further the goal of creating an organic language system. For language models, one might try to reduce their perplexity by modeling hidden topics on-the-fly using techniques such as Latent Dirichlet Analysis. In a personal photo search domain, for example, it might be reasonable to treat a set of photos and their annotations as a document, and perform dynamic language model adaptation using variational Bayes inference as in [134]. Alternatively, language model caching techniques might be applied to domains such as *Movie Browser*, where words related to trending movies would be highly weighted and then decay exponentially as they become outdated [69]

The lexicon also presents opportunities to improve organic systems. As described in

Chapter 5, when the lexicon is treated as another stochastic citizen of the speech recognizer, we are able to improve it on-the-fly using data-driven techniques. While the PMM was successful, the underlying graphone language model upon which it relies is trained on a large lexicon of expertly crafted pronunciations. One useful line of research would be to try to remove this dependency, perhaps with the assistance of automatically learned acoustic units [148]. Were such techniques developed, expert-quality lexicons could be learned for low-resource languages. Following the work of [15], an approach that could effectively connect the language and acoustic models without the aid of an expert might also assist with long-standing problems with out-of-vocabulary words.

The acoustic modeling work in Chapter 6 might be extended in a number of ways. Beyond simple adaptation, one can envision systems that play an active role in their own learning, perhaps by choosing the data they wish to have transcribed. It is not hard to imagine how the fields of active learning and crowdsourcing could be combined in a number of areas. One example of such a combination is global entropy reduction maximization (GERM) [150]. Whereas in our work we provide an domain-specific mechanism for which the system self-selects a set of utterances that are transcribed for free, GERM proposes a domain-independent approximation that reduces the number of utterances that need to be transcribed in order to achieve the same recognition accuracy. It might be interesting to combine these approaches.

More generally, it would seem that spoken language systems can play a large role in selecting or identifying data useful for their own training. While in some domains, relying on micropayment platforms will be necessary for transcription, in others – especially in education – utterances can be self transcribed. Other examples might include adapting acoustic models for children using simple math or word games. Models for non-native speech might be constructed from a language-learning game where learners perform small-domain tasks in the target language. Moreover, utterances recorded from the game could be made available to a teacher, who might provide feedback to the student by correcting pronunciation errors. A byproduct of this entire process would be a teacher-labeled corpus of non-native speech, which might be directly used in algorithms that emulate a teacher's pronunciation corrections.

Beyond education, user actions might be indicators of self-transcription in other domains as well. Suppose *Flight Browser* is showing the user two flights departing at two different times. If the user speaks and the recognition result indicates one of the two times showing, part of the recognition result is most likely correct since a misrecognition resulting in a contextually-correct flight-time is unlikely. Clearly one must be careful when using this technique in combination with a previous suggestion like dynamic language models, where the constraints might be such that these times have higher likelihoods. Another downside to this approach is that it is inherently domain specific. It may be interesting to extend GERM to identify domain-specific context that might be used for *self*-transcription.

The work presented here has addressed the issue of using crowd-supervised training techniques to improve the automatic speech recognizer. There are, however, other components of spoken language systems which might benefit from this approach. Some believe that all components of a spoken dialogue system might one day be stochastic [149]. Examples in our work that might have benefitted from crowd-supervised training are the conditional random field in the *Movie Browser* domain [93] and the dialogue manager of *Flight Browser* [147].

This thesis has laid the groundwork for others wishing to experiment with crowd-supervised training for spoken language systems. We believe some systems do not yet tap the power of the very crowds that use them. For other systems that need to ask arbitrary questions of the outside world, platforms such as mTurk give them a voice. Through APIs for humans, these systems can now query humans for answers to questions like "What is spoken in this audio clip?" or "How do you pronounce *blicket*?" Looking forward, it is clear that an important part of building these *human-in-the-loop* applications will be ensuring that they ask the right questions to cost-effectively improve their performance.

Appendix A

Web-based Audio Capture Technology

As previously described, one clear advantage of collecting audio through a website rather than over the phone is that the collection task can be supplemented with visual information. To elicit read speech, static prompts can be formatted and displayed using HTML. Basic interactivity, such as the ability to click a button to skip a prompt, can be performed using JavaScript. Web 2.0 technologies, such as Asynchronous JavaScript and XML (AJAX), allow a website to communicate with a server to integrate a database or code that would otherwise be difficult to implement in the browser.

While most of the aforementioned technologies are relatively mature, it is surprisingly difficult to implement a web-based framework to enable the transmission of speech captured from a microphone to a remote server. This is because browsers currently do not offer native support for the microphone. While mobile browsers do not even support a plug-in architecture that allows third-party developers to incorporate native code, desktop browsers can be extended to allow for microphone access. Developing a plug-in, while perhaps the easiest way to control the look-and-feel of the interaction, requires the user to download and install code that they may be hesitant to trust. Fortunately, there are a number of plug-ins (e.g. Silverlight, Java, and Flash) developed by reputable third-parties that make microphone access possible and which the user is likely to already have installed. Each, however, comes with its own set of caveats.

At first glance, it would seem that the server-side technology should offer greater freedom of choice. After all, the collection of audio need only be supported by a single server whereas on the client-side, a multitude of browser configurations must be supported for transmission. Unfortunately, there are times when the choice of client-side technology dictates that of the the server-side. Adobe Flash, for instance, can stream Speex encoded audio from any of the popular desktop browsers. Currently, however, this comes at the high cost of requiring the installation of Adobe's proprietary Flash Media Server. This dilemma comes with its own workarounds, again with their own costs.

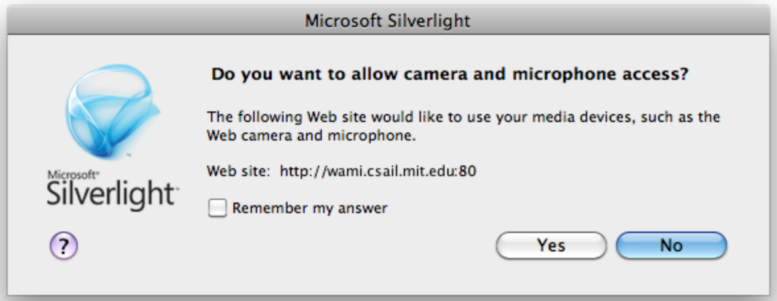
In the remainder of this section, we will delineate the current state of technology regarding audio collection from the web. We begin with the three desktop browser plug-ins that currently enjoy the greatest global market penetration: Silverlight, Java, and Flash. In an ideal world, none of these plug-ins would be required, and browser vendors themselves would implement a standard for microphone access, and perhaps even audio streaming. Thus, we next discuss the current state of the HTML5 specification with respect to these developments, but ultimately, we come to the conclusion that native microphone support in all of the major browsers is unlikely in the near future. Since the plug-in solution does not work for most smartphones, we end this section with a work-around involving the development of native apps.

A.1 Silverlight

Microsoft Silverlight was first released in 2007 as a means of creating rich internet applications using a subset of the .NET framework. There are official plug-ins for all of the major browsers on both Windows and Macs. There is even an open-source version available for Linux, although there is no guarantee that it contains the features necessary for audio collection given that microphone support itself was only recently added to Silverlight in its fourth release in 2010.

Microsoft claims that over 60% of the browsers accessing the web today have Silverlight installed. This may be due to the success of Netflix, which uses Silverlight to stream movies to consumers over the web, in addition to its movies-by-mail service. It is also quite likely that this statistic varies by country. Silverlight currently does not have quite the name recognition of Adobe's Flash framework, but, from the user's perspective, it is just as easy to install and provides many of the same capabilities. Note that, while it was at one point supported on Windows smartphones, Silverlight's future in the mobile arena is far from certain.

In addition to providing microphone access, Silverlight can communicate with the containing web-page via JavaScript, or to a remote server using HTTP requests. HTTP stands for Hypertext Transfer Protocol, and serves as the web's standard request/response protocol. An HTTP POST in particular can be a useful way to transmit audio data across the wire to an HTTP server. A user interface to control starting or stopping the recording process can be created within the Silverlight application itself, or using JavaScript and communicating the relevant actions directly to Silverlight. Regardless of the look and feel of the rest of the interface, microphone access comes with one aspect that third-party developers have no control over. The alert message below pops up the first time a website requests the microphone through Silverlight:


Microsoft Silverlight

Do you want to allow camera and microphone access?

The following Web site would like to use your media devices, such as the Web camera and microphone.

Web site: http://wami.csail.mit.edu:80

Remember my answer

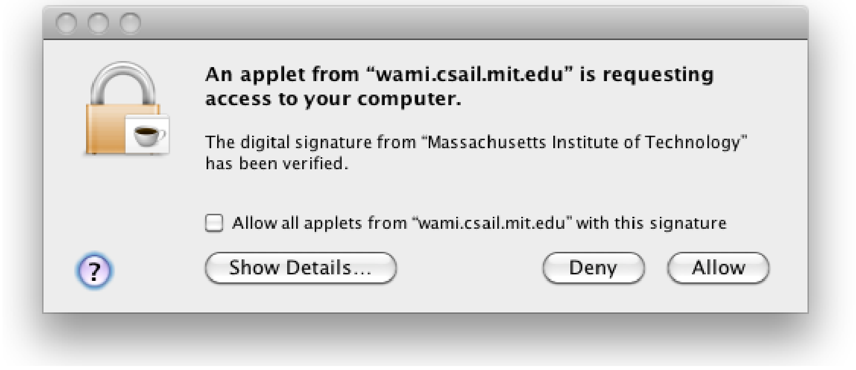
Silverlight application development is only well-supported on Windows machines for which Microsoft's development tools are available. The minimum set of tools necessary to develop a Silverlight application are free, but support is greatest in Microsoft's premium integrated development environment, Visual Studio. For other platforms, ad hoc solutions do exist, including an open source .NET development framework called Mono, which appears to have earned a sizable following. Such solutions, however, inevitably lag behind the officially supported software development kit (SDK).

A.2 Java

Java Applet technology has been around since the mid to late nineties, when applets quickly became the most popular way to add dynamic content to a web site. It was not long before the larger websites of the day were incorporating Java in high-traffic applications. For example, most of the original games on *Yahoo!* were originally implemented in applet-form and executed in the Java Runtime Environment (JRE) via the Java browser plug-in.

Even today Java has retained a surprisingly large portion of the plug-in market share. Currently Java's coverage on desktop browsers is on par with, if not slightly better than, that of Silverlight. Although somewhat more cumbersome for the user to install and operate, once up and running, a Java applet provides the microphone access necessary for recording-related tasks. Indeed, Java has been explored by a number of researchers as a means of collecting speech from the web, [86, 52].

Java runs in a sandboxed environment, meaning that it does not have access to certain system functionality by default. Of interest to us is that the microphone falls under these special security considerations, requiring us to take additional action. Just as in Silverlight, the solution is to present the user with a pop-up message that requests permission to access this feature. Unfortunately, as can be seen in the dialog below, there is no way to specify that only the microphone is of interest, and the message can thus appear ominous to some users.



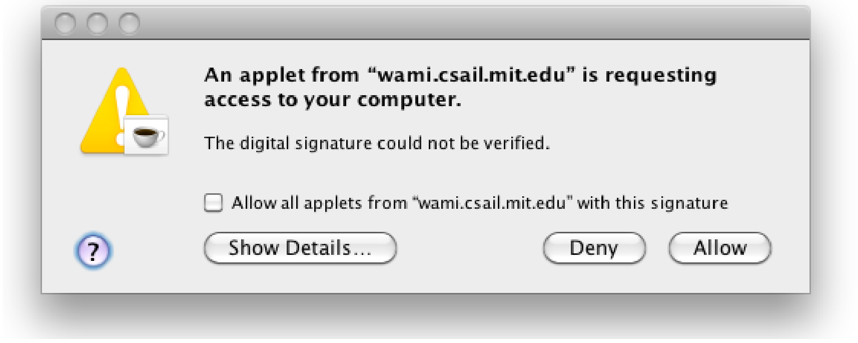
An applet from "wami.csail.mit.edu" is requesting access to your computer.

The digital signature from "Massachusetts Institute of Technology" has been verified.

☐ Allow all applets from "wami.csail.mit.edu" with this signature

To be more precise, even presenting the security warning above is not quite so simple. Applets requesting access to features outside the restricted sandbox are required to *sign* their code. Typically this involves sending a certificate signing request (CSR) to a certificate authority such as VeriSign. For between $500 to $1,000, the certificate authority will then return a public key which can be incorporated into the Java Archive (JAR) file which contains the applet. When the applet is accessed by a user, the certificate authority's servers are contacted to verify the identity of the code's signer.

Understandably, the signing process may be prohibitively expensive to some. Fortunately, there is a free work-around which comes at the cost of a slightly more daunting security message. In particular, it is also possible to *self-sign* an applet. In this scenario, no third-party is required to generate a pop-up message asking the user for microphone permissions. The downside, however, is that, with self-signing, the already ominous message becomes even more daunting.



An applet from "wami.csail.mit.edu" is requesting access to your computer.

The digital signature could not be verified.

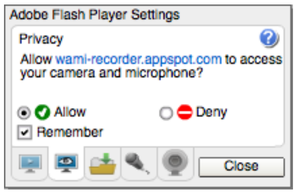
☐ Allow all applets from "wami.csail.mit.edu" with this signature

Aside from the aforementioned code-signing issue, Java is arguably the most developer-friendly of the plug-ins we explore here. The Java SDK can be installed on any operating system, and advanced developer tools are both free and cross-platform. The Eclipse integrated development environment (IDE), for instance, offers on-the-fly compilation and full-featured development tools for Java. With the application of a Java-based server-side technology, such as Apache Tomcat, most of the development for a recording application can be written in a single language.

A.3 Flash

Through much of the last decade, rich internet applications transitioned away from the heavy-weight Java applet architecture, and (where JavaScript alone would not suffice) settled on Adobe's relatively light-weight Flash solution. Currently, Flash can be found on the web in many forms including games, video players, and advertisements. Some popular sites, most notably YouTube, have begun making the switch to HTML5; however, others, including Hulu, have continued to rely on Flash.

Flash enjoys the largest market of desktop browser plugins, with over 95% coverage across the most popular browsers. In fact, Google's Chrome browser comes with Flash pre-installed. For this reason, Flash is arguably the audio collection method that offers the fewest impediments from the user's perspective. As with the Java and Silverlight solutions, Flash can communicate with JavaScript, and can even be hidden from view, allowing the web site to appear plug-in free, save for the initial microphone permissions security panel. Even the microphone permission settings are relatively unobtrusive.


Adobe Flash Player Settings
Privacy
Allow wami-recorder.appspot.com to access your camera and microphone?
[✓ Allow] Deny
[✓ Remember]

Unlike Java or Silverlight, the security settings dialog box for Flash appears embedded directly in the web page. While this is convenient for the client, this can make development difficult unless the width and height of the Flash content is kept at least as large as the 214x137 pixel security window. Resizing or moving the Flash object will work in some browsers, but not in others, where such actions seem to reload the content, losing the security settings. One solution to these issues is to force the user to check the *Remember* box, which ensures that microphone permissions are retained through page refreshes or the resizing of the Flash content.

Once microphone permissions are obtained, getting the audio from the browser to a server is fraught with additional difficulties. Flash was designed to stream content using the Real Time Messaging Protocol (RTMP) to the proprietary Flash Media Server (FMS). With the purchase of an FMS license for around $1,000, one can stream Speex-encoded audio over the web using the Adobe-sanctioned method. For audio collection, perhaps this meets the requirements. For an interactive application, however, one still needs to figure out how to access the audio stream once it gets to the Flash Media Server, in addition to decoding it and passing it along for further processing.

Less expensive media server options do exist. Wowza Media Server is a Java-based server that can be used to stream content to a variety of devices. Audio collection, however, still requires that the client have access to the microphone. Although it has since expanded, Wowza was originally built to support RTMP, and can thus be used to handle Speex audio data collected through Flash. Licenses for Wowza are generally more flexible than those of Flash Media Server. There even exist daily or monthly licenses that can be used in conjunction with Amazon's Elastic Compute Cloud (EC2).

Finally, there is an open source media server called Red5 which offers a free alternative to Wowza or Adobe's FMS. Red5 is also built on Java and provides support for RTMP. While Red5 supports the functionality necessary for collecting audio, the support and documentation are likely to be lacking in some respects relative to those of the licensed media servers.

Regardless of pricing or licensing issues, media servers supporting RTMP are necessarily far more complicated than necessary for what ought to be a relatively simple task of transporting audio from client to server. One of the reasons is that RTMP was designed to transport audio in real time, dropping frames if necessary. Whereas applications like voice chat have strict real time constraints, many applications for audio do not. Certainly, collecting audio to be processed offline need not impose latency restrictions on incoming audio. Even a speech recognizer that is run on-the-fly is arguably easier to configure if one does not have to consider arbitrarily dropped frames from a protocol such as RTMP.

A simpler approach, albeit with the potential for a certain amount of latency, is to use an HTTP POST to submit audio data from the Flash client to a server. This technique eliminates the need for a media server altogether, allowing any HTTP compliant web server to collect the audio. Adobe made this possible in late 2008 with the release of Flash 10. Previous versions of Flash did not give direct access to audio samples on the client-side, forcing developers to open a `NetStream` that piped audio from the microphone directly to an RTMP compliant media server. Now, however, the samples can be buffered on the client-side and sent to a web server via a standard HTTP POST.

Later, we will describe the WAMI recorder, which is an open-source project that makes use of this technique to transport the audio. There do exist a few caveats to using the HTTP

POST approach, however, the largest of which is probably that Speex-encoded audio data has *not* yet been made available to the client-side code. In other words, ActionScript, the scripting language used to program Flash content, only has access to raw audio samples. While it is easy to wrap uncompressed audio into a container, such as WAV, there is not yet a client-side solution to compressing the audio using a codec such as Speex, unless one is willing to use RTMP. Other lossy compression techniques, such as the μ-law encoding often used in telecommunications, might be straightforward to implement in ActionScript, but only cut file sizes roughly in half.

Development in Flash is not as straightforward as in Java, but fortunately, it is not quite as platform-dependent as Silverlight. Adobe's full-fledged IDE, Flash Professional, costs a few hundred dollars, but is equipped with the functionality necessary to create complex animations as well as to write, compile, and debug ActionScript code. Fortunately, a free alternative exists for those of us primarily interested in application development rather than fancy animations.

Flex is a free open-source application framework also created by Adobe. Unlike Flash Professional, it is not particularly well suited to creating movies and animations, but suffices for compiling ActionScript and generating a SWF file, which can be embedded in a site as a Flash object. The Flex SDK is command-line driven; however, Adobe does release a modified version of Eclipse called Adobe Flash Builder, that provides all the amenities of a typical IDE. An education license for Adobe Flash Builder is free, but the general public must pay a hefty fee, and for this reason may prefer to stick with the command line tools.

A.4 HTML and JavaScript

Most computers have audio recording software pre-installed. Thus, one solution to collecting audio over the web is to require the user to figure out how to record the audio themselves, and then provide a means of uploading the audio through an HTML form. Clearly this greatly restricts the types of applications into which one can incorporate audio recording. Without a plug-in, however, there is currently no way to access the microphone on any of the major browsers. Still, the future of microphone access in the web browser lies in JavaScript and HTML itself. It is unclear, however, how far away this future lies.

The World Wide Web Consortium (W3C), founded and headed by Tim Berners-Lee in 1994, was created in part to guide the development of web standards such as HTML, and to ensure compatibility across browsers. A specification goes through a number of draft stages before being implemented, reviewed, and finally ratified in a W3C recommendation. It is within these documents that the fate of microphone access within HTML and JavaScript will be settled. Ultimately, however, it is up to the browser vendors to implement the specifications.

In early 2004, discontent with the W3C's handling of the HTML specification motivated the formation of the Web Hypertext Application Technology Working Group (WHATWG). Their standardization efforts have since been adopted back into the W3C's HTML working group and renamed HTML5. These two working groups continue to develop their mostly overlapping specifications in parallel. The WHATWG, however, has dropped the term HTML5 and now refers to its specification as a *living* standard for HTML, noting that pieces of the specification (e.g. the `<canvas>`, `<audio>`, and `<video>` tags) have already been implemented in a number of browsers. The W3C, on the other hand, still intends to issue an official HTML5 recommendation; however, recent official estimates put the release date of such a recommendation sometime in 2014.

Ideally, HTML or JavaScript provide a standard mechanism for accessing the microphone regardless of the particular browser, mobile or desktop, and this would entirely alleviate the need for plug-ins. Perhaps the standard would be developed along lines similar to the way that browsers are beginning to support the `<audio>` tag for *playing* audio. Standardization, however, is a lengthy, complicated process involving the often competing, rarely spoken motives of several interested parties, especially the well-known browser-vendors: Microsoft, Apple, Google, Mozilla, and Opera. Indeed, it is most often employees of these companies who serve as members of the aforementioned working groups.

To provide a concrete example of the difficulty in standards development, one need only look at the current working draft of the `<audio>` tag for playback. There is currently no single audio format supported across the five most popular browsers. MP3, for instance, is supported by the latest versions of Internet Explorer, Chrome, and Safari, but is not supported in Firefox and Opera, presumably for reasons having to do with the license. Surprisingly, uncompressed audio does not fare much better. The WAV format, a simple container for linear PCM audio data developed in part by Microsoft in the early nineties, is supported by four out of five major browser vendors. The lone hold-out is Microsoft Internet Explorer. Hopefully, as the working draft reaches maturity, at least one format will be supported by all of the major browsers.

Currently, the proposal on the table for microphone access, the `getUserMedia()` function, falls under the purview of the W3C WebRTC working group, whose stated mission is "to enable rich, high quality, Real-Time Communications (RTC) applications to be developed in the browser via simple Javascript APIs and HTML5." The project is supported by Google, Mozilla, and Opera, which thus represents buy-in from a majority of the browsers accessing the web today. In theory, this API would allow one to collect audio in

JavaScript and submit it to a server using a simple XMLHttpRequest (XHR), which would appear on the server side as an HTTP POST containing the audio data. To date, however, this portion of the specification is still in a very early draft, and has not been implemented within any officially released browsers. For the time being, it is clear that we will have to rely on plug-ins for our web-based audio recording needs.

Appendix B

Example: WAMI Recorder

In this section, we describe the WAMI recorder, an open-source Flash recording tool that is wrapped in a simple JavaScript API. The original Web-Accessible Multimodal Interface (WAMI) toolkit [52] served primarily as a means of transporting audio from the browser to a speech recognizer running remotely. Initially a Java technology, it has since been rewritten in Flash and the recorder itself has been placed into a small open-source project, making it easy to adapt it for a variety of uses. The source code for the WAMI recorder can be checked out from a Google Code repository found at the following URL:

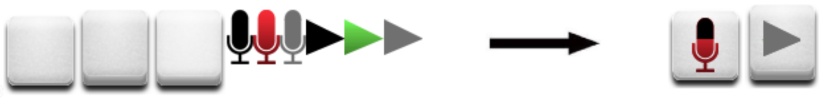
`https://wami-recorder.googlecode.com`

B.1 The JavaScript API

The simplest WAMI recorder example might consist of just three files. First, an HTML file would define a few buttons to start and stop recording. The HTML file would then include a JavaScript file to set up the recorder, which resides in the third SWF file. The JavaScript would also be responsible for linking the button clicks to actions in the recorder.

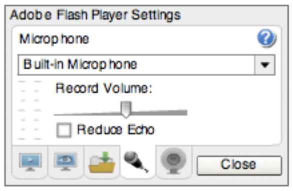
The example project we have created has a few more basic features. To ease the process of embedding the Flash into the web-page, the project relies on an external dependency called SWFObject. Fortunately, this code is just a single, well-maintained JavaScript file, which is reliably hosted by third parties, meaning it can be conveniently included by linking to those resources directly.

We also provide a somewhat more appealing graphical user interface (GUI) than can be created using HTML alone. With a single PNG image containing possible button backgrounds and foregrounds, one can duplicate and manipulate the master image in JavaScript to create attractive buttons that even boast a special effect. In particular, when audio is recorded through the microphone, a few JavaScript tricks can be used to display an audio level meter by adding some red to the microphone's silhouette. The same can be done in green for the playback button. When a button is disabled its silhouette becomes gray.



Before this GUI is put into place, however, we must ensure that the proper microphone permissions have been granted. To do this, we need to check whether WAMI already has the appropriate permissions, and, if not, show the privacy settings panel. The security settings are the only piece of the recorder for which the user must interact directly with the Flash itself. It may be worthwhile at this stage to encourage users to click the *Remember* button in the privacy panel in order to avoid going through this procedure every time the page is refreshed.

While granting permissions in the privacy panel is essential, other panels in the settings might be of interest as well. The microphone panel in particular can be useful for selecting an alternative input source, adjusting the record volume, or reducing echo. The following is a screenshot of this panel.



Once the microphone permissions have been granted, the application has full access to the core WAMI recording API though JavaScript. This API consists of six functions that lie in the `Wami` namespace and control actions related to both recording and playing audio:

```
Wami.startRecording(myRecordURL);
Wami.stopRecording();

Wami.startPlaying(myAudioURL);
Wami.stopPlaying();

Wami.getRecordingLevel();
Wami.getPlayingLevel();
```

WAMI's `startRecording` function will begin recording and prepares to perform an HTTP POST of the audio data to the URL provided. When `stopRecording` is called, recording ends and the data are sent. Note that this means audio is buffered on the client during recording, which may make long recordings or continuous listening unfeasible. In a later section, we will describe a work-around that we have implemented to simulate streaming, and list the inevitable caveats that come with trying to perform an action Flash was not designed to handle without RTMP.

The `startPlaying` function also takes a URL, but in this case the HTTP request made is a GET. The server is then expected to place a WAV file in the body of the response. This is precisely what a browser does when one types the URL of a WAV into the address bar and hits enter. The Flash downloads the WAV to the client and plays it back. The `stopPlaying` function can be used to stop the audio in the middle of playback.

The remainder of the API just supplements the recording and playing functions with a few additional features. The remaining two functions in the list above provide access to the microphone activity level. In our sample GUI, these two functions help animate the buttons when recording and playing so that the silhouettes in the buttons can act as audio meters. There are also a few optional parameters not shown above. The `startRecording` and `startPlaying` functions also accept arguments that specify callbacks that will fire when an action starts, stops, or if an error occurs. This can be useful, for instance, when an application needs to play two short audio files in a row, or simply change the visual content of the page after audio playing has stopped.

B.2 Audio Formats

Audio processing vocabulary (e.g. "format", "encoding", "container") tends to cause a fair amount of confusion. An *encoding* refers to the way the bits represent the audio itself, whereas the *container* is a wrapper for the audio bits that specifies meta-information. Confusing everything is the word *format*, which can be used to refer to the container, the encoding, or both together.

An audio container is usually capable of holding more than one audio encoding. Encodings can be lossless, in which case the original high-fidelity audio data can be recovered, or lossy, which generally discards some of the information in return for a better compression. To be completely clear what audio format one is referring to, it's best to specify both a container and an encoding.

The primary container used in the WAMI recorder is WAV. A second, less well-known, container that we support is called AU. Both can be used to store uncompressed audio in the linear pulse code modulation (PCM) representation, as well as other encodings such as G.711. Released in 1972, G.711 is a standard for audio companding commonly used in telecommunications. In North America, μ-law encoding is used to allow a sample to be encoded in 8-bits rather than 16, resulting in lossy compression. Although the μ-law algorithm would not be difficult to implement in ActionScript, WAMI currently leaves the PCM samples unaltered.

Uncompressed PCM audio generates files that are larger than one might wish, making bandwidth consumption a limitation of this recorder. Lossy encodings generally solve problems with network constraints; however, as mentioned previously, Flash currently makes compression in ActionScript difficult for all but the most basic algorithms. MP3 encoding, for example, is a computationally intensive task that would likely require the ability to run native code, assuming that the licensing constraints were not a problem. Speex, on the other hand is an open-source codec that is well-suited to the task of the compression of voice data. Often found in the OGG container format, Speex is also supported in Flash using the FLV container. Unfortunately, Speex has only been made available for recording to a flash media server, and currently cannot be used to send audio via a POST request.

Adobe has given developers a preview release of an architecture called Alchemy that may offer a solution to the encoding problem. Alchemy allows developers to compile C and C++ code targeted to run in the sandboxed ActionScript virtual machine. It would be feasible, then, to port an efficient audio encoder using Alchemy to be efficiently executed on the client-side. Alchemy has been released as a preview, but may not be officially supported until late in 2012.

With these caveats in mind, there is still a way to control the file size of the audio data in the WAMI recorder by manipulating the sample rate used. The following sample rates are supported by Flash's `flash.media.Microphone` object: 8kHz, 11.025kHz, 16kHz, 22.05kHz, and 44.1kHz. We recommend a minimum of 16-bits per sample, unless a lossy compression algorithm such as μ-law is employed.

Playing the audio back to the speaker is often a requirement of an audio collection application, if only to let the speaker know that their microphone configuration is working. Unfortunately, for playback, Flash only accepts uncompressed audio at 44.1kHz, making resampling necessary for the lower rates. Resampling a rate that divides evenly into 44.1kHz can be approximated easily in ActionScript, but resampling 8kHz and 16kHz sound is more problematic. With additional server-side code the resampling task can be performed remotely. Alternatively, the playback task could be left to the browser using HTML5's `<audio>` tag. Recall that WAV playback is currently supported by all the major browsers except for Internet Explorer.

Appendix C

Example: The WAMI Server

One of the most appealing features of the WAMI project is that it can be used with a variety of server-side technology. We have experimented with implementations that use Java in Apache Tomcat, PHP, pure python as well as python within the Google App Engine. In this section, we describe two of these configurations in detail. First, we show a simple PHP script that can run from any PHP-capable web server. Since there are countless ways to set up such a server, we leave the general configuration to the reader. For those who have no experience with setting up a server whatsoever, we provide a second example that makes use of the Google App Engine. In this example, the server-side configuration is standardized by Google, and friendly tools are available for debugging and deploying the web service. Finally, we conclude this section with some additional considerations that may be important for some non-standard use-cases of WAMI.

C.1 PHP Script

As a simple case-study of how one might use WAMI in practice, we will take a look at a simple PHP implementation of the necessary server-side code. Suppose that we have a file called `collect.php` with the following contents:

```php
<?php
parse_str($_SERVER['QUERY_STRING'], $params);
$name = isset($params['name']) ? $params['name'] : 'output.wav';
$content = file_get_contents('php://input');
$fh = fopen($name, 'w') or die("can't open file");
fwrite($fh, $content);
fclose($fh);
?>
```

This six-line PHP script is enough to implement a basic back-end for WAMI. PHP is a server-side scripting language typically used to generate dynamic web pages. To allow the WAMI recorder to POST data to the script, it must be made available at a particular URL using a PHP-enabled web-server. Apache is one such server that can be outfitted with PHP. Assuming such a web-server was running on port `80` (the port used most often for HTTP requests), and that a file called `collect.php` was hosted at the root, the following line of code in JavaScript would initiate recording to it:

```
recorder.startRecording('http://localhost/collect.php');
```

Assuming that the proper file-writing permissions were in place, the PHP code would generate a file called `output.wav`. The audio can be made available easily for playback by ensuring that its location is accessible from the web using the same server. With a typical PHP installation under Apache, the audio in our first example would become available at `http://localhost/output.wav`. Of course, in this example, if we were to record again, the audio would be overwritten. For that reason, it can be useful to define a mechanism for naming the files. The PHP code above handles this by parsing the URL's query parameters, which are separated from the rest of the URL using a '`?`'.

```
var name = userID + "." + sessionID + "." + utteranceID + ".wav";
recorder.startRecording('http://localhost/collect.php?name=' + name);
```

Note that this description is just for illustrative purposes, and does not take into concern the security risks of setting up such a server. Suppose for instance, that a malicious user performed a recording targeted to a URL with `name=collect.php`, overwriting the script with their own content. In practice, it might be wise to ensure on the server-side that the file name follows a pre-specified format, and ends up in a pre-determined directory.

One can use the file name as a simple means of storing meta-information about the audio. The example above depicts a possible file name format that includes a user ID, a session ID, and an utterance ID, all of which are defined and maintained in the JavaScript. The user ID might be stored in a browser cookie which saves information across page reloads. The session might remain the same for the duration of the time that a particular WAMI-enabled page is loaded. Finally, the utterance ID would be incremented each time the user recorded new audio.

C.2 Google App Engine

To show the flexibility and simplicity of implementing WAMI's server-side, here we provide a second example of a WAMI server. Unlike the PHP example, however, we provide a single standard set-up procedure, streamlined by Google, so that the novice web-programmer can begin collecting audio without struggling with the difficulties of hosting one. The server-side code is then hosted using Google's cloud technology called Google

App Engine (GAE). We have open-sourced the necessary code and placed it in a Mercurial repository accessible on Google code:

`https://wami-gapp.googlecode.com`

Signing up for a Google App Engine account is almost as easy as creating a Google account. There is, however, a verification process that requires the creator to provide a cell number to which a verification code will be sent. Once the account has been created, the account holder can create up to 10 applications. Each application must be given a unique name and descriptive title.

There are two main commands that one uses when developing on GAE in python: `dev_appserver.py` and `appcfg.py`. On Mac and Windows machines, however, it can be more convenient to download the SDK, which abstracts these commands away, providing a convenient GUI for development. The launcher can run the application locally to ease debugging, and the SDK console provides an interface to data stored locally so that one's quota is not wasted during development.

For small amounts of audio collection, use of Google App Engine is free. Each GAE account begins with 5GB of free storage space, which comes to over 6 hours of audio recorded at 22,050Hz. Another 5GB is less than $1 per month. When collecting audio on Amazon Mechanical Turk, the subject of the next section's tutorial, paying the workers will undoubtedly dominate the cost of collection, relative to the small amount of money one might have to spend to use Google's cloud service.

There are actually two types of data storage on GAE that will be of interest for our application. The first is the *blobstore*. The term BLOB stands for Binary Large OBject; perfect for the audio data we intend to collect from the client. The second storage mechanism on GAE is called the *datastore*. The datastore typically contains smaller, queryable information such as integers, strings, dates, etc. This type of storage can be useful for saving meta-information about the audio.

Web services hosted on Google App Engine can be implemented in one of three programming languages. The data storage APIs, however, are currently only available in two: Java and Python. This example will make use of Python, since it is typically less verbose. We begin with the simplest possible service capable of collecting and replaying a single audio recording.

```
from google.appengine.ext import webapp
from google.appengine.ext.webapp import util

class WamiHandler(webapp.RequestHandler):
    type = ""
    data = []

    def get(self):
        self.response.headers['Content-Type'] = WamiHandler.type
        self.response.out.write(WamiHandler.data);

    def post(self):
        WamiHandler.type = self.request.headers['Content-Type']
        WamiHandler.data = self.request.body
```

Overriding the `RequestHandler` superclass allows us to handle HTTP requests such as POST and GET. Notice that there is nothing specific to audio in the `WamiHandler` class shown above. We are simply storing the bits of the request body along with the content type in the POST handler. In the GET handler we set the content-type in the response header and serve the data by writing it to the response. Obviously, this simple servlet does not yet store the data to a permanent location, but it does provide a feel for the ease of working within the Google App Engine framework.

The next step is to deploy our simple script to test it out. If we suppose the `RequestHandler` above is implemented in a file called `simple.py` along with some scaffolding to expect requests at the path `/audio`, we can set up the server in just a few quick steps. YAML, a human-readable serialization standard, is used to specify mappings between URLs and scripts or static content such as HTML and JavaScript. In fact, we can host the WAMI recorder, including the JavaScript, SWF file, GUI images, and any HTML content, using the Google App Engine.

```
application: NAME
version: 1
runtime: python
api_version: 1

handlers:
- url: /audio
  script: python/simple.py

- url: /client
  static_dir: public/client
```

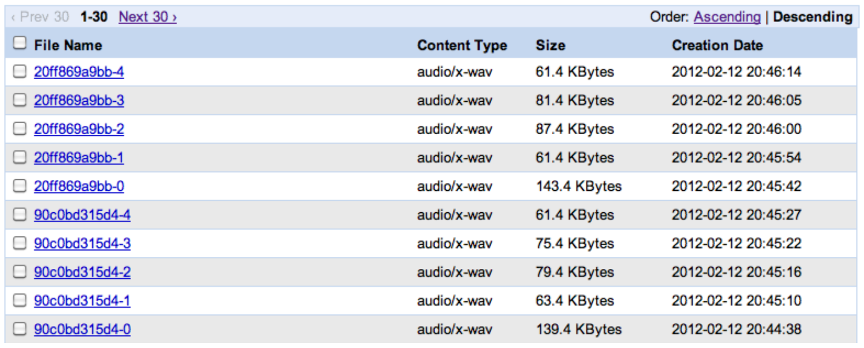
The `.yaml` file above declares a web service called `wami-recorder` along with a few URL mappings. `http://wami-recorder.appspot.com/audio` points to the python and `http://wami-recorder.appspot.com/client` points to a folder containing the WAMI recorder. No additional configuration is necessary to host this simple example. In general, the application is deployed to a Google-managed address `NAME.appspot.com`, where NAME is the unique application identifier chosen upon its creation. In addition to the `simple.py` script described above, we have also included `sessions.py`, which is a blobstore-capable script. With a few more lines of code, it is relatively easy to create a blob and place it into more permanent storage. Finally, a small change must be made to the `app.yaml` file to change the handler for the `/audio` path.

```
- url: /audio
  script: python/sessions.py
```

One of the advantages of using a well-packaged service like the Google App Engine is that there are developer features that make it easy to visualize the back-end of the application. The dashboard of GAE graphs the number of requests per second an application receives over the course of its deployment.

A *Datastore Viewer* and a *Blob Viewer* give the developer a peak into what is being stored. Perhaps most importantly for someone collecting audio using this service, clicking on the link to a (moderately sized) file in the Blob Viewer will allow the developer to play the corresponding audio without explicitly downloading it. Chrome, for instance, recognizes the `audio/x-wav` type that the WAMI recorder sends and that the Google App Engine saves. Thus, clicking a link will play the audio directly in the browser using a simple user interface:

. This feature can be quite useful for keeping track of a remote collection task. Below is an image of the BlobViewer with a few utterances in it.



‹ Prev 30 **1-30** Next 30 › Order: Ascending | **Descending**

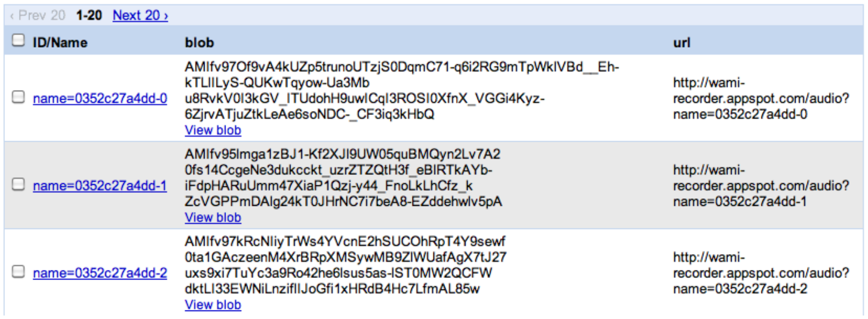
File Name	Content Type	Size	Creation Date
20ff869a9bb-4	audio/x-wav	61.4 KBytes	2012-02-12 20:46:14
20ff869a9bb-3	audio/x-wav	81.4 KBytes	2012-02-12 20:46:05
20ff869a9bb-2	audio/x-wav	87.4 KBytes	2012-02-12 20:46:00
20ff869a9bb-1	audio/x-wav	61.4 KBytes	2012-02-12 20:45:54
20ff869a9bb-0	audio/x-wav	143.4 KBytes	2012-02-12 20:45:42
90c0bd315d4-4	audio/x-wav	61.4 KBytes	2012-02-12 20:45:27
90c0bd315d4-3	audio/x-wav	75.4 KBytes	2012-02-12 20:45:22
90c0bd315d4-2	audio/x-wav	79.4 KBytes	2012-02-12 20:45:16
90c0bd315d4-1	audio/x-wav	63.4 KBytes	2012-02-12 20:45:10
90c0bd315d4-0	audio/x-wav	139.4 KBytes	2012-02-12 20:44:38

Sometimes it can be useful to store additional information about a blob. In this case, the datastore is the more appropriate storage mechanism. In our `sessions.py` example we show how one can store the retrieval URL of the audio along with a reference to its blob in the BlobStore. Our script accomplishes this with a simple data model containing just two properties.

```
from google.appengine.ext import db

# A simple database model to store a URL and an associated blob.
class DataModel(db.Model):
  url = db.StringProperty(required=True)
  blob = blobstore.BlobReferenceProperty(required=True)
```

It is not hard to imagine storing other types of information, such as a user ID, session ID, or utterance ID, by explicitly giving them their own columns in the datastore. GQL, a subset of SQL, can then be used to create queries over the datastore. Here, however, our goal is to provide a simple example of datastore usage. The resulting Datastore Viewer interface looks like the following:



‹ Prev 20 **1-20** Next 20 ›

ID/Name	blob	url
name=0352c27a4dd-0	AMIfv97Of9vA4kUZp5trunoUTzjS0DqmC71-q6i2RG9mTpWkIVBd__Eh-kTLlILyS-QUKwTqyow-Ua3Mb u8RvkV0I3kGV_lTUdohH9uwlCqI3ROSI0XfnX_VGGi4Kyz-6ZjrvATjuZtkLeAe6soNDC-_CF3iq3kHbQ View blob	http://wami-recorder.appspot.com/audio?name=0352c27a4dd-0
name=0352c27a4dd-1	AMIfv95lmga1zBJ1-Kf2XJI9UW05quBMQyn2Lv7A2 0fs14CcgeNe3dukcckt_uzrZTZQtH3f_eBIRTkAYb-iFdpHARuUmm47XiaP1Qzj-y44_FnoLkLhCfz_k ZcVGPPmDAlg24kT0JHrNC7i7beA8-EZddehwlv5pA View blob	http://wami-recorder.appspot.com/audio?name=0352c27a4dd-1
name=0352c27a4dd-2	AMIfv97kRcNliyTrWs4YVcnE2hSUCOhRpT4Y9sewf 0ta1GAczeenM4XrBRpXMSywMB9ZlWUafAgX7tJ27 uxs9xi7TuYc3a9Ro42he6lsus5as-lST0MW2QCFW dktLI33EWNiLnziflIJoGfi1xHRdB4Hc7LfmAL85w View blob	http://wami-recorder.appspot.com/audio?name=0352c27a4dd-2

There are a few additional considerations to keep in mind when using the Google App Engine. First is that it has size limitations for a single query (a few minutes of speech recorded at 22,050Hz.) Second, the programming environment does not allow the installation of native libraries, which might be useful for tasks such as audio resampling. Still, it suffices as a means to get an audio collection task up-and-running quickly.

C.3 Server Configuration Details

The preceding server descriptions suffice for the most basic uses of the WAMI recorder. In certain situations, however, additional configuration is necessary. In some cases it may be necessary to play audio from locations other than where `Wami.swf` is hosted, requiring the use of a cross-domain policy file. Other times, it is necessary to ensure secure HTTP requests using HTTPS. Finally, there are certain applications that require audio streaming rather than the simple bulk-uploading we have presented so far. We describe how to achieve these advanced configurations where possible, and describe work-arounds to approximate their behavior when need be.

C.3.1 Cross-domain Policy File

Flash imposes a security measure called a cross-domain policy file that comes into play when the content being requested by Flash resides on a domain different from that of the Flash application itself. Suppose, for example that the SWF for the recorder is hosted in `http://website-1.com/recorder.swf`. Performing the action `Wami.startPlaying("http://website-2.com/audio.wav")` will then cause a runtime error in ActionScript unless you have the proper cross-domain policy file in place at `http://website-2.com/crossdomain.xml`. An example of what this file might look like, albeit one with very liberal permissions, is shown below.

```
<cross-domain-policy>
    <site-control permitted-cross-domain-policies="master-only"/>
    <allow-access-from domain="*" secure="false" />
    <allow-http-request-headers-from domain="*" headers="*"/>
</cross-domain-policy>
```

Note that manipulating `crossdomain.xml` implies that one needs to have a certain amount of control over the server. For example, in the preceding Google Apps Engine example, the following addition needs to be made to the `app.yaml` file.

```
- url: /crossdomain.xml
  mime_type: text/xml
  static_files: public/crossdomain.xml
  upload: public/crossdomain.xml
```

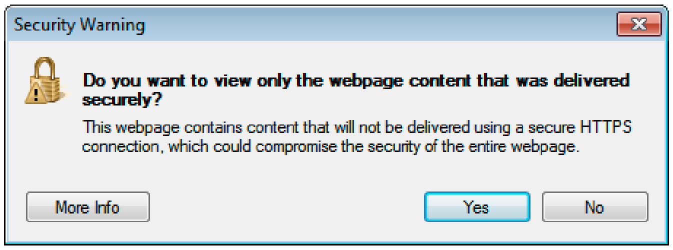
This can be problematic if one would prefer to make use of a third-party's service over which one has no server-side control. Suppose, for example, that a WAMI application needs to play audio from a text-to-speech server set up by a third party to take requests of the form: `http://tts.com?words=hello%20world`. Without a cross-domain policy file in place, the application developer would not be able to play the audio through Flash. One work-around in this scenario is to proxy the audio through one's own servers to make it appear as though it is coming from a domain that has the proper policy file in place. Most web servers make proxies of this sort relatively easy to set up.

C.3.2 HTTPS

In some situations, it is imperative to use HTTPS to send requests rather than vanilla HTTP. This secure communication protocol combines HTTP with SSL to verify server identities and encrypt information going across the network. As an increasing number of web-related security concerns have become public in recent years, major web sites have responded by supporting the `https://` scheme.

Similar to the Applets described in section A.2, web site developers must acquire certificates from a certificate authority (e.g. VeriSign or Microsoft) to host an easily accessible secured site. All the popular browsers of today ship preconfigured to communicate with certain certificate authorities. Cheap or free certificates run the risk of not being fully supported by the popular browsers, however, causing the web sites to retain the undesirable security pop-ups that a pricier certificate would resolve.

Including insecure content (anything communicated through an `http://` URL) in a secure site can also cause security pop-ups in some browsers. The precise behavior varies, however, depending on the manner in which the content is embedded, the user's security settings, as well as the browser in question. For example, including `<iframe src="http://..." />` in a secure site will cause the following pop up message in IE8.


Security Warning

Do you want to view only the webpage content that was delivered securely?

This webpage contains content that will not be delivered using a secure HTTPS connection, which could compromise the security of the entire webpage.

Notice that the default option is "Yes" to a question that asks if viewing only the secure content is OK. This is the opposite of most security dialog messages, even those for previous versions of Internet Explorer. The reasoning for the change is most likely that users often click "Yes" without thinking, which would have previously caused the browser to display insecure content. The error message shown above, however, will at least give the user pause, if only to try to figure out what actually happens when they click "Yes."

C.3.3 Streaming

Data transferred via HTTP are associated with a content-type that specifies the kind of data that will be found in either the request or the response body. For a WAV file, `audio/wav` or `audio/x-wav` are commonly used content types. For the AU container `audio/basic` is used. By default the WAMI recorder gathers up all the audio on the client side before shipping it over to the server via an HTTP POST with the `audio/x-wav` content-type.

There are times, however, when waiting for the audio to finish recording is not ideal. Given that the audio is continuously buffering on the client side, for example, it is probably only reasonable to use the recorder for collecting files of a few megabytes in size. While a couple of minutes of data might be enough for many applications, there are certainly applications that require more. Unless silence can be detected, and the microphone restarted automatically, collecting larger amounts of audio is made difficult with this technique. Moreover, some applications, such as server-side endpoint detection or incremental speech recognition results, actually *must* receive the data before an utterance has ended. In these cases, only a streaming solution will suffice.

Oddly enough, HTTP 1.1 solves this problem by requiring HTTP 1.1 compliant applications to be able to receive a chunked transfer encoding. This method breaks the data down into a series of chunks, which are reassembled on the server-side. Unfortunately, of the three plugins mentioned in previous sections, only Java supports the chunked transfer encoding. Flash does not provide a mechanism for performing POSTs in this fashion, and the only solution (without resorting to using a flash media player) is to implement a custom chunking protocol using multiple sequential HTTP POSTs. The individual POSTs, perhaps each containing a couple hundred milliseconds of audio, can be reassembled on the server-side to perform a sort of psuedo-streaming. Streaming a WAV file, however, is not really possible since the header information of a WAV expects a length, which would, of course, be unknown a priori. Since the WAV specification does not officially support data of unknown length, one could turn to an audio container that does, such as AU.

Since the streaming features are somewhat more involved to implement on both the server and client side of the audio collection equations, we leave them out of our default implementation in WAMI. Fortunately, for simple collection tasks like the one described in the next section, advanced streaming features are unnecessary.

Appendix D

Example: Speech Collection on mTurk

This section is a high-level overview of the steps necessary to set up an audio collection HIT on Amazon Mechanical Turk. More detailed, step-by-step instructions as well as the complete code for this tutorial can be found in the Google Code repository located here:

`https://wami-gapp.googlecode.com`

This example collection task will have workers read movie titles aloud, but generalizes easily to any prompted audio collection task. We begin by describing the necessary server-side set up for the task, making use of the Google App Engine framework. We then discuss two ways of deploying the task. The first, and simplest, utilizes the Amazon Mechanical Turk web interface to deploy and manage the audio collection HIT. The second method of deployment is through Amazon Mechanical Turk's command line tools by way of their `ExternalQuestion` HIT, which allows developers to embed a web page hosted outside of AMT. The command line tools themselves make it possible to script HIT management, making working with a large number of HITs easier.

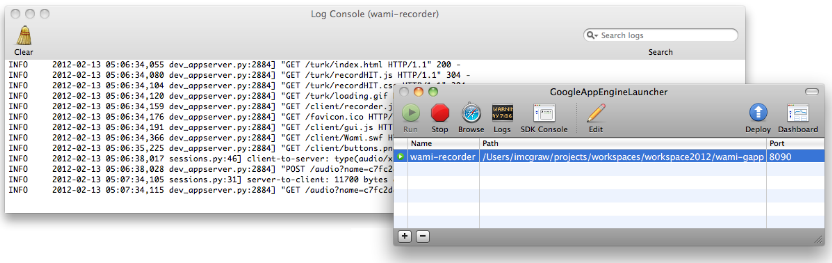
There is a third, more advanced method, whereby o'ne can deploy HITs programmatically using Java, C#, or a number of other languages. These SDKs not only offer fine-grained control over a HIT, they enable the development of human-in-the-loop applications that can ask questions of the outside world, such as *"What does this person say in this audio clip?"* or *"How do you pronounce Caddyshack?"*. Using this method, an application could conceivably automatically collect its own training data without expert supervision. These SDKs, however, are beyond the scope of this tutorial. In future sections, we describe nascent efforts in this arena.

D.1 Server Setup

The first step is to check out the code (perhaps to ~/wami-gapp) and run it on the local machine. To do so, however, one first needs to become familiar with Google App Engine, since we will let this cloud service do the heavy lifting on the server-side of our application. Signing up for an account is relatively straightforward.

With a new Google App Engine account in hand, it's possible to use the web interface to create an application by giving it a unique application ID and a title. The application ID will be used in the URL, so it is preferable to choose something easy to type if need be. Choosing `NAME` as an ID reserves `NAME.appspot.com`; however, many IDs may already be taken. Once the application has been created, a dashboard is accessible, although no application has yet been deployed.

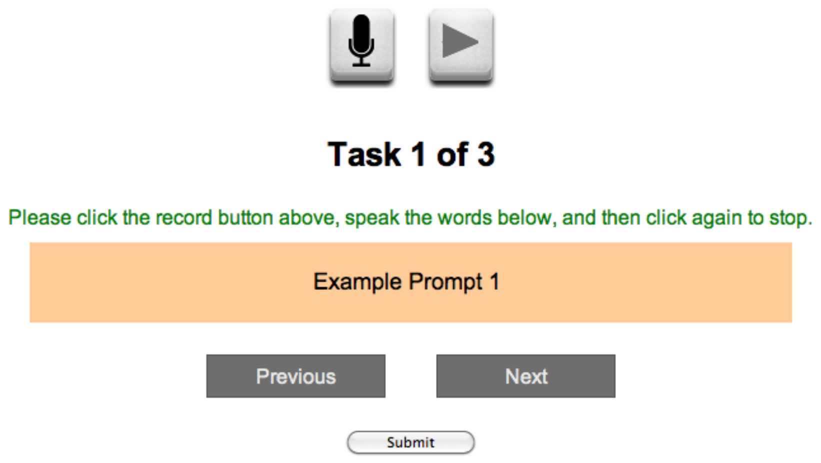
Before deploying an application, it is wise to run and test it locally. For this purpose, Google has provided an App Engine SDK for python. The GUI version of the SDK, available for both Mac OS X and Windows, is somewhat more user friendly than the command line scripts available for Linux. The following image depicts the Google App Engine Launcher and the corresponding logging console.



```
INFO    2012-02-13 05:06:34,055 dev_appserver.py:2884] "GET /turk/index.html HTTP/1.1" 200 -
INFO    2012-02-13 05:06:34,080 dev_appserver.py:2884] "GET /turk/recordHIT.js HTTP/1.1" 304 -
INFO    2012-02-13 05:06:34,104 dev_appserver.py:2884] "GET /turk/recordHIT.cs
INFO    2012-02-13 05:06:34,120 dev_appserver.py:2884] "GET /turk/loading.gif
INFO    2012-02-13 05:06:34,159 dev_appserver.py:2884] "GET /client/recorder.j
INFO    2012-02-13 05:06:34,176 dev_appserver.py:2884] "GET /favicon.ico HTTP/
INFO    2012-02-13 05:06:34,191 dev_appserver.py:2884] "GET /client/gui.js HTT
INFO    2012-02-13 05:06:34,366 dev_appserver.py:2884] "GET /client/Wami.swf H
INFO    2012-02-13 05:06:35,225 dev_appserver.py:2884] "GET /client/buttons.pn
INFO    2012-02-13 05:06:38,017 sessions.py:46] client-to-server: type(audio/x
INFO    2012-02-13 05:06:38,028 dev_appserver.py:2884] "POST /audio?name=c7fc2
INFO    2012-02-13 05:07:34,105 sessions.py:31] server-to-client: 11700 bytes
INFO    2012-02-13 05:07:34,115 dev_appserver.py:2884] "GET /audio?name=c7fc2d
```

Name	Path	Port
wami-recorder	/Users/imcgraw/projects/workspaces/workspace2012/wami-gapp	8090

With the SDK installed, attaching the project is simply a matter of performing the *Add Existing Application* operation to attach ~/wami-gapp to the application launcher. A default port, such as 8080, will be provided such that running the app will make it available at `localhost:8080`. In particular, ~/wami-gapp/turk/index.html will now be available at `http://localhost:8080/turk/index.html`. This is the interface that we will be deploying to Amazon Mechanical Turk for audio collection.


Task 1 of 3
Please click the record button above, speak the words below, and then click again to stop.
Example Prompt 1
Previous
Next
Submit

Even when everything is running smoothly, it is probably wise to check the logging console for anomalies. The `sessions.py` python code contains some examples of how a programmer can write messages to this console. If the logs look clean, however, there is just one small change that must be made before deployment. The top line of `~/wami-gapp/app.yaml`, the file that configures the server, specifies the name of the app to be deployed. This must be changed to the unique ID of the app that was created through the Google App Engine web interface. Failure to do so will result in an error message in the log along the lines of *"this application does not exist."*

A successful deployment will make the application available to the outside world at `http://NAME.appspot.com/turk/index.html`. Note that the special port is no longer part of the URL. We also provide an example of the recorder without the Amazon Mechanical Turk interface at `http://NAME.appspot.com/client/index.html`. With either site, it should now be possible to record an utterance and then see it appear in the *Blob Viewer* in the Google App Engine web interface.

D.2 Deploying to Amazon Mechanical Turk

Now that audio collection has been tested end-to-end, we are ready to deploy a Human Intelligence Task (HIT) on Amazon Mechanical Turk. There are a number of ways to access AMT, but the web-interface is arguably the simplest, so we use it here. We recommend starting with the AMT *requester's sandbox*, which is a website identical to the real AMT, but which will not cost money to experiment on. Of course, this means there are no workers willing to complete the tasks, but the sandbox is a great way to test out a HIT for the first time.

Once logged into the sandbox with an Amazon account, the first step is to navigate to the *Design* page, where HIT templates are created. Often it is easiest to start with a blank template. After choosing the blank template option, the requester is presented with a page to enter the properties of the HIT, such as a name, description and keywords. This is also where the payment is specified, as well as the number of assignments per HIT. Any particular worker will only be able to perform a given HIT one time, but if you specify a number of assignments per HIT greater than one, another worker will be allowed to come along and perform that HIT as well.

Perhaps the most important properties for a task, other than price, are the qualification restrictions placed on a HIT. With speech, in particular, the criteria by which workers are filtered has a large effect on the results. Restricting worker by location is one way to ensure that the language and accent roughly meet the criteria of the task, though it is by no means a guarantee. Qualification tests are another way to filter workers. With these, it may be possible to pre-approve the voices that are allowed to perform a certain HIT.

Once the properties have been selected, the requester must design the layout of the HIT. Fortunately Amazon has given us a large amount of flexibility. In particular, in addition to a WYSIWYG[1] editor, there is a button on the interface that allows one to edit the HTML directly: Edit HTML Source. Using the HTML editor we can replace everything in the text area with HTML and JavaScript of our choosing. In the case, of our example audio collection HIT, we can replace all of the HTML code present in the HIT by default with just a few simple lines of code similar to those found in ~/wami-gapp/public/turk/index.html file.

```
<style type="text/css">@import url(recordHIT.css);</style>
<script type="text/javascript" src="recordHIT.js"></script>
<p><input type="hidden" id="wsession" /></p>
<script>
  var wsessionid = Wami.RecordHIT.create("${prompts}");
  document.getElementById('wsession').value = wsessionid;
</script>
<noscript>Please enable JavaScript to perform a HIT.</noscript>
```

Previewing the task should yield an interface similar to the one shown previously. It is possible, at this stage, to add a header describing the task, or perhaps a feedback text area to be submitted with the HIT. Once the template is saved, we are ready to begin filling in the `${prompts}` variable with prompts for the particular task at hand. In our case, we will insert the movie titles into our HIT using the list found in ~/wami-gapp/turk/movies.txt. To do this, we navigate to the *Publish* page of Amazon Mechanical Turk and select the audio collection template we have just saved. If all goes well, the interface will then prompt the requester to upload input data.

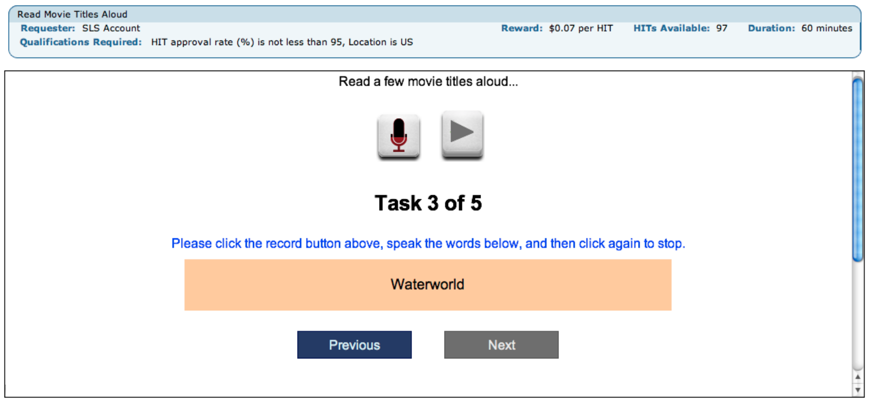
[1]What You See Is What You Get!

```
prompts
"The Green Mile<>Babe<>Waterworld<>Mad Max Beyond Thunderdome<>Caddyshack"
"The Thin Red Line<>Pitch Black<>Bridge to Terabithia<>Ed Wood<>Godspell"
...
```

Notice that the prompts are separated by a delimeter: <>. This allows our JavaScript to bundle multiple prompts up into a single HIT. For large HITs, this bundling technique can save a significant amount of money on Amazon's commission. Although Amazon usually takes a 10% commission on a task, one that pays less than 5¢ is subject to Amazon's minimum commission charge of 0.5¢. Thus, tasks above 5¢ get more value-per-cent spent than those below.

Given the format described above, there are some special characters that cannot appear within a prompt. Fortunately, the prompts are inserted directly into the page as HTML. This means that HTML escape codes can be used to correctly display special characters. Moreover, HTML tags and styles can be used to insert any desired content in place of a prompt. For example, it would not be difficult to insert a YouTube video to prompt for a verbal response. In our simple example, we will stick with plain text.

After uploading the input data file, a preview of the interface should show the movie titles in place of the `${prompts}` variable. The requester is then asked to confirm the cost (even the sandbox provides a sample cost summary), after which one can publish the HITs. Publishing HITs in the requester's sandbox will make them available in a corresponding worker's sandbox. From this interface it is possible to test the task in full.



Read Movie Titles Aloud

Requester: SLS Account **Reward:** $0.07 per HIT **HITs Available:** 97 **Duration:** 60 minutes

Qualifications Required: HIT approval rate (%) is not less than 95, Location is US

Read a few movie titles aloud...

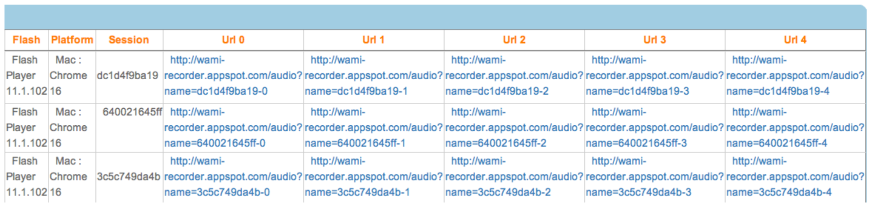
Task 3 of 5

Please click the record button above, speak the words below, and then click again to stop.

Waterworld

Had we been deploying this HIT to the real Amazon Mechanical Turk interface, there is one very important step that we have not yet described. In many settings, research involving the use of human subjects requires the approval of an Institutional Review Board (IRB). Typically, tasks in which workers do not remain anonymous are subject to additional scrutiny and one could reasonably consider a voice to be identifying information. If approved, IRBs often require that a consent form be supplied to the worker, either as part of the HIT or as a separate qualification step.

Make sure the full submission process works from a few different browsers and always check to see that the results are being collected as expected. To do this, return to the requester's sandbox after submitting a few HITs. Find the page where requesters can *Manage* HITs, and get the results of the current task. They should look something like the following.



Flash	Platform	Session	Url 0	Url 1	Url 2	Url 3	Url 4
Flash Player 11.1.102	Mac : Chrome 16	dc1d4f9ba19	http://wami-recorder.appspot.com/audio?name=dc1d4f9ba19-0	http://wami-recorder.appspot.com/audio?name=dc1d4f9ba19-1	http://wami-recorder.appspot.com/audio?name=dc1d4f9ba19-2	http://wami-recorder.appspot.com/audio?name=dc1d4f9ba19-3	http://wami-recorder.appspot.com/audio?name=dc1d4f9ba19-4
Flash Player 11.1.102	Mac : Chrome 16	640021645ff	http://wami-recorder.appspot.com/audio?name=640021645ff-0	http://wami-recorder.appspot.com/audio?name=640021645ff-1	http://wami-recorder.appspot.com/audio?name=640021645ff-2	http://wami-recorder.appspot.com/audio?name=640021645ff-3	http://wami-recorder.appspot.com/audio?name=640021645ff-4
Flash Player 11.1.102	Mac : Chrome 16	3c5c749da4b	http://wami-recorder.appspot.com/audio?name=3c5c749da4b-0	http://wami-recorder.appspot.com/audio?name=3c5c749da4b-1	http://wami-recorder.appspot.com/audio?name=3c5c749da4b-2	http://wami-recorder.appspot.com/audio?name=3c5c749da4b-3	http://wami-recorder.appspot.com/audio?name=3c5c749da4b-4

Notice that there is a URL for every piece of audio collected. To hear the audio, click on these URLs from a browser that can handle the `audio/x-wav` mime-type. We have also saved some information about the user's system including their OS, browser, and Flash versions. This can be useful for debugging the HIT when behavior varies across system configuration.

D.3 The Command Line Interface

We can perform the same HIT with AMT's `ExternalQuestion` template using the command line tools (CLT). Within the samples provided by the CLT, the `external_hit` sample is set up almost perfectly for the task at hand. We simply replace the URL in the `.question` file with our application's URL:

```
https://NAME.appspot.com/turk/index.html?prompts=${helper.urlencode($prompts)}
```

The prompts file format is the same as in the previous section. The call to `helper.urlencode($prompts)` takes the prompts field of our input data, and escapes the special characters so that it can be inserted as a URL parameter. Our JavaScript then does the job of retrieving the parameter and parsing individual prompts. Note that with some browser/server configurations URLs with more than a few thousand characters cause errors. In these situations, it may be necessary to load data into the web-page dynamically using AJAX. In many cases, however, embedding prompts in the URL should suffice.

Amazon Mechanical Turk HITs, even those deployed through the web-interface, are really nothing more than HTML forms whose results are posted to Amazon's servers. Thus, we can create the same form, called `mturk_form` in our external hit. We can explicitly add HTML `input` elements to the form, or dynamically create them in JavaScript. The `hidden` input type is useful for saving result fields without the worker having to view them. At the very least, we must specify the assignment ID in this form. Finally, since we have full control over the submit button, we can validate results programmatically before submission.

The web site used in the `ExternalQuestion` must be served via HTTPS. Recall that serving insecure content via HTTP will cause security alerts under some browser configurations. More importantly, the Amazon Mechanical Turk site recently began requiring the use of HTTPS. Fortunately, there is a free way to serve almost any content with a properly configured HTTPS certificate: one can piggyback off of the Google App Engine certificates. We have, in fact, been doing this already for everything hosted through `https://NAME.appspot.com`. Although slightly more work, even content not hosted on Google App Engine can be proxied through a GAE account, ensuring that the requests made from AMT are secure.

Bibliography

[1] Hua Ai, Antoine Raux, Dan Bohus, Maxine Eskenazi, and Diane Litman. Comparing spoken dialog corpora collected with recruited subjects versus real users. In *Proc. of SIGdial*, 2007.

[2] Vamshi Ambati, Stephan Vogel, and Jaime G. Carbonell. Active learning and crowd-sourcing for machine translation. In *Proc. of LREC*, 2010.

[3] Judd Antin and Aaron Shaw. Social desirability bias in reports of motivation for us and india workers on mechanical turk. In *Proc. of CSCW*, 2011.

[4] Harald Aust, Martin Oerder, Frank Seide, and Volker Steinbiss. Experience with the philips automatic train timetable information system. In *Proc. of the Second IEEE Workshop on Interactive Voice Technology for Telecommunications Applications (IVTTA)*, pages 67 –72, September 1994.

[5] Ibrahim Badr, Ian McGraw, and James R. Glass. Learning new word pronunciations from spoken examples. In *Proc. of INTERSPEECH*, pages 2294–2297, 2010.

[6] Ibrahim Badr, Ian McGraw, and James R. Glass. Pronunciation learning from continuous speech. In *Proc. of INTERSPEECH*, pages 549–552, 2011.

[7] L. R. Bahl, S. Das, P. V. Desouza, M. Epstein, R. L. Mercer, B. Merialdo, D. Nahamoo, M. A. Picheny, and J. Powell. Automatic phonetic baseform determination. In *in Proc. IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 173–176. IEEE, 1991.

[8] Jerome R. Bellegarda. Unsupervised, language-independent grapheme-to-phoneme conversion by latent analogy. *Speech Communication*, 46(2):140–152, June 2005.

[9] Jared Bernstein, Kelsey Taussig, and Jack Godfrey. Macrophone: an american english telephone speech corpus for the polyphone project. In *Proc. of ICASSP*, 1994.

[10] MIchael Bernstein. Crowd-powered systems, 2012. Ph.D. thesis, Department of Electrical Engineering and Computer Science, MIT.

[11] Michael S. Bernstein, Joel Brandt, Robert C. Miller, and David R. Karger. Crowds in two seconds: enabling realtime crowd-powered interfaces. In *Proc. of UIST*, 2011.

[12] Michael S. Bernstein, Greg Little, Robert C. Miller, Björn Hartmann, Mark S. Ackerman, David R. Karger, David Crowell, and Katrina Panovich. Soylent: a word processor with a crowd inside. In *Proc. of UIST*, 2010.

[13] Michael S. Bernstein, Robert C. Miller, David R. Karger, and Joel Brandt. Analytic methods for optimizing realtime crowdsourcing. In *Proc. of Collective Intelligence*, 2012.

[14] Jeffrey P. Bigham, Chandrika Jayant, Hanjie Ji, Greg Little, Andrew Miller, Robert C. Miller, Robin Miller, Aubrey Tatarowicz, Brandyn White, Samual White, and Tom Yeh. VizWiz: nearly real-time answers to visual questions. In *Proc. of UIST*, 2010.

[15] Maximilian Bisani and Hermann Ney. Open vocabulary speech recognition with flat hybrid models. In *Proc. of INTERSPEECH*, pages 725–728, 2005.

[16] Maximilian Bisani and Hermann Ney. Joint-sequence models for grapheme-to-phoneme conversion. *Speech Communication*, 50(5):434–451, May 2008.

[17] Peter F. Brown, Peter V. deSouza, Robert L. Mercer, Vincent J. Della Pietra, and Jenifer C. Lai. Class-based n-gram models of natural language. *Computational Linguistics*, 18(4):467–479, December 1992.

[18] Sabine Buchholz and Javier Latorre. Crowdsourcing preference tests, and how to detect cheating. In *Proc. of INTERSPEECH*, 2011.

[19] Stephan Busemann and Helmut Horacek. A flexible shallow approach to text generation. In *Proc. of the Ninth International Workshop on Natural Language Generation*, 1998.

[20] Jun Cai, Jacques Feldmar, Yves Laprie, and Jean-Paul Haton. Transcribing southern min speech corpora with a web-based language learning system. In *Proc. of International Workshop on Spoken Language Technologies for Under-resourced languages (SLTU)*, May 2008.

[21] Chris Callison-Burch. Fast, cheap, and creative: Evaluating translation quality using Amazon's Mechanical Turk. In *Proc. of the Conference on Empirical Methods in Natural Language Processing*, Singapore, August 2009.

[22] Chris Callison-Burch and Mark Dredze. Creating speech and language data with amazon's mechanical turk. In *Proc. of the NAACL HLT Workshop on Creating Speech and Language Data with Amazon's Mechanical Turk*, 2010.

[23] Alexandra Canavan, David Graff, and George Zipperlen. CALLHOME American English Speech, 1997. *Available from the Linguistic Data Consortium* (LDC).

[24] Jean Carletta. Assessing agreement on classification tasks: the kappa statistic. *Computational Linguistics*, 22(2):249–254, June 1996.

[25] Dana Chandler and Adam Kapelner. Breaking monotony with meaning: Motivation in crowdsourcing markets. *University of Chicago*, 2010.

[26] Jonathan Chang, Jordan Boyd-Graber, Chong Wang, Sean Gerrish, and David M. Blei. Reading tea leaves: How humans interpret topic models. In *Proc. of NIPS*, 2009.

[27] Stanley F. Chen. Conditional and Joint Models for Grapheme-to-Phoneme Conversion. In *Proc. of INTERSPEECH*, 2003.

[28] Chevrolet. MyLink. http://www.gracenote.com/events/chevrolet_mylink, May 2012.

[29] Ghinwa F. Choueiter, Mesrob I. Ohannessian, Stephanie Seneff, and James R. Glass. A turbo-style algorithm for lexical baseforms estimation. In *Proc. of ICASSP*, 2008.

[30] Grace Chung, Stephanie Seneff, and Chao Wang. Automatic induction of language model data for a spoken dialogue system. In *In Proc. of SIGDIAL*, 2005.

[31] Christopher Cieri, Linda Corson, David Graff, and Kevin Walker. Resources for new research directions in speaker recognition: the mixer 3, 4 and 5 corpora. In *Proc. of INTERSPEECH*, 2007.

[32] Ford Motor Company. SYNC. http://www.ford.com/syncmyride, May 2012.

[33] Deborah A. Dahl, Madeleine Bates, Michael Brown, William Fisher, Kate Hunicke-Smith, David Pallett, Christine Pao, Alexander Rudnicky, and Elizabeth Shriberg. Expanding the scope of the ATIS task: the ATIS-3 corpus. In *Proc. of HLT*, 1994.

[34] Peng Dai, Mausam, and Daniel S. Weld. Artificial intelligence for artificial artificial intelligence. In *Proc. of AAAI*, 2011.

[35] Christopher Cieri David, David Miller, and Kevin Walker. The Fisher corpus: a resource for the next generations of speech-to-text. In *Proc. of LREC*, 2004.

[36] A P Dawid and A M Skene. Maximum likelihood estimation of observer error-rates using the EM algorithm. *Journal of the Royal Statistical Society Series C Applied Statistics*, 28(1):20–28, January 1979.

[37] Arthur Dempster, Nan Laird, and Donald Rubin. Maximum likelihood from incomplete data via the EM algorithm. *JOURNAL OF THE ROYAL STATISTICAL SOCIETY, SERIES B*, 39(1):1–38, 1977.

[38] Giuseppe Di Fabbrizio, Thomas Okken, and Jay G. Wilpon. A speech mashup framework for multimodal mobile services. In *Proc. of ICMI*, 2009.

[39] Steven Dow, Anand Kulkarni, Scott Klemmer, and Björn Hartmann. Shepherding the crowd yields better work. In *Proc. of CSCW*, 2012.

[40] Christoph Draxler. WWWTranscribe – a modular transcription system based on the World Wide Web. In *Proc. of EUROSPEECH*, 1997.

[41] Christoph Draxler. Exploring the Unknown – Collecting 1000 speakers over the Internet for the Ph@ttSessionz Database of Adolescent Speakers. In *Proc. of INTERSPEECH*, Pittsburgh, PA, 2006.

[42] William Fisher, George Doddington, and Kathleen Goudie-Marshall. The DARPA speech recognition research database: Specification and status. In *Proc. of the DARPA Speech Recognition Workshop*, 1986.

[43] Michael J. Franklin, Donald Kossmann, Tim Kraska, Sukriti Ramesh, and Reynold Xin. CrowdDB: answering queries with crowdsourcing. In *Proc. of SIGMOD*, 2011.

[44] João Freitas, António Calado, Daniela Braga, Pedro Silva, and Miguel Sales Dias. Crowd-sourcing platform for large-scale speech data collection. In *Proc. of VI Jornadas en Tecnología del Habla and II Iberian SLTech Workshop FALA*, 2010.

[45] M. Gašić, F. Jurčíček, B. Thomson, K. Yu, and S. Young. On-line policy optimisation of spoken dialogue systems via live interaction with human subjects. In *Proc. of ASRU*, 2011.

[46] David Geiger, Stefan Seedorf, Thimo Schulze, Robert Nickerson, and Martin Schader. Managing the crowd : Towards a taxonomy of crowdsourcing processes. In *Proc. of AMCIS*, 2011.

[47] James Glass. A probabilistic framework for segment-based speech recognition. *Computer Speech and Language*, 17(2-3):137–152, 2003.

[48] John J. Godfrey, Edward C. Holliman, and Jane McDaniel. SWITCHBOARD: telephone speech corpus for research and development. In *Proc. of ICASSP*, 1992.

[49] Allen L Gorin, Giuseppe Riccardi, and Jeremy H. Wright. How may I help you? *Speech Communication*, 23(12):113 – 127, 1997.

[50] Samsung Group. SmartTV. http://www.samsung.com/us/2012-smart-tv/, May 2012.

[51] Alex Gruenstein and Stephanie Seneff. Releasing a multimodal dialogue system into the wild: User support mechanisms. In *Proc. of SIGdial*, 2007.

[52] Alexander Gruenstein, Ian McGraw, and Ibrahim Badr. The WAMI toolkit for developing, deploying, and evaluating web-accessible multimodal interfaces. In *Proc. of ICMI*, 2008.

[53] Alexander Gruenstein, Ian McGraw, and Andrew Sutherland. A self-transcribing speech corpus: Collecting continuous speech with an online educational game. In *Proc. of Speech and Language Technology in Education Workshop (SLaTE)*, 2009.

[54] Alexander Gruenstein and Stephanie Seneff. Releasing a multimodal dialogue system into the wild: User support mechanisms. In *Proc. of SIGdial*, 2007.

[55] Alexander Gruenstein, Stephanie Seneff, and Chao Wang. Scalable and portable web-based multimodal dialogue interaction with geographical databases. In *Proc. of INTERSPEECH*, 2006.

[56] Shelby J. Haberman. *Analysis of Qualitative Data: Volume 2, New Developments*. Academic Press, New York, 1979.

[57] Christopher Harris. You're hired! An examination of crowdsourcing incentive models in human resource tasks. In *Proc. of the Workshop on Crowdsourcing for Search and Data Mining at WSDM*, 2011.

[58] Helen Wright Hastie, Rashmi Prasad, and Marilyn Walker. Automatic evaluation: Using a date dialogue act tagger for user satisfaction and task completion prediction. In *Proc. LREC*, 2002.

[59] Timothy J. Hazen, Lee Hetherington, Han Shu, and Karen Livescu. Pronunciation modeling using a finite-state transducer representation. *Speech Communication*, 46(2):189 – 203, June 2005.

[60] Timothy J. Hazen, Stephanie Seneff, and Joseph Polifroni. Recognition confidence scoring and its use in speech understanding systems. *Computer Speech and Language*, 16:49–67, 2002.

[61] Charles T. Hemphill, John J. Godfrey, and George R. Doddington. The ATIS spoken language systems pilot corpus. In *Proc. of the DARPA Speech and Natural Language Workshop*, 1990.

[62] Hynok Hermansky. Perceptual linear predictive (plp) analysis of speech. *Journal of the Acoustical Society of America*, 87(4):1738–1752, 1990.

[63] Lee Hetherington. The MIT finite-state transducer toolkit for speech and language processing. In *Proc. of ICSLP*, 2004.

[64] Lynette Hirschman. Multi-site data collection for a spoken language corpus. In *Proc. of the workshop on Speech and Natural Language at HLT*, 1992.

[65] Xuedong Huang, Alex Acero, and Hsiao-Wuen Hon. *Spoken Language Processing: A Guide to Theory, Algorithm, and System Development*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition, 2001.

[66] Edward Hurley, Joseph Polifroni, and James Glass. Telephone data collection using the world wide web. In *Proc. of ICSLP*, 1996.

[67] Apple Inc. Siri. http://www.apple.com/iphone/features/siri.html, May 2012.

[68] Panagiotis G. Ipeirotis, Foster Provost, and Jing Wang. Quality management on amazon mechanical turk. In *Proc. of HCOMP*, 2010.

[69] Rukmini Iyer and Mari Ostendorf. Modeling long distance dependence in language: topic mixtures versus dynamic cache models. *IEEE Transactions on Speech and Audio Processing*, 7(1):30–39, January 1999.

[70] Sittichai Jiampojamarn and Grzegorz Kondrak. Online discriminative training for grapheme-to-phoneme conversion. In *Proc. of INTERSPEECH*, 2009.

[71] Daniel Jurafsky and James H. Martin. *Speech and Language Processing (2nd Edition).* Prentice Hall, 2 edition, 2008.

[72] F. Jurčíček, S. Keizer, M. Gašić, F. Mairesse, B. Thomson, K. Yu, and S. Young. Real user evaluation of spoken dialogue systems using amazon mechanical turk. In *Proc. of INTERSPEECH*, 2011.

[73] Michael Kaisser and John Lowe. Creating a research collection of question answer sentence pairs with amazon's mechanical turk. In *Proc. of LREC*, 2008.

[74] Ronald M. Kaplan and Martin Kay. Regular models of phonological rule systems. *Computational Linguistics*, 20(3):331–378, 1994.

[75] Nicolas Kaufmann and Daniel Veit. More than fun and money. Worker motivation in crowdsourcing – A study on mechanical turk. *Proc. of AMCIS*, 2011.

[76] Paul Kingsbury. PronLex, COMLEX English Pronunciation Dictionary, 1995. *Available from the Linguistic Data Consortium* (LDC).

[77] Aniket Kittur. Crowdsourcing, collaboration and creativity. *XRDS*, 17(2):22–26, December 2010.

[78] Aniket Kittur, H. Chi, and Bongwon Suh. Crowdsourcing user studies with mechanical turk. In *Proc. of CHI*, 2008.

[79] Aniket Kittur, Susheel Khamkar, Paul André, and Robert Kraut. Crowdweaver: visually managing complex crowd work. In *Proc. of CSCW*, 2012.

[80] Aniket Kittur and Robert E. Kraut. Harnessing the wisdom of crowds in wikipedia: quality through coordination. In *Proc. of CSCW*, 2008.

[81] Aniket Kittur, Boris Smus, Susheel Khamkar, and Robert E. Kraut. Crowdforge: crowdsourcing complex work. In *Proc. of UIST*, 2011.

[82] Ikuo Kudo, Takao Nakama, Tomoko Watanabe, and Reiko Kameyama. Data collection of japanese dialects and its influence into speech recognition. In *Proc. of ICSLP*, 1996.

[83] Anand Kulkarni, Matthew Can, and Björn Hartmann. Collaboratively crowdsourcing workflows with Turkomatic. In *Proc. of CSCW*, 2012.

[84] Stephen A. Kunath and Steven H. Weinberger. The wisdom of the crowd's ear: speech accent rating and annotation with Amazon Mechanical Turk. In *Proc. of the NAACL HLT Workshop on Creating Speech and Language Data with Amazon's Mechanical Turk*, 2010.

[85] John Lafferty, Andrew McCallum, and Fernando Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning*, 2001.

[86] Ian Lane, Alex Waibel, Matthias Eck, and Kay Rottmann. Tools for collecting speech corpora via mechanical-turk. In *Proc. of the NAACL HLT Workshop on Creating Speech and Language Data with Amazon's Mechanical Turk*, 2010.

[87] Antoine Laurent, Paul Delglise, and Sylvain Meignier. Grapheme to phoneme conversion using an SMT system. In *Proc. of INTERSPEECH*, 2009.

[88] Florian Laws, Christian Scheible, and Hinrich Schütze. Active learning with amazon mechanical turk. In *Proc. of EMNLP*, 2011.

[89] Nolan Lawson, Kevin Eustice, Mike Perkowitz, and Meliha Yetisgen-Yildiz. Annotating large email datasets for named entity recognition with Mechanical Turk. In *Proc. of the NAACL HLT Workshop on Creating Speech and Language Data with Amazon's Mechanical Turk*, 2010.

[90] Jonathan Ledlie, Billy Odero, Einat Minkov, Imre Kiss, and Joseph Polifroni. Crowd Translator: On building localized speech recognizers through micropayments. In *Third Annual Workshop on Networked Systems for Developing Regions (NSDR)*, 2009.

[91] Xiao Li, A. Gunawardana, and A. Acero. Adapting grapheme-to-phoneme conversion for name recognition. In *Proc. of ASRU*, 2007.

[92] Greg Little, Lydia B. Chilton, Max Goldman, and Robert C. Miller. Turkit: tools for iterative tasks on mechanical turk. In *Proc. of HCOMP*, New York, NY, USA, 2009.

[93] Jingjing Liu, Scott Cyphers, Panupong Pasupat, Ian McGraw, and Jim Glass. A conversational movie search system based on conditional random fields. Submitted to *INTERSPEECH*, 2012.

[94] Jean luc Gauvain and Chin hui Lee. Maximum a posteriori estimation for multivariate gaussian mixture observations of markov chains. *IEEE Transactions on Speech and Audio Processing*, 2:291–298, 1994.

[95] François Mairesse, Milica Gašić, Filip Jurčíček, Simon Keizer, Blaise Thomson, Kai Yu, and Steve Young. Phrase-based statistical language generation using graphical models and active learning. In *Proc. of ACL*, 2010.

[96] Benoît Maison. Automatic baseform generation from acoustic data. In *Proc. of INTERSPEECH*, 2003.

[97] Yannick Marchand and Robert I. Damper. A multi-strategy approach to improving pronunciation by analogy. *Computational Linguistics*, 26(2):195–219, June 2000.

[98] Adam Marcus, Eugene Wu, David Karger, Samuel Madden, and Robert Miller. Human-powered sorts and joins. In *Proc. of VLDB*, 2011.

[99] Matthew Marge, S Banerjee, and A Rudnicky. Using the amazon mechanical turk for transcription of spoken language. In *Proc ICASSP*, 2010.

[100] Matthew Marge, Satanjeev Banerjee, and Alexander I. Rudnicky. Using the Amazon Mechanical Turk to transcribe and annotate meeting speech for extractive summarization. In *Proc. of the NAACL HLT 2010 Workshop on Creating Speech and Language Data with Amazon's Mechanical Turk*, 2010.

[101] Winter Mason and Siddharth Suri. Conducting behavioral research on Amazon?s Mechanical Turk. *Behavior Research Methods*, 5(5):1–23, 2011.

[102] Winter Mason and Duncan J. Watts. Financial incentives and the "performance of crowds". In *Proc. of HCOMP*, 2009.

[103] Ian McGraw, Scott Cyphers, Panupong Pasupat, Jingjing Liu, and Jim Glass. Automating crowd-supervised learning for spoken language systems. Submitted to *INTERSPEECH*, 2012.

[104] Ian McGraw, James Glass, and Stephanie Seneff. Growing a spoken language interface on Amazon Mechanical Turk. In *Proc. of INTERSPEECH*, 2011.

[105] Ian McGraw, Alexander Gruenstein, and Andrew Sutherland. A self-labeling speech corpus: Collecting spoken words with an online educational game. In *Proc. of INTERSPEECH*, 2009.

[106] Ian McGraw, Chia ying Lee, Lee Hetherington, Stephanie Seneff, and James R. Glass. Collecting voices from the cloud. In *Proc. of LREC*, 2010.

[107] Mehryar Mohri. Weighted automata algorithms. *Handbook of weighted automata*, 3(1):213–254, 2009.

[108] Tan Nguyen, Shijun Wang, Vishal Anugu, Natalie Rose, Matthew McKenna, Nicholas Petrick, Joseph Burns, and Ronald Summers. Distributed human intelligence for colonic polyp classification in computer-aided detection for CT colonography. *Radiology*, 262(3):824–833, March 2012.

[109] Jon Noronha, Eric Hysen, Haoqi Zhang, and Krzysztof Z. Gajos. Platemate: crowdsourcing nutritional analysis from food photographs. In *Proc. of UIST*, 2011.

[110] Scott Novotney and Chris Callison-Burch. Cheap, fast and good enough: Automatic speech recognition with non-expert transcription. In *Proc. of HLT NAACL*, 2010.

[111] David Oleson, Alexander Sorokin, Greg P. Laughlin, Vaughn Hester, John Le, and Lukas Biewald. Programmatic gold: Targeted and scalable quality assurance in crowdsourcing. In *Proc. of HCOMP*, 2011.

[112] Tim Paek, Yun-Cheng Ju, and Christopher Meek. People watcher: A game for eliciting human-transcribed data for automated directory assistance. In *Proc. of INTERSPEECH*, 2007.

[113] Aditya Parameswaran, Hyunjung Park, Hector Garcia-Molina, Neoklis Polyzotis, and Jennifer Widom. Deco: Declarative crowdsourcing. Technical report, Stanford University, 2012.

[114] Gabriel Parent and Maxine Eskenazi. Speaking to the crowd: looking at past achievements in using crowdsourcing for speech and predicting future challenges. In *Proc. of INTERSPEECH*, 2011.

[115] Jeremy Peckham. A new generation of spoken dialogue systems: results and lessons from the Sundial project. In *Proc. of EUROSPEECH*, 1993.

[116] Fernando C. N. Pereira and Rebecca N. Wright. Finite-state approximation of phrase structure grammars. In *Proc. of ACL*, 1991.

[117] John F. Pitrelli, Cynthia Fong, Suk H. Wong, Judith R. Spitz, and Hong C. Leung. PhoneBook: a phonetically-rich isolated-word telephone-speech database. In *Proc. of ICASSP*, 1995.

[118] Alexander J. Quinn and Benjamin B. Bederson. Human computation: a survey and taxonomy of a growing field. In *Proc. of CHI*, 2011.

[119] Lawrence Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *IEEE*, 77(2):257–286, 1989.

[120] Antoine Raux, Dan Bohus, Brian Langner, Alan W Black, and Maxine Eskenazi. Doing research on a deployed spoken dialogue system: One year of lets go! experience. In *Proc. of INTERSPEECH*, 2006.

[121] Manny Rayner, Ian Frank, Cathy Chua, Nikos Tsourakis, and Pierrette Bouillon. For a fistful of dollars: using crowd-sourcing to evaluate a spoken language CALL application. In *Proc. of Speech and Language Technology in Education Workshop (SLaTE)*, 2011.

[122] Joel Ross, Lilly Irani, M. Six Silberman, Andrew Zaldivar, and Bill Tomlinson. Who are the crowdworkers? Shifting demographics in Mechanical Turk. In *Proc. of CHI*, 2010.

[123] Conrad Sanderson and Kuldip K. Paliwal. Effect of different sampling rates and feature vector sizes speech recognition performance. In *Proc. of IEEE TENCON - Speech and Image Technologies for Computing and Telecommunications*, 1997.

[124] Lauren A Schmidt. Crowdsourcing for human subjects research. In *Proc. of Crowd-Conf*, 2010.

[125] Tanja Schultz, Alan W Black, Sameer Badaskar, Matthew Hornyak, and John Kominek. Spice: Web-based tools for rapid language adaptation. In *Proc. of INTERSPEECH*, 2007.

[126] Terrence J. Sejnowski and Charles W. Rosenberg. Parallel networks that learn to pronounce English text. *Complex Systems*, 1:145–168, 1987.

[127] Stephanie Seneff. Response planning and generation in the MERCURY flight reservation system. *Computer Speech and Language*, 16:283–312, 2002.

[128] Stephanie Seneff. Reversible sound-to-letter/letter-to-sound modeling based on syllable structure. In *Proc. of HLT NAACL*, 2007.

[129] Stephanie Seneff, Ed Hurley, Raymond Lau, Christine Pao, Philipp Schmid, and Victor Zue. Galaxy-II: A reference architecture for conversational system development. In *Proc. of ICSLP*, 1998.

[130] Jongho Shin, Shrikanth Narayanan, Laurie Gerber, Abe Kazemzadeh, and Dani Byrd. Analysis of user behavior under error conditions in spoken dialogs. In *Proc. of ICSLP2002*, 2002.

[131] M. Six Silberman, Joel Ross, Lilly Irani, and Bill Tomlinson. Sellers' problems in human computation markets. In *Proc. of HCOMP*, 2010.

[132] Size of wikipedia. http://en.wikipedia.org/wiki/Wikipedia:Size_of_Wikipedia, accessed March, 2012.

[133] Rion Snow, Brendan O'Connor, Daniel Jurafsky, and Andrew Y. Ng. Cheap and fast—but is it good?: evaluating non-expert annotations for natural language tasks. In *Proc. of EMNLP*, 2008.

[134] Yik-Cheung Tam and Tanja Schultz. Dynamic language model adaptation using variational bayes inference. In *Proc. of INTERSPEECH*, 2005.

[135] Oriol Vinyals, Li Deng, Dong Yu, and Alex Acero. Discriminative pronounciation learning using phonetic decoder and minimum-classification-error criterion. In *Proc. of ICASSP*, 2009.

[136] Luis von Ahn. reCAPTCHA: Human-Based Character Recognition via Web Security Measures. *Science*, 321(5895):1465–1468, September 2008.

[137] Luis von Ahn and Laura Dabbish. Labeling images with a computer game. In *Proc. of CHI*, 2004.

[138] Luis von Ahn, Mihir Kedia, and Manuel Blum. Verbosity: a game for collecting common-sense facts. In *Proc. of CHI*, 2006.

[139] M. Walker, J. Aberdeen, J. Boland, E. Bratt, J. Garofolo, L. Hirschman, A. Le, S. Lee, S. Narayanan, K. Papineni, B. Pellom, J. Polifroni, A. Potamianos, P. Prabhu, A. Rudnicky, G. Sanders, S. Seneff, D. Stallard, , and S. Whittaker. DARPA communicator dialog travel planning systems: The June 2000 data collection. In *Proc. of EUROSPEECH*, 2001.

[140] Marilyn Anne Walker, Rebecca Passonneau, and Julie E. Boland. Quantitative and qualitative evaluation of DARPA communicator spoken dialogue systems. In *Proc. of ACL*, pages 515–522, Morristown, NJ, USA, 2001. Association for Computational Linguistics, Association for Computational Linguistics.

[141] Willie Walker, Paul Lamere, Philip Kwok, Bhiksha Raj, Rita Singh, Evandro Gouvea, Peter Wolf, and Joe Woelfel. Sphinx-4: A flexible open source framework for speech recognition. Technical report, Carnegie Mellon University, 2004.

[142] H.C. Wang. Mat – a project to collect mandarin speech data through networks in taiwan. *International Journal of Computational Linguistics and Chinese Language Process*, 12(1):73–89, February 1997.

[143] Stanley Wang. Using graphone models in automatic speech recognition, 2009. M.Eng. thesis, Department of Electrical Engineering and Computer Science, MIT.

[144] Peter Welinder, Steve Branson, Serge Belongie, and Pietro Perona. The multidimensional wisdom of crowds. In *Proc. of NIPS*, 2010.

[145] Barbara Wheatley and Joseph Picone. Voice across america: Toward robust speaker-independent speech recognition for telecommunications applications. *Digital Signal Processing*, 1(2):45 – 63, 1991.

[146] Maria K Wolters, Karl B Isaac, and Steve Renals. Evaluating speech synthesis intelligibility using amazon mechanical turk. In *Proc. of the Speech Synthesis Workshop*, 2010.

[147] Yushi Xu and Stephanie Seneff. Dialogue management based on entities and constraints. In *Proc. of SIGdial*, 2010.

[148] Chia ying Lee and James R. Glass. A nonparametric bayesian approach to acoustic model discovery. To be presented at *ACL*, 2012.

[149] Steve Young. Still talking to machines (cognitively speaking). In *Proc. of INTERSPEECH*, 2010.

[150] Dong Yu, Balakrishnan Varadarajan, Li Deng, and Alex Acero. Active learning and semi-supervised learning for speech recognition: A unified framework using the global entropy reduction maximization criterion. *Computer Speech and Language*, 24(3):433–444, July 2010.

[151] Victor Zue. On organic interfaces. In *Proc. INTERSPEECH*, 2007.

[152] Victor Zue, Stephanie Seneff, and James Glass. Speech database development at MIT: TIMIT and beyond. *Speech Communication*, 9(4):351 – 356, August 1990.

[153] Victor Zue, Stephanie Seneff, James Glass, Joe Polifroni, Christine Pao, Timothy J. Hazen, and Lee Hetherington. JUPITER: a telephone-based conversational interface for weather information. *IEEE Transactions on Speech and Audio Processing*, 8(1):85 –96, jan 2000.