Efficient Correction Interfaces for Speech Recognition

Keith Vertanen

Darwin College

Inference Group

Cavendish Laboratory

University of Cambridge

A dissertation submitted in candidature for the degree of

Doctor of Philosophy

April 2009

Abstract

The recognition of speech by computers is a challenging task and recognition errors are ultimately unavoidable. Error correction is thus a crucial part of any speech recognition interface. In this thesis, I look at how to improve the correction process in speech recognition.

Before errors can be corrected, they must first be detected. I look at improving error detection by visualizing the recognizer's confidence in each word. After detection, errors must be corrected. I examine three distinct ways of correcting errors: by speech, by touch, and by navigation. I also look at applying touch-based correction to the problem of entering web search queries by voice.

I tested several new correction interfaces in a series of user studies. I found that using my touch-based interface, Parakeet, users wrote at 13 words per minute while walking outdoors. Using my navigation-based interface, Speech Dasher, users wrote at 40 words per minute using only speech and a gaze tracker. Using a system I built for entering web search queries by voice, users entered queries in about 18 seconds while walking indoors. In these user studies, the speech recognizer's initial error rate, prior to user correction, was high. But by using a good correction interface, I found users were able to complete their tasks easily and efficiently.

Declaration

I hereby declare that my dissertation entitled "Efficient Correction Interfaces for Speech Recognition" is not substantially the same as any that I have submitted for a degree or diploma or other qualification at any other University.

I further state that no part of my dissertation has already been or is being concurrently submitted for any such degree or diploma or other qualification.

Except where explicit reference is made to the work of others, this dissertation is the result of my own work and includes nothing which is the outcome of work done in collaboration. This dissertation does not exceed sixty thousand words in length.

Date: .... Signed: ....

Keith Vertanen
Darwin College
Cambridge
September 24th, 2009

Acknowledgements

First and foremost, I would like to thank my supervisor David. Throughout this work he has been a constant source of good ideas and encouragement. It's been grand and I've learned a lot.

Thanks to Inference Group members past and present for helpful advice and conversations over the years. In particular, I'd like to acknowledge Per Ola for his help and advice, for the productive collaborations, and for being a valuable sounding board for ideas. To Oli my writing-up buddy, it's been great – but let's not do it again. Thanks to Carl for his Python wizardry. Thanks to Chris and *001* (Phil) for their help with Dasher. Thanks to Tamara, *002* (Philip), and *003* (Philipp) for valuable feedback on my interfaces.

Thank you to Matt Stuttle and Phil Woodland for their help with the HTK toolkit. Thanks to David Huggins-Daines for his help with the PocketSphinx recognizer. Thanks to Anna Langley for digging up corpora from the CUED archives. Thanks to Alan Blackwell and the other members of the Rainbow Group for helpful advice.

What lies inside would be more painful to read if it weren't for the tireless efforts of my personal proofreading team. Thanks to: David MacKay, Per Ola Kristensson, Tony Robinson, Dave English, Catherine Breslin, Philip Sterne, Tamara Broderick, Oliver Stegle, and Fergal O'Shea.

A huge thank you to all the folks who took time out of their busy lives to take part in my many user trials. It really did take a village.

Thanks to Tobii for loaning us a gaze tracker. Thanks to Nuance for donating a copy of the Dragon NaturallySpeaking SDK. Thanks to

Nokia for donating some of the devices used in my research. Thanks to Gaggia for the caffeine.

None of this would have been possible without the Clerk Maxwell scholarship provided by the Cavendish Laboratory. I am also indebted to Nokia for providing additional financial support.

To my family and friends back in the US. I appreciate your patience and support throughout my (seemingly) never-ending education. Sorry for missing so much while I was here in Cambridge. To Mom, Dad, Michelle, Tim, Melissa, Chris, and Kristian – thanks for all the encouragement and for looking after me on my many visits.

To all the friends I've made in Cambridge – thank you for the good times, the many adventures, and for making my time here so great.

Contents

Chapter 1

Introduction

This thesis is about improving speech recognition interfaces by making the correction process easier and more efficient. I will look at how to best utilize the rich information available to the recognizer to better enable user-guided correction. I will show that speech recognition does not necessarily need to be perfect to be useful. I will demonstrate that even at high initial recognition error rates, an appropriate interface allows users to complete their tasks quickly.

1.1 The Problem

Writing text on a computer is no longer an isolated activity reserved for a few skilled office workers. It is something that most people want to or need to do everyday. There are essays to write for school, emails to send at work, and text messages to send to friends. But all these activities require some method to input our thoughts into the computer. The keyboard is currently the de facto standard for text input on a computer.

For most users, the keyboard is an easy-to-use and effective tool for inputting text. Unfortunately, the keyboard is not a solution in all cases. Some people have a temporary or permanent disability which prevents them from using a keyboard. Some computing devices are small and may lack the space for a full-size keyboard. In such cases, an alternate text entry method is required. In this thesis, I explore one such alternative: speech recognition.

While there has been great progress in improving accuracy, speech recognition is still not perfect. Recognition errors will sometimes occur and those errors will need correction by users. This user-guided correction of speech recognition errors is the central problem I address in this thesis.

1.2 Important Themes

In this thesis, I propose and test a number of novel speech recognition interfaces. These interfaces and the evaluations thereof, all share a number of important themes:

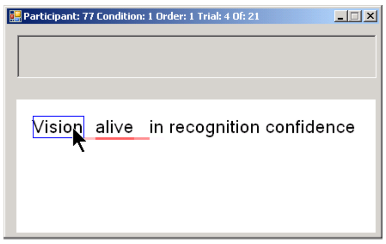
- **Use all information** – Speech recognizers work hard to search through a multitude of possibilities before deciding on *the* best recognition result. But when this best result is wrong, there is still useful information that can be gleaned from the recognizer's search. My interfaces will leverage this information to better enable user-guided correction.
- **Don't be a slave to the numbers** – My interface designs were often informed by computational experiments carried out on recorded audio. But I did not let the results of those experiments dictate my designs. In several cases, I made choices that were suboptimal from the standpoint of the numbers, but in my judgment resulted in interfaces that were easier to understand and use.
- **Understand the envelope** – Knowing how well an interface does at one particular recognition operating point is not that informative. Recognition accuracy is a moving target. It varies depending on the current state of recognition technology, as well as on the specifics of the recognition task. In this work, I try understand the performance *envelope* of each design. This envelope shows how well the interface performs at a variety of recognition error rates. This provides a clearer picture of how an interface might perform given a newer recognizer, a harder task, etc.

1.3 Overview and Contributions

Each chapter discusses a different topic or interface related to correcting speech recognition errors. Each chapter stands on its own and the chapters can be read in any order. I assume the reader has a basic understanding of speech recognition. For an overview of speech recognition, see appendix A.

Here is a short summary and a list of the highlights of each chapter:

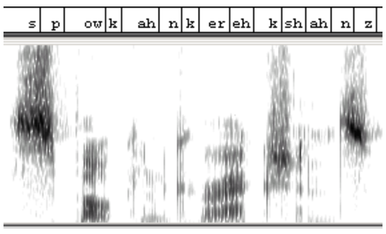
Chapter 2: Visualizing Recognition Confidence


Participant: 77 Condition: 1 Order: 1 Trial: 4 Of: 21
Vision alive in recognition confidence

A recurring idea in the speech field has been to convey to the user the recognizer's confidence in its result. I test whether underlining likely errors in a shade of red makes users faster or more accurate at detecting errors.

- Underlining likely errors did not make users faster or more accurate overall.
- Users found correctly underlined errors more often, but tended to miss errors that failed to be underlined.

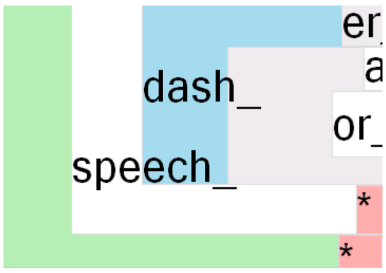
Chapter 3: Spoken Corrections



I analyze how users change their speech in the face of recognition errors. I study the impact these changes have on recognition accuracy using three different speech recognizers. I look at how to better recognize corrections consisting of a single word or part of a sentence.

- Users significantly alter their speech in response to recognition errors.
- Hyperarticulate speech was no harder to recognize than normal speech.
- Recognition of short, single word or partial-sentence corrections was difficult.
- Adapting the model to short corrections improved accuracy by 13% relative.

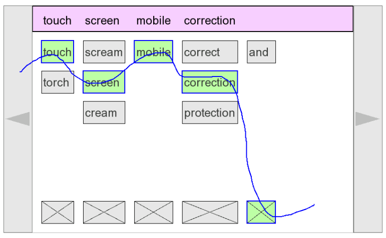
Chapter 4: Speech Dasher



I describe Speech Dasher, an interface that allows users to easily perform corrections by navigating through the speech recognition hypothesis space. I demonstrate the effectiveness of Speech Dasher in a user study in which participants wrote using only speech and their gaze (obtained via a gaze tracker).

- Adding speech to Dasher doubled users' writing speed.
- Users wrote at 40 words per minute using just speech and gaze. This was despite an initial recognition word error rate of 22%.
- Speech Dasher degraded gracefully in the face of errors. Even a user with a high recognition error rate of 47% still wrote slightly faster with Speech Dasher compared to using Dasher without speech.

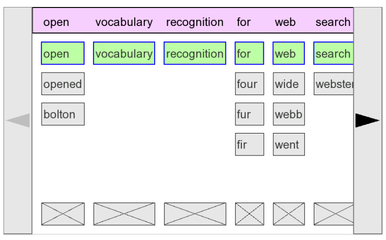
Chapter 5: Touch-Screen Mobile Correction



I describe Parakeet: a touch-screen interface designed for efficient mobile text entry using speech. Users correct errors by selecting words from a confusion network or by using an on-screen keyboard. In a user study, participants used Parakeet both while seated indoors and while walking outdoors.

- In computational experiments, Parakeet's confusion network interface allowed correction of over half of all recognition errors.
- Users took advantage of the confusion network interface to correct errors whenever possible.
- Users wrote at 13 words per minute while walking outdoors. This was despite significant recognition delays and a word error rate of 26%. As a reference point, users of T9 wrote at 16 words per minute while seated indoors after 15 sessions [162].
- Without recognition delays, users would have written at 26 words per minute while walking outdoors.

Chapter 6: Open Vocabulary Recognition for Web Search



I describe methods to handle large vocabularies by automatically generating pronunciations and by allowing recognition of novel words. I built a system to recognize spoken web search queries. In a user study, participants used my system to enter queries on a mobile device while walking.

- Accurate letter-to-phone conversion is possible using simple letter/phone units combined with standard n-gram language modeling techniques. My best model had a phone error rate of 6.5%.
- A new technique incorporating out-of-vocabulary words into word confusion networks helped reduce word error rates.
- A corpus of 10 million search queries was collected over a period of 4 years.
- Using the search query corpus, letter-to-phone conversion, and open vocabulary recognition, the word error rate on spoken web search queries was reduced by 34% relative compared to an in-vocabulary newswire language model.
- Users spoke and corrected web search queries in 18 seconds while walking. As a reference point, *Google* reports web search queries took about 40 seconds to enter from a phone or PDA [79].

Chapter 2

Visualizing Recognition Confidence

2.1 Overview

In a typical speech dictation interface, after the user speaks their desired text, the best recognition hypothesis is displayed as normal, unannotated text. The user must then proofread this unannotated text and spot any recognition errors. Detecting recognition errors can sometimes be difficult as errors often involve subtle changes in spelling (e.g. "were" versus "where") or the inclusion or exclusion of small words (e.g. "a" or "to"). An interesting question is whether users would perform corrections faster or more accurately if the recognizer provided feedback about locations that were predicted to be possible recognition errors. In this chapter, I investigated providing such feedback about likely word errors using shaded, red underlining (see figure 2.1). The intensity of the red underlining was based on a measure of the recognizer's confidence in each word in the best hypothesis. I created a speech interface that conveyed information about low-confidence words to the user. I then conducted a study to investigate if confidence visualization improved users' detection of errors.

While there have been numerous studies in which users dictate text and cor-

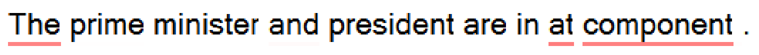

The prime minister and president are in at component .

Figure 2.1: Example of a recognition result with red underlining for low-confidence words. The actual sentence was "The prime minister and president are in good company".

rect errors using unannotated recognition results (e.g. [45; 80; 86; 92]), there are few studies that have used any sort of visualization of recognizer confidence. Two studies that have looked at confidence visualization for correcting text are Suhm et al. [139] and Endo et al. [47]. Suhm et al. found that confidence visualization did not speed up corrections while Endo et al. found that confidence visualization did slightly speed up corrections. Both studies measured the time it took users to both detect and correct errors. Unlike these previous studies, here I focused on the first part of the correction problem only: detecting errors. This has the advantage of eliminating the often time-consuming correction process. Performing corrections can easily dominate a user's task completion time, making any speed improvement offered by confidence visualization difficult to measure. There are several additional differences in my study compared to these two past studies. In Suhm's study, users experienced a high word error rate (WER) of 25%. In contrast, I tested users at a much lower WER of 8%. This lower WER is similar to what users might experience using a current commercial recognizer. In Endo's study, users experienced simulated recognition errors and were artificially forced to correct all errors. I tested users on real recognition results and did not require that users locate all errors.

The rest of this chapter is structured as follows. First, I describe the system I built to investigate confidence visualization. Second, I describe the evaluation I conducted to assess if confidence visualization benefited users. Third, I present my results. Fourth, I discuss limitations of my study and point to avenues for future research.

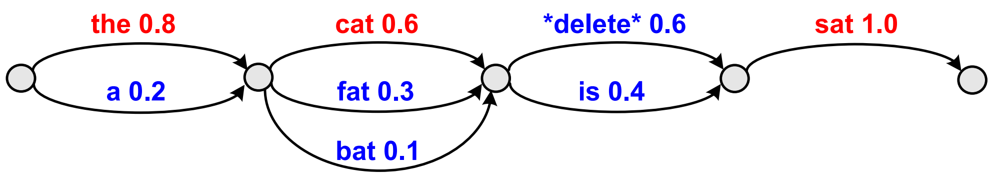


Figure 2.2: Example word confusion network with four clusters. The best recognition result is "the cat sat". Confidence scores for the word hypotheses are shown on the edges.

2.1.1 Publication Note

The work presented in this chapter was joint work with Per Ola Kristensson. The information contained in this chapter was published in part in the paper "On the Benefits of Confidence Visualization in Speech Recognition" at the ACM Conference on Human Factors in Computing Systems (CHI 2008) [151].

2.2 Confidence Visualization

In this section, I describe the details of how I provided feedback to the user about recognition confidence.

2.2.1 Word Confusion Networks

As a measure of confidence, I used the word posterior probabilities given by a word confusion network (WCN) [104]. A WCN is a time-ordered sequence of clusters where each cluster contains competing words and their posterior probabilities (figure 2.2). The probabilities in a cluster sum to one. A WCN is built using the time- and phonetic-overlap of words in a recognizer's word lattice output. Clusters in a WCN can contain a special "delete" word which represents the hypothesis that nothing was said in that cluster. The best recognition result can be obtained by taking the highest probability edge in each cluster.

Error type	Before correction	After correction
Substitution	The fat sat	The cat sat
Insertion	The cat is sat	The cat sat
Deletion	The sat	The cat sat

Table 2.1: Examples of the three types of errors and their visualization in my system.

2.2.2 Substitution and Insertion Errors

The first class of recognition errors my system visualizes are substitution and insertion errors. A substitution error is a word mistakenly recognized as another word. An insertion error is an extra word which was mistakenly added. Both errors result in visible words in the user's display that need to be replaced in the case of a substitution error, or deleted in the case of an insertion error.

Potential substitution errors and insertion errors were underlined using a 3-pixel wide red line (see table 2.1). The red color of the line was made more intense for words that were more likely to be a recognition error. The color was calculated as a red-green-blue (RGB) color triplet using linear interpolation:

$$\text{color} = (1.0, c, c) \tag{2.1}$$

where c is a word's confidence score which is in the range $[0, 1]$.

2.2.3 Deletion Errors

The second class of recognition errors my system visualizes are deletion errors. A deletion error occurs when a word is missing from the recognition result. A problem with deletion errors is they do not result in a word appearing in the best recognition hypothesis. This can result in deletion errors going unvisualized if the confidence scoring method used for visualization is based solely on words in the best result (as is the case for methods such as n-best homogeneity [27], hypothesis density [84], acoustic stability [50], or word graph posterior probabilities [84]). Instead, by using a WCN for confidence scoring, I was able to use the presence of the delete word and its associated probability to provide information about when deletion events may have occurred. For example, in figure 2.2, the recognizer's best hypothesis is that there was no word between "cat" and "sat". But the probability of the delete word between "cat" and "sat" was 0.6 with the word "is" also having a large probability of 0.4. This provides evidence that a deletion event may have occurred here.

I developed a technique for visualizing deletion errors. Possible deletion errors were visualized as empty horizontal gaps at the position where the deleted words should have been in the text (see table 2.1). These horizontal gaps make it easier for the user to select deletion errors as Fitts' law [52] tells us that larger targets are faster to click than smaller targets (assuming the same amplitude). Also, an error with a low confidence is more likely to be a true error and thus a larger gap makes the potential error more noticeable. In my system, I made the width (in pixels) of the horizontal gap denoting a deletion error a function of the confidence score c:

$$w = 1 + 15 \cdot (1 - c)\,. \tag{2.2}$$

Besides the empty horizontal gap, I also indicated possible deletion errors by underlining the horizontal gap in red in the same manner as substitution and insertion errors.

2.3 Speech Recognition

In this section, I describe the details of the speech recognizer I used for my confidence visualization experiment.

2.3.1 Acoustic and Language Model

I used the CMU Sphinx speech recognizer. I trained a UK-English acoustic model on 16 hours of WSJCAM0 data [124] using my Sphinx training recipe described in [147] (slightly modified for UK-English). I used a 3-state left-to-right HMM topology using cross-word triphones.

I parameterized audio into a 39-dimensional feature vector consisting of 12 Mel-frequency cepstral coefficients plus the 0^{th} cepstral, deltas and delta deltas. The model had 3000 tied-states with each state having 8 continuous Gaussians with diagonal covariance matrices. I used the CMU phone set without stress markings (39 phones plus silence).

I trained a trigram language model using newswire text from the CSR-III text corpus [64] (222M words). I trained the language model with interpolated modified Knesser-Ney smoothing and a count cutoff of 1 for unigrams, 1 for bigrams, and 3 for trigrams. The language model's vocabulary was the top 5K words occurring in the corpus and included verbalized commas and periods. The resulting language model had 3.2M bigrams and 5.7M trigrams. I created the recognizer's pronunciation dictionary from the BEEP UK-English dictionary [123] with additional words and pronunciation variants added from the CMU US-English dictionary [25].

My software combined PortAudio [15] for audio capture, Sphinx-3 [128] for speech decoding, and SRILM [138] for lattice pruning and word confusion network creation. I streamed audio sampled at 16 kHz to the recognizer as soon as the user enabled the microphone. I used cepstral mean normalization based on a prior window of audio.

2.3.2 Gender and Speaker Adaptation

I created gender-dependent models by adapting the model's means using maximum likelihood linear regression (MLLR) [96] with 40 regression classes (one class for each of the 40 base phones). This was followed by maximum a-posteriori

(MAP) adaptation [59] of the means, variances, mixture weights and transition probabilities.

I further adapted the gender-dependent models to each participant's voice using 29 sentences collected at the start of each participant's session. For adaptation material, I used 17 phonetically diverse sentences from the WSJ corpus [118]. In addition, I had each participant read 12 sentences which were similar in style to the target paragraphs used in the main study. This not only gave me more adaptation data, but also allowed participants to practice reading text in which they had to verbalize commas and periods. I created speaker-dependent acoustic models by using MLLR-adaptation with 1 regression class followed by MLLR-adaptation with 7 regression classes. I created the 7 regression classes by dividing the 40 base phones according to their place of articulation.

2.3.3 Word Error Rate Target

In order to gather data relevant to real-world applications, I wanted my experimental system to have a WER similar to what novices might encounter using a modern commercial recognizer. Past user studies using commercial speech recognizers for dictation-style tasks have reported WERs of 6–11% [81], 7–15% [45], and 15% [86].

These prior studies used somewhat outdated versions of commercial recognizers. I wanted to know what error rate a novice might expect using a current state-of-the-art recognizer. So I conducted my own testing with Nuance Dragon NaturallySpeaking v9. For my testing, I used a corpus of speech collected in prior work [148] (see also chapter 3). This corpus consisted of 24 novices who had each adapted Dragon's acoustic model to their own voice using the "Talking to your computer" enrollment text. For test data, I used utterances from my corpus in which participants had spoken 42 sentences from the San Jose Mercury sentences in the WSJ1 si_dt_s2 (Spoke 2) test set [5]. Each participant recorded every sentence twice yielding a total of 2016 test set utterances. My chosen subset of WSJ1 si_dt_s2 was somewhat hard by design as it included some sentences with proper names and uncommon vocabulary. My subset had an out-of-vocabulary

(OOV) rate of 6.9% using the WSJ 20K-vocabulary and 3.1% using the WSJ 64K-vocabulary. Using participants' speaker-dependent acoustic models and Dragon's maximum accuracy setting, I found an overall WER of 7.9%.

Given the error rates of past studies and my own experiment with Dragon, I decided to target a WER of around 8% for my study. Prior to the study, I tested my Sphinx recognition setup to ensure I was operating at close to my WER target. I tested recognition on 19 speakers from the WSJCAM0 5K-vocabulary test set (si_dt_5b). I adapted each speaker's acoustic model using 35 utterances from that speaker. On this test set, my recognizer had a 9.5% WER, operating at 0.6 $\times$ real-time (recognition took 0.6 times as long as the total audio time). During my experiment, to be described shortly, my actual study participants had a WER of 8.5%.

2.4 User Study

In this section, I describe the user study I conducted to evaluate whether confidence visualization was beneficial to users. I used a within-subjects experimental design with two conditions:

1. **Baseline** – Words were presented without confidence visualization.
2. **Visualization** – Confidence scores from the recognizer were used to underline likely errors in red.

2.4.1 Participants and Apparatus

I recruited 16 volunteer participants from the university campus (13 men, 3 women). Their ages ranged between 22 and 33 (mean = 26.6, sd = 2.7). They were paid £5 for participating in a single 45-minute study session. Participants used a Dell laptop with a 15" screen with a resolution of 1400 $\times$ 1280. They wore a Plantronics DSP-400 USB headset microphone.

2.4.2 Method and Setup

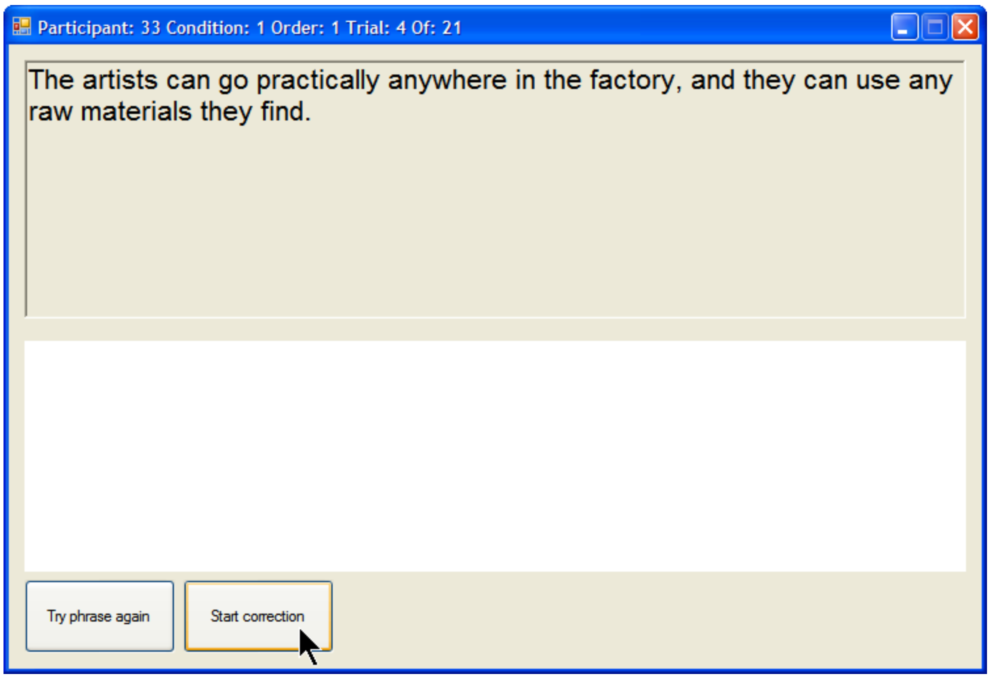
Each participant first trained the speech recognizer for about 10 minutes. The participant then proceeded to their first condition (either the visualization or baseline condition depending on their order). The order of conditions was counterbalanced across participants.

In each condition, the participant spoke short paragraphs consisting of 1 to 2 sentences from the set-aside directory of the CSR-III newswire corpus. These sentences were excluded from language model training. The average paragraph length was 20 words, in line with a previous study by Suhm [139]. All paragraphs used only words that were in the 5K-vocabulary of the speech recognizer. I chose a small in-vocabulary task so my research recognizer would have a WER similar to current commercial recognizers. The order in which participants received paragraphs was counterbalanced across participants. In each condition, the participant did 1 practice and 20 trial paragraphs. During the first practice paragraph, an experimenter described how to use the interface. The practice paragraph was excluded from all analysis.

After each condition, the participant filled in a brief questionnaire. Between the two conditions, the participant was given a 5-minute break. After the last condition, the participant filled in a final questionnaire.

In both conditions, I presented the target paragraph to the participant in a text box (figure 2.3). To encourage reading of the paragraph before speaking, I displayed the paragraph in a teleprompter-style (each character was added to the text box after a small time delay). After the entire paragraph was displayed, the participant pressed a BEGIN SPEAKING button to start streaming audio to the recognizer. After finishing speaking, the participant pressed a STOP SPEAKING button.

After a small recognition delay of 2.3 s $\pm$ 1.9 s, a beep alerted the participant that recognition was complete and a BEGIN CORRECTING button appeared. In case the participant had misspoken, a TRY AGAIN button allowed the participant to dictate the paragraph again. When the participant pressed the BEGIN COR-

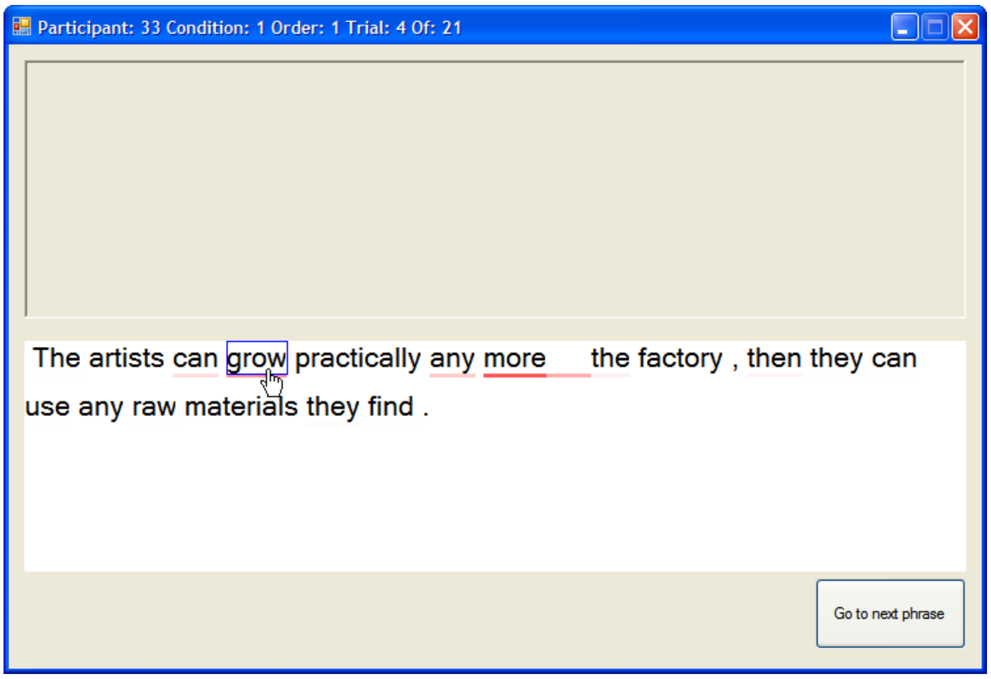

Participant: 33 Condition: 1 Order: 1 Trial: 4 Of: 21

The artists can go practically anywhere in the factory, and they can use any raw materials they find.

Figure 2.3: Screenshot of the experimental interface. The participant has just dictated the text in the top text box. But the recognition result is not displayed in the bottom text box until the START CORRECTION button is pressed.

RECTING button, the target paragraph was hidden and the recognition result was displayed. In the visualization condition, red underlining denoted possible recognition errors (figure 2.4). In all other aspects, the interaction in each condition was identical.

The participant was instructed to "quickly and accurately" indicate all errors in the recognized text by clicking on them. When an error was clicked, it was automatically corrected. This was possible because the software knew the correct text. If something was clicked that was already correct, it was left unchanged (but the click was logged as a mistake). I emphasize that I used "oracle" knowledge (that is knowledge of the target paragraph) only to perform automatic *correction*; the recognized text and confidence scores were obtained from real recognition results from the participant's audio. Furthermore, all aspects of the confidence visualization display used only the recognition output and made no use of oracle knowledge.

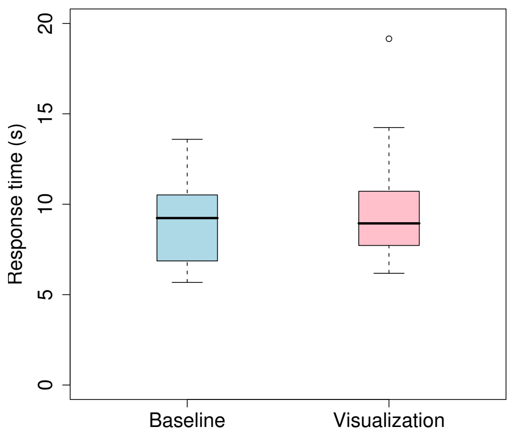


The artists can grow practically any more the factory , then they can use any raw materials they find .

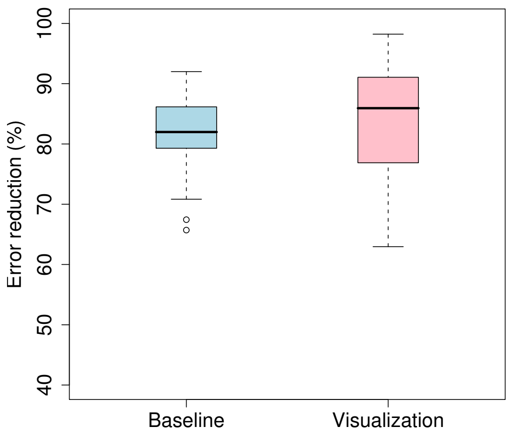
Figure 2.4: Screenshot of a recognition result with confidence visualization. Words with low-confidence are underlined in a shade of red. The more intense the red underlining, the higher the chance that word is an error. When the user positions the mouse over a word, the word is outlined with a blue box. If the user clicks on a word that is an actual error, the word is automatically replaced with the correct word.

I did not allow manual correction of errors since I was interested in whether visualization enabled faster error *detection*. If I had allowed corrections, the time required to perform corrections would likely have dominated. While in principle, a participant's correction time could be deduced by observing mouse and keyboard activity, it introduces a potential confounding random variable. It is possible that while manually correcting an error (e.g. by re-positioning the text caret and erasing the incorrect word), the participant might also discover errors in other parts of the text. This would make it difficult to accurately separate the error detection time from the error correction time.

After the participant corrected a paragraph to the best of his or her ability, the participant pressed the FINISHED CORRECTION button and proceeded to the next paragraph.



(a) Detection time



(b) Error reduction rate

Figure 2.5: The two main metrics analyzed for each condition. Detection time (left) measured how long it took to complete corrections. Error reduction rate (right) measured how many recognition errors were successfully corrected.

2.5 Results and Discussion

In this section, I present and discuss my results from the user study. I give both quantitative measures of user performance and also qualitative feedback obtained from paper questionnaires.

2.5.1 Detection Time

I defined detection time as the duration (in seconds) between the user pressing the START CORRECTION button and pressing the FINISHED CORRECTION button. The mean response time was 9.7 s ± 3.6 s in the visualization condition and 9.0 ± 3.2 s in the baseline condition (figure 2.5a). Repeated measures analysis of variance showed that this difference was not significant ($F_{1,15} = 0.85, p = 0.37$).

2.5.2 Error Reduction Rate

I measured the error reduction rate by taking the number of recognition errors corrected by the participant and dividing by the total number of recognition errors. For example, if the recognizer made 40 errors and the participant corrected 30, the error reduction rate was 75%. I determined the significance of error reduction rates between the conditions by using repeated measures analysis of variance.

Participants reduced 84% $\pm$ 10% of errors in the visualization condition and 81% $\pm$ 7% in the baseline condition (figure 2.5b). The difference in error reduction rate was not significant ($F_{1,15} = 0.79, p = 0.39$). Overall, confidence visualization did not improve participants' ability to detect errors.

However, I was curious why visualization did not help. A possible problem is that confidence scores are imperfect and can lead to mistakes in the confidence visualization. These mistakes can cause real recognition errors to be missed (false-accepts) or cause correct words to be flagged as errors (false-rejects). In my system, a confidence score > 0.9 resulted in so pale an underlining as to be almost imperceptible. I therefore split my errors into two sets: visibly underlined errors (confidence ≤ 0.9), and not visibly underlined errors (confidence > 0.9). At this implicit threshold of 0.9, my false-accept rate was 3.4% (non-visibly underlined words that were errors) and my false-reject rate was 7.7% (visibly underlined words that were correct).

For errors with a confidence > 0.9 (errors that were visibly underlined to participants in the visualization condition), participants' mean error reduction rate was 92% $\pm$ 8.9% in the visualization condition and 80% $\pm$ 9.6% in the baseline condition (figure 2.6a). The 12% increase in participants' ability to reduce errors in the visualization condition was statistically significant ($F_{1,15} = 15.38, p = 0.001$). This means that confidence visualization helped participants detect more low-confidence errors than in the baseline. However, since I found *overall* visualization did not improve participants' error reduction rate, this win must be balanced by a loss somewhere else. Indeed, for errors with a confidence > 0.9, participants' error reduction rate was 82%$\pm$11% in the baseline but only 71% $\pm$ 20%

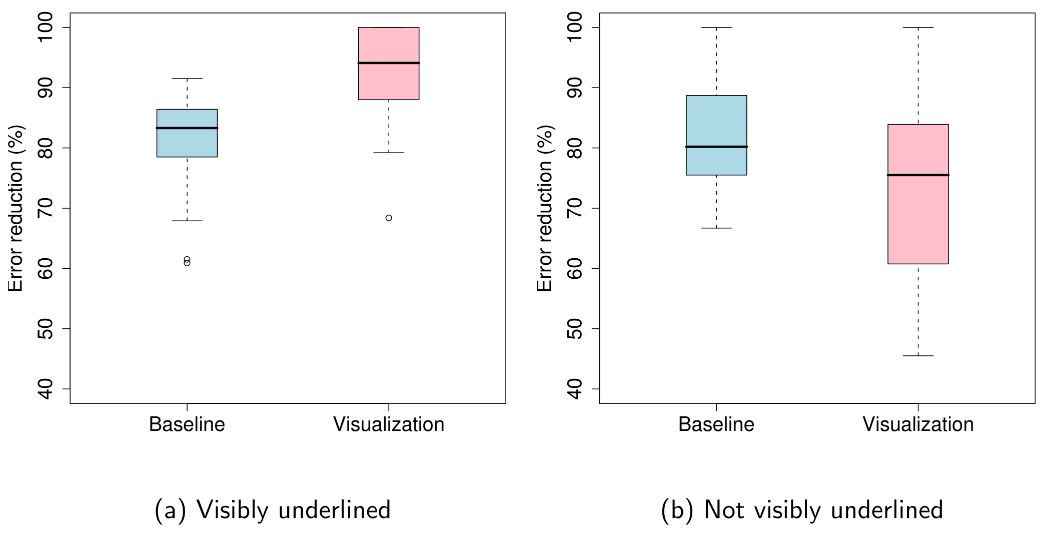


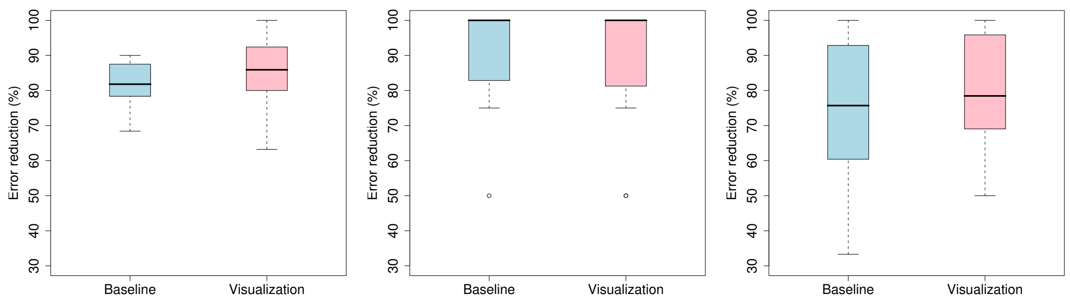
Figure 2.6: Error reduction rate in the two conditions depending on whether errors were visibly underlined (left) or not visibly underlined (right).

in the visualization condition (figure 2.6b). Although this result was not quite significant ($F_{1,15} = 3.40, p = 0.09$), this reduction in users' ability to detect unannotated errors in the visualization condition explains why confidence visualization did not, overall, improve error detection.

I draw two conclusions from the error reduction results. First, confidence visualization did work in the sense that participants took advantage of the red underlining to detect recognition errors. Second, it is plausible that participants trusted confidence visualization and stopped actively verifying non-highlighted words. Their trust in confidence visualization may have caused erroneous words that were not strongly underlined to go undetected.

2.5.3 Error Reduction by Type

During the study, 68% of word errors were substitution errors, 13% were insertion errors, and 19% were deletion errors. As previously discussed, overall, participants' ability to reduce errors was not improved by confidence visualization. But perhaps confidence visualization was helpful for a particular type of recognition



(a) Substitution errors (b) Insertion errors (c) Deletion errors

Figure 2.7: Error reduction rate in the two conditions depending on the type of recognition error.

error. For example, maybe users were much better at finding deletion errors due to my visualization of this error type using a horizontal gap and red underlining. For each type of error, I analyzed participants' error reduction rates between the two conditions. This analysis was post-hoc and therefore used a Bonferroni correction [6].

- **Substitution errors** – Participants reduced 85% ± 9.7% of substitution errors in the visualization condition and 81% ± 6.7% in the baseline condition (figure 2.7a). This difference was not significant ($F_{1,15} = 2.33, p = 0.15$).
- **Insertion errors** – Participants reduced 85% ± 26% of insertion errors in the visualization condition and 86% ± 27% in the baseline condition (figure 2.7b). This difference was not significant ($F_{1,15} = 0.013, p = 0.91$).
- **Deletion errors** – Participants reduced 73% ± 28% of deletion errors in the visualization condition and 73% ± 22% in the baseline condition (figure 2.7c). This difference was not significant ($F_{1,15} = 0.0003, p = 0.99$).

2.5.4 Recognition WER

Recall that I had targeted a WER of around 8% for my study. I came close to this target with participants having a WER of 8.5% (averaged over both conditions).

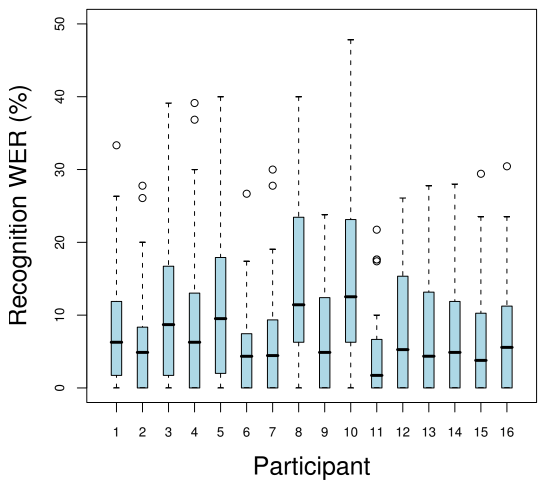


Figure 2.8: Recognition WER (before correction) experienced by each participant in the study over the 40 paragraphs tasks (combining the 20 paragraphs from each condition).

Despite adapting the acoustic model to each participant's voice, I still observed wide variability in error rates (figure 2.8). The best participant had a WER of 4.2% while the worst participant had a WER of 15.9%.

2.5.5 Incorrect Correction Attempts

Before the evaluation, I was concerned that participants might react to the colorfully underlined words by clicking on all of them to remove the underlining. In the visualization condition, such behavior would have caused an inflated number of incorrect clicks (attempts to fix something that was correct). I calculated participants' average number of incorrect clicks per paragraph in each condition. I found participants only occasionally committed incorrect clicks. In the visualization condition, the mean incorrect clicks per paragraph was 0.20 ± 0.24. In the baseline condition, the mean incorrect clicks was 0.27 ± 0.33. This difference was not significant ($F_{1,15} = 1.18, p = 0.30$). Hence it appears participants were careful and did not click on underlined items indiscriminately. As shown in figure 2.9, there was wide variability between participants in how often they clicked

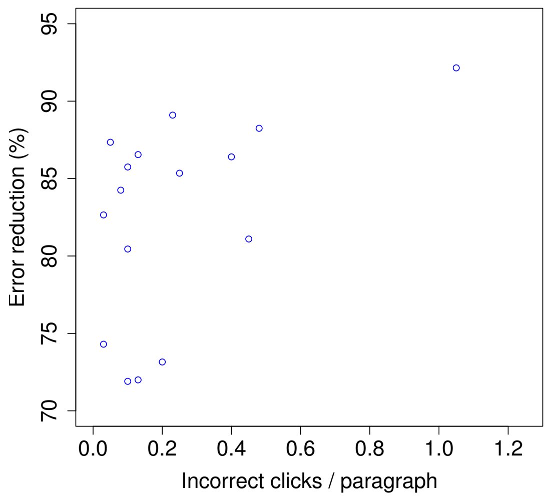


Figure 2.9: This plot shows how often participants clicked incorrectly and the error reduction they achieved. Each point represents a single participant and combined data from both conditions.

incorrectly and the error reduction they achieved.

2.5.6 Influence of WER on Performance

Typically, a recognizer's word error rate has a strong influence on user performance in a speech recognition interface. I looked at how my two main quantitative measures, response time and error reduction rate, changed depending on the recognition WER. For the response time analysis, I divided the total response time by the number of words in the target paragraph.

In figure 2.10, the response time per word for each utterance is shown versus the WER. As expected, the higher the WER, the longer it took to correct the paragraph. Simple linear fits to the data in each condition show that from 0% WER to 10% WER, response times roughly doubled. This shows the strong influence recognition error rate has on the time required to proofread. As demonstrated by the utterances at 0% WER, even completely correct recognitions had

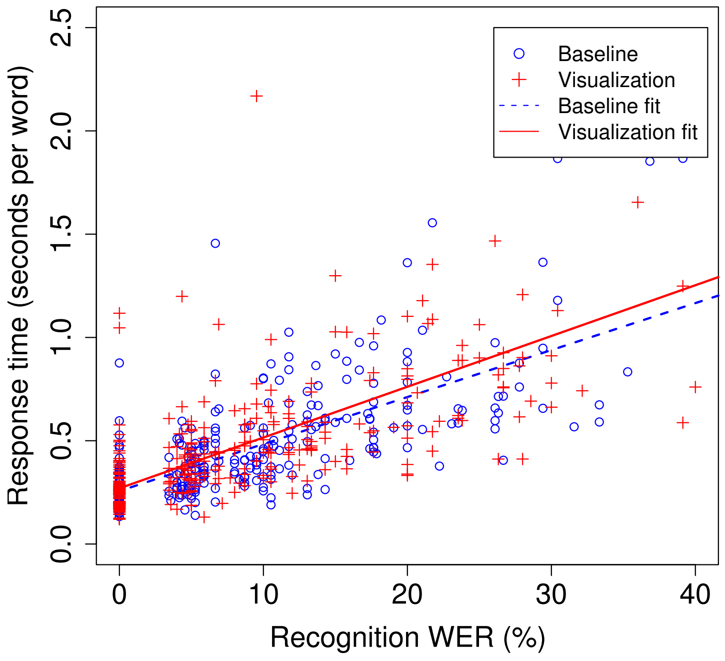


Figure 2.10: This plot shows how the recognition WER (before correction) affected participants' response time (measured in seconds per word). The lines show simple linear fits to the data in each condition.

significant time-related costs, requiring about 0.25 seconds per word for proofreading.

I also analyzed how well participants were able to correct paragraphs based on the recognition error rate. Figure 2.11 shows the after correction WER given the recognition WER. Paragraphs that had no errors corrected (those on the diagonal of figure 2.11) accounted for 7% of all utterances. Only paragraphs with lower WERs (below 20%) were found to have no errors corrected (figure 2.12a). Apparently at higher WERs, the errors became so conspicuous that users noticed at least some of them.

Paragraphs that had all errors corrected (those on the horizontal line of figure 2.11) accounted for 46% of all utterances. As demonstrated by the broad distribution in figure 2.12b, for a wide range of recognition error rates, users were

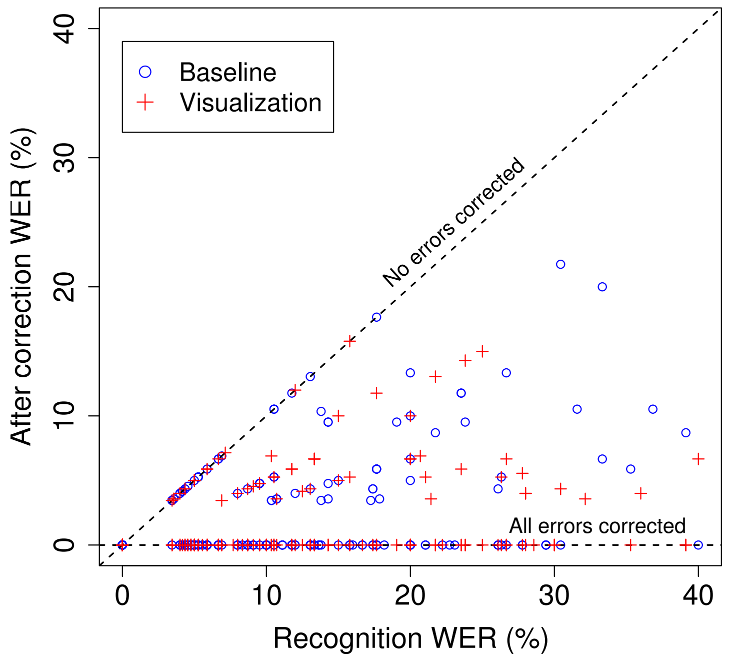


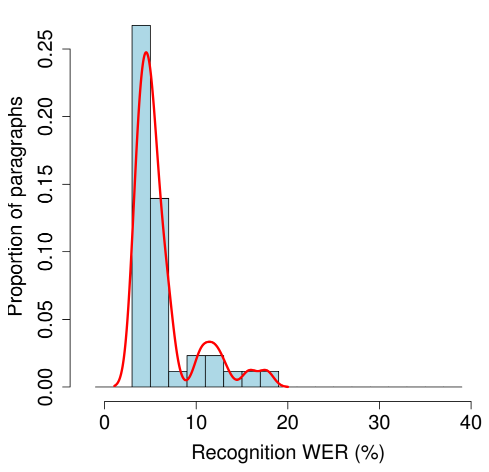
Figure 2.11: This plot shows how the recognition WER affected participants' ability to detect errors. The y-axis shows the remaining WER in each paragraph after correction by the participant. Points on the diagonal line show paragraphs in which no errors were corrected. Points on the horizontal line show paragraphs were all errors were corrected.

successful in correcting all errors.

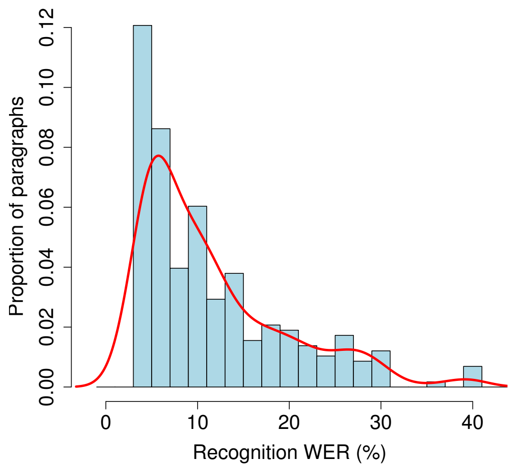
2.5.7 Questionnaire

After each condition, participants filled out a questionnaire which asked them to rate their agreement or disagreement with five statements on a 7-point Likert scale (1 = strongly disagree, 7 = strongly agree). The five statements were:

- **Liked the interface** – "I liked the speech and display interface."
- **Easy to find errors** – "I found it easy to find errors in the text."
- **Looked carefully for errors** – "I had to look carefully at the text area to spot errors."



(a) Error reduction of 0%



(b) Error reduction of 100%

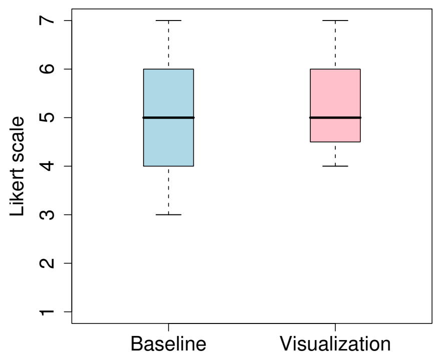
Figure 2.12: Distribution over WER of paragraphs that saw no reduction in errors (left) and paragraphs in which all errors were corrected (right).

- **Accurate speech recognition** – "The speech recognizer accurately recognized my speech."
- **Fun to use** – "It was fun to use the speech recognition and display interface."

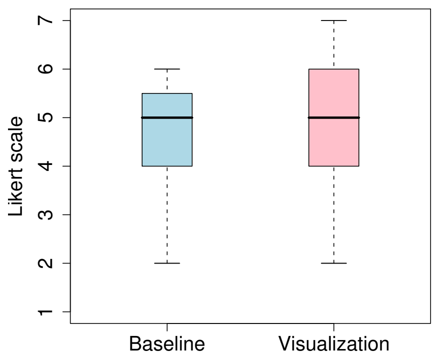
As shown in figure 2.13, I found only small differences in how participants rated the statements between the two conditions. Overall, participants tended to rate the system in both conditions somewhat positively. In particular, speech recognition accuracy was rated highly with a mean score of 5.8 ± 0.5 in the visualization condition and 6.0 ± 0.7 in the baseline condition.

2.5.8 Final Questionnaire

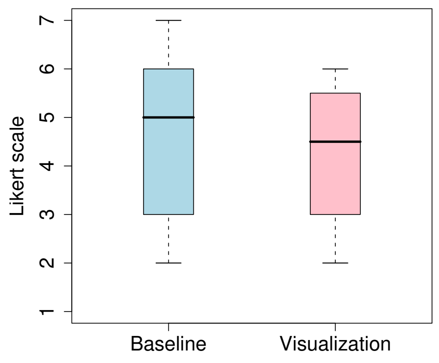
After competing both conditions, participants filled out a final questionnaire. Participants were asked to rate their agreement or disagreement with four statements on a 7-point Likert scale (1 = strongly disagree, 7 = strongly agree). The four statements were:



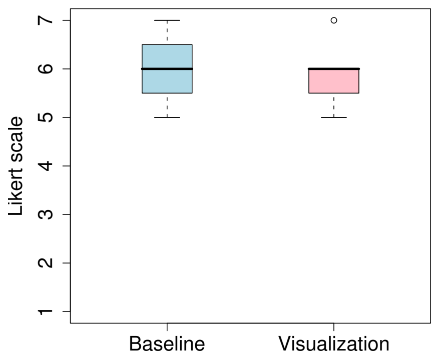
(a) Liked the interface



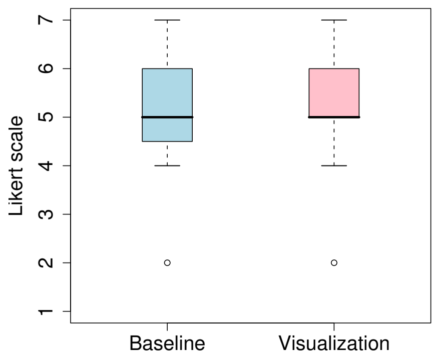
(b) Easy to find errors



(c) Looked carefully for errors



(d) Accurate speech recognition



(e) Fun to use

Figure 2.13: How participants rated five statements about the interface after the completion of each condition (1 = strongly disagree, 7 = strongly agree).

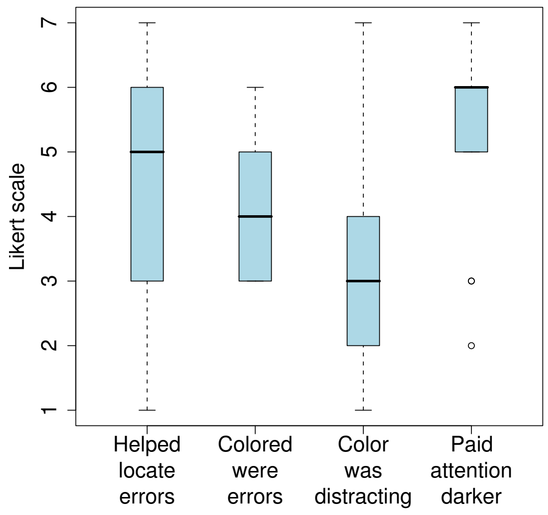


Figure 2.14: How participants at the end of the experiment rated four statements about the confidence visualization (1 = strongly disagree, 7 = strongly agree).

- **Helped locate errors** – "I used the colored underlining to help locate errors."
- **Colored were errors** – "Things with colored underlining were actual errors."
- **Color was distracting** – "The colored underlining was distracting."
- **Paid attention to darker** – "I paid more attention to things underlined with darker shades of red."

As shown in figure 2.14, participants generally found confidence visualization at least somewhat helpful and did not find it overly distracting.

Finally, participants were asked to chose which interface they preferred and also to explain why. In response to the forced choice question, 69% of participants said they preferred the interface with confidence visualization. So, similar to the study by Burke et al. [24], users tended to subjectively report liking confidence visualization. But I urge caution at drawing too strong a conclusion from such a subjective reporting measure. It may be that participants rated the visualization interface highly because they didn't want to criticize my system. Reeves et al. [122] observed humans are reluctant to poorly rate a system if they perceive the system is the fruit of the hard work of the person conducting the study. It could also be that participants erroneously believed they had done a better job of correcting in the visualization interface and thus rated this interface higher.

Here are some reasons people gave as to why they preferred the confidence visualization interface:

- "Drew attention to subtle errors that might otherwise have been missed."
- "Rapidly identified areas that need to be read carefully."
- "You get some sense of feedback."
- "Identified areas that needed to be read carefully."

Here are some reasons people gave for preferring the baseline interface:

- "I ignored the underlining!"
- "I read the whole sentence rather than focusing on single words."
- "The underlining didn't seem to help or be that accurate."

2.6 Limitations

In this section, I discuss some of the limitations of my study and suggest some possible avenues for future research.

2.6.1 WER Operating Point

In this study, participants experienced a recognition WER of around 8%. This was similar to the error rate I found novices experienced using Dragon v9. But an experienced speech recognition user using the latest version of Dragon and using well-trained acoustic and language models might have a substantially lower WER. It is possible that confidence visualization would be more useful at a lower WER operating point. As WERs approach 0%, users may become accustomed to assuming the recognition result is right. In such cases, confidence visualization could prove useful if it was accurate enough to point out the occasional suspect word. But the exact behavior of humans using a confidence visualization system at a lower WER would need to be tested experimentally.

2.6.2 Confidence Threshold

In my study, the point at which the red underlining became noticeably visible imposed an implicit threshold of 0.9 on the probability obtain from the word confusion network. At this threshold, the system had a false-accept rate of 3.4% and a false-reject rate of 7.7%. This is just one possible operating point on the detection error tradeoff (DET) curve [105] depicted in figure 2.15. It is possible that participants would have benefited more from confidence visualization if the red underlining had been used more or less often. On the one hand, underlining too often might risk users ignoring the visualization altogether. On the other hand, only underlining very likely errors might lead users to believe they do not need to proofread the rest of their text. It would be interesting to do a further study testing human performance at different false accept/reject operating points. It might also be interesting to study the effectiveness of coloring schemes different from the one I used.

2.6.3 Confidence Score Accuracy

For a measure of word confidence, I used the posterior probabilities from a word confusion network. A closely related lattice-based technique uses the posterior probabilities and timing information in the recognition lattice to produce a confidence score [160]. Both the WCN- and lattice-based approaches to confidence scoring provide state-of-the-art accuracy compared to other techniques [48; 65; 160]. But it is possible that in the future more advanced methods will produce more accurate confidence scores. For example, it has been suggested that confidence scores could be improved by using information sources besides the recognition lattice, such as syntactic or semantic information [77]. Improved

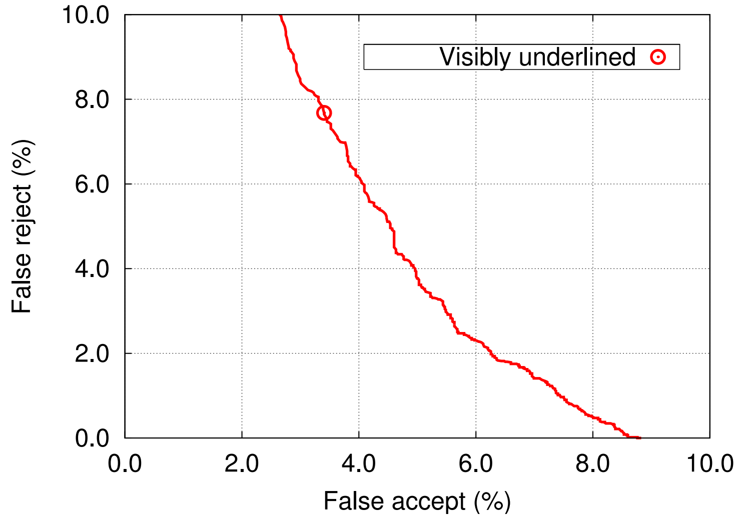


Figure 2.15: This curve shows the trade-off between false-accepts and false-rejects in the visualization condition. A certain accept/reject threshold defines a point on this curve. The confidence visualization in my study became (subjectively) visible at a false-accept rate of 3.4% and a false-reject rate of 7.7% (denoted by the circle).

confidence scores would reduce the number of false-accept and false-reject errors, potentially making confidence visualization more effective.

2.7 Related Work

In this section, I discuss past work related to confidence visualization.

To my knowledge, the earliest work on confidence visualization was by Schmandt [126]. In this work, waveforms of dictated speech were displayed alongside text from a word-spotting recognizer. The interface was designed to aid in the editing of waveforms. The text was displayed in a brighter color according to the recognizer's confidence score. No user study was done to test whether the varying brightness level was useful to users. Schmandt's goal for using confidence visualization was also different from mine. He wanted to draw attention to *correctly* recognized words as they could serve as good reference points into the audio. In contrast, I wanted to draw attention to *incorrectly* recognized words to facilitate error detection.

Another application of confidence visualization is to help users understand the gist of an audio file without actually listening to it. In Burke et al. [24], they performed speech recognition on voice mail messages and shaded the recognition results using a WCN-based confidence measure. Their objective was to allow quicker reading of voice mail summaries by deemphasizing low-confidence words. This objective was different from my goal of focusing user attention on potential errors. Additionally, they never treated confidence visualization as an independent variable. In a questionnaire, they did find that participants thought confidence visualization was "helpful for identifying mistakes" (scoring 4.1 on a 5-point scale where 5=completely agree). I also found participants subjectively reported finding confidence visualization somewhat helpful for finding errors (scoring 4.7 on a 7-point scale). However, as previously mentioned, these subjective results may be suspect as participants tend to rate a system highly in the presence of the system's designer [122].

Vermuri et al. [145] also tested the use of confidence visualization to improve comprehension of audio recordings. In this study, participants saw both the recognition result and heard the audio file. The recognition results were displayed both with and without confidence visualization. No difference in participants' comprehension rate was found. Again, their goal of aiding comprehension was different from my goal of facilitating error detection in the context of a text dictation task.

A number of studies have looked at using confidence visualization to aid users in correcting errors. The first such study was by Suhm et al. [139]. In this study, participants corrected speech recognition results using a speech- and pen-based correction interface. Suhm et al. found no statistically significant difference between the corrected words-per-minute achieved using an interface with confidence visualization and one without. Similarly, my results also showed no difference in the time it took to detect errors with and without visualization. However, there are several differences between their study and mine. First, the times Suhm measured included speech- or pen-based correction. In contrast, my study used an oracle to instantly correct errors that users detected. This allowed me to measure error detection time – the time it takes to proofread recognized text and indicate errors. Second, Suhm et al. had a 25% WER on their participants' original speech. This was much higher than my WER of 8.5%. I suspect that confidence visualization becomes increasingly useless as WER increases due to the large number of things highlighted. With such a high WER, it is likely that users would ignore confidence visualization altogether and carefully check the entire recognition result for errors.

A second study of a correction-style interface was by Endo et al. [47]. In this study, they developed a mathematical model for how highlighting of low-confidence words might speed the error correction process. They conducted a user trial using simulated errors producing a WER of between 10 and 19%. Participants were required to correct all errors in each sentence. Endo et al. showed that using confidence visualization resulted in a 13% speedup in correction. This difference was below the 46% performance predicted by their mathematical model. The statistical significance of the difference was not reported.

Another study of a correction-style interface was by Collins et al. [35]. In this work, they visualized machine translation and speech recognition results by displaying the recognition lattice. They visualized confidence in the alternate lattice paths by using color, border size and transparency. They used the lattice visualization both to represent uncertainty and to facilitate the correction process. No user study was reported regarding the effectiveness of their representation.

A possible application of confidence scores is to help users navigate more efficiently to errors. Feng et al. [49] used confidence scores to place navigation anchors at likely error points in a user's recognition result. In simulations, confidence-based anchors required slightly fewer navigation commands than naive fixed-interval anchors. They performed a user trial using confidence-based anchors, but did not compare it to a fixed-interval baseline.

2.8 Conclusions

I have presented a system capable of visualizing the recognizer's confidence in all three types of recognition errors: deletion, insertion and substitution. I used this system to investigate if confidence visualization helped users find errors in a dictation task.

An evaluation of my system showed that confidence-based underlining did not, overall, improve users' speed or accuracy at finding errors. However, unlike previous work, I found that it was not confidence visualization per se that caused the non-result. Rather, I found that when confidence visualization "did the right thing" and highlighted incorrect recognition results, participants detected significantly more errors when using visualization than without it. However, participants also trusted the confidence visualization and tended to miss errors that (incorrectly) had a high confidence. This suggests that confidence visualization must be used cautiously. A poor confidence measure may distract attention away from legitimate errors, leading to an actual increase in user errors. On the other hand, a good confidence measure may focus attention on errors that would otherwise go unnoticed, leading to a decrease in user errors.

In my study, it seemed that confidence visualization was at a break-even point. It neither helped nor hurt users' ability to detect errors. Even using state-of-the-art confidence scores and at a low word error rate, my system still mistakenly underlined 8% of correct words and failed to underline 3% of erroneous words. It is possible in the future, better confidence scores or more accurate recognition may tip the balance in the favor of confidence visualization. But such a change would need to be confirmed by empirical experiments.

Chapter 3

Spoken Corrections

3.1 Overview

When using a speech recognizer to dictate text, correcting errors dominates task time [80; 86]. While users show a strong preference for correcting errors by voice, this strategy usually proves error-prone, inefficient, and frustrating [66].

So why are spoken corrections problematic? A possible reason is that while correcting dictation errors, people adopt a more hyperarticulated speaking style. During error resolution, users have been shown to slow their speaking rate, add pauses, and pronounce words more carefully [97; 115]. This change in speaking style may cause a mismatch between users' hyperarticulate speech and the naturally-read speech typical of a recognizer's training data. Some studies have shown this mismatch has a negative impact on recognition accuracy. For example, Bell et al. [13] and Shriberg et al. [131] showed an increase in recognition errors for utterances users repeated after encountering errors in a spoken dialog system. Soltau et al. [133] showed an increase in errors for hyperarticulated isolated words. However, not all studies show hyperarticulation has a negative impact on accuracy. A study by Stent et al. [136] showed hyperarticulate speech actually improved accuracy.

If hyperarticulation is problematic, a simple solution might be to instruct users not to hyperarticulate. But this could prove difficult for users as it requires users override a common strategy used in human-to-human communication [117]. Past work has found instructing users to always "speak naturally" reduces but does not eliminate users' tendency to hyperarticulate [131; 155].

Another possible problem with spoken corrections is they tend to be short, often consisting of a single word or only a few words. These short correction utterances could be problematic because they do not fit the full sentences typical of the training data for the recognizer's acoustic and language model.

So how *do* novice users speak to a dictation interface? Do they hyperarticulate? If so, what effect does this have on recognition accuracy? How is recognition performance affected by the short utterances typical of spoken corrections? In this chapter, I investigate these questions. To do this, I first collected speech data using a simulated dictation interface. My dictation interface had users speak sentences and then correct simulated errors by respeaking. Using three state-of-the-art commercial and research speech recognizers, I conducted experiments to see how users' speaking style impacted recognition accuracy. I also explored how to adapt a recognizer's acoustic model to provide better accuracy on the short utterances typical of spoken corrections.

This chapter is structured as follows. First, I describe a corpus I collected of users dictating sentences and performing spoken corrections. Second, I analyze the audio in my corpus to show how the correction process affected users' speech. Third, I perform recognition experiments to investigate what impact hyperarticulate speech has on recognition accuracy. Fourth, I investigate improving recognition on corrections that consist of a single word or part of a sentence.

3.1.1 Publication Note

The work presented in this chapter was published in part in the paper "Speech and Speech Recognition during Dictation Corrections" at the International Conference on Spoken Language Processing (ICSLP 2006) [148].

3.2 Data Collection

In this section, I describe the procedure I used to collect speech of users dictating sentences and performing spoken corrections. This audio data will serve as the basis for the experiments in the rest of the chapter.

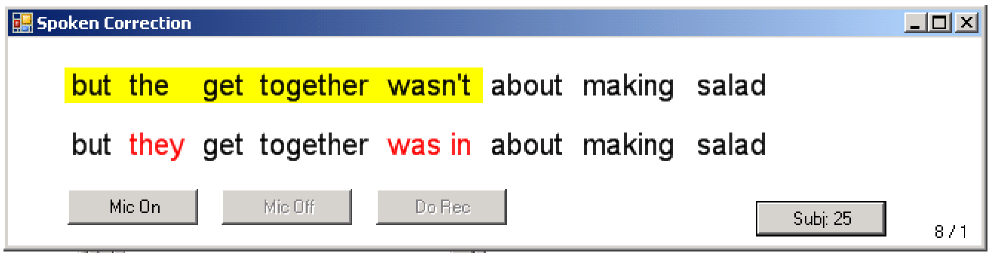
3.2.1 Participants

I recruited 24 volunteer participants to use a dictation-style interface. Participants were native speakers of North American English and had no prior experience dictating to a computer (one participant had experience dictating to a human). Participants were gender balanced and aged from 24 to 59 (average 31). A majority of the participants had Midwestern or Canadian ascents.

3.2.2 Conditions and Method

Participants were told they would be using a speech recognizer and correcting any errors by respeaking parts of the text. This was not actually true, as the "dictation" interface was merely recording their voices. Each trial took place in a quiet office-like environment and used the same laptop and Plantronics DSP-400 USB headset microphone.

In part one of the experiment, participants completed the standard "Talking to your computer" enrollment session in Nuance Dragon NaturallySpeaking v8.1. The enrollment text reminded participants multiple times to speak naturally – like "newscasters read the news". The enrollment session helped to teach participants how to dictate to the computer. It also served to help fool participants into thinking the computer was doing actual recognition in the next phase of the experiment. During enrollment, Dragon provided visual feedback of the current location in the enrollment text via colored highlighting. If Dragon detected problems with a section of speech, participants were required to speak that section again.



but the get together wasn't about making salad

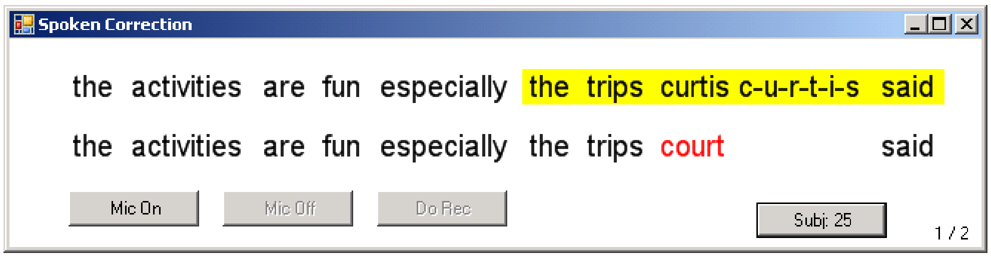
but they get together was in about making salad

Figure 3.1: The participant is being asked to respeak "but the get together wasn't". The yellow background highlighting indicates which words the participant should respeak. The words in red show the simulated recognition errors.

In part two of the experiment, participants read 42 newswire sentences drawn from the San Jose Mercury sentences in the WSJ1 si_dt_s2 (Spoke 2) test set [5]. A quarter of the sentences were hard by design, including words that were not in the WSJ 20K and 64K vocabularies. Overall, the sentences had an out-of-vocabulary (OOV) rate of 6.9% using the WSJ 20K vocabulary and 3.1% using the 64K vocabulary. The order of the sentences, aside from two initial practice sentences, was randomized for each participant. Participants were told no recognition would occur in part two of the experiment and that they should read the sentences "naturally".

In part three of the experiment, participants were instructed to read the 42 sentences again but that this time the computer would try and recognize their speech. After a "successful" recognition, a happy tone would play and the next sentence would appear. After a "failed" recognition, a sad tone would play. After a "failure", a "recognition result" would appear below the original sentence with word errors displayed in red (figure 3.1). A word, part of a sentence, or the entire original sentence would then be highlighted in yellow and participants would respeak the highlighted text. Participants repeated speaking the highlighted text until recognition was "successful". On some sentences, a final "speak-and-spell" correction asked participants to spell a particular word after speaking that word in the correction utterance (figure 3.2). If participants made a genuine speaking mistake such as omitting a word, they were instructed to re-record that utterance.

In actuality, no recognition took place in part three. Participants' audio



the activities are fun especially the trips curtis c-u-r-t-i-s said

the activities are fun especially the trips court said

Figure 3.2: The participant is being asked to repeat a portion of the original sentence and to both speak and spell the word "curtis".

was recorded but had no influence on the "success" or "failure" of recognition. Each sentence had a predetermined set of simulated errors which every person experienced. These simulated errors were based in part on actual recognition results I obtained reading the sentences to a desktop speech recognizer.

3.2.3 Corpus Statistics

The number of corrections per sentence was varied from zero to five with an average of 3.5 corrections per sentence. For any sentence with one or more errors, participants were required to repeat a single word (8 sentences), a portion of the sentence (16 sentences), or the entire sentence (8 sentences). A speak-and-spell correction was recorded as the final correction for 10 of the partial-sentence corrections and for 1 of the single-word corrections. The speak-and-spell utterances were not used in the experiments presented in this chapter. For full details of the sentences and corrections collected, see appendix B.1.

A total of 7.5 hours of audio was collected. Of this, 1.6 hours was adaptation data from the Dragon enrollment session and 5.9 hours was from the reading of sentences and corrections.

3.2.4 Utterance Types and Judging

In the rest of this chapter, utterances will be identified as follows:

- `adapt` - Adaptation utterance, collected during the initial Dragon "Talking to your computer" enrollment session (part one of the experiment).
- `pre` - Complete sentence, collected before any simulated errors on any sentence (part two of the experiment).
- `init` - Complete sentence, collected before any errors on that particular sentence (part three of the experiment).
- `err1-4` - Error correction, collected after a simulated error (part three of the experiment). The participant would respeak a word, part of a sentence, or an entire sentence (`err1` is the first correction, `err2` is the second correction, etc).

As in [131], I judged the degree of hyperarticulation on each utterance using a three-point scale (0 = normal, 1 = somewhat hyperarticulate, 2 = strongly hyperarticulate). I judged the utterances myself in random order and without knowledge of the utterance type I was scoring.

3.2.5 Collection Errata

In the original design of the data collection experiment, users proceeded from the Dragon enrollment session (part one of the experiment as described in section 3.2.2) straight into using the interface with simulated recognition errors (part three of the experiment). After observing the speech of the first two participants, I realized participants might hyperarticulate in response to errors on past sentences. As I wanted a naturally-read version of each sentence, I added an intermediate step in which I told participants that no recognition would take place and that they should read each sentence "naturally" (part two of the experiment). Participants one and two were subsequently asked to re-read the sentences after being told no recognition would occur and that they should speak naturally. The procedure for all other participants proceeded as described in section 3.2.2.

During part two of participant 13's experiment, the microphone was mistakenly muted. This participant was asked to redo part two before proceeding to part three. In informal interviews conducted after the experiment, participants 6 and 18 indicated they were suspicious about whether the computer was really doing speech recognition. All other participants indicated that they believed the computer was actually recognizing their speech.

3.3 Analyzing Speech During Corrections

In this section, I analyze the acoustic properties of the speech in my corpus.

3.3.1 Analysis Procedure

I first obtained word- and phone-segmentations of each utterance by aligning the correct text transcription using the HTK recognizer [169]. I then calculated the pitch, intensity, and formant frequencies of each utterance using the software program praat [22].

For purposes of comparison, I analyzed just the audio sections in the `pre` and `init` utterances corresponding to the words in the `err1-4` utterances. I found the corresponding sections in the `pre` and `init` utterances by using the segmentations provided by the forced alignment. Speak-and-spell corrections were not included in the `err1-4` set. I excluded sentences with single word corrections from the analysis as the audio segments were often very short and this caused difficulties for praat's pitch analysis algorithm.

I compared the utterance types using the cumulative distribution for each of the speech factor I analyzed. I found two-sigma error bars by using the method described in appendix D.1.

3.3.2 Duration and Pausing

I obtained the total number of syllables in each utterance using a dictionary with syllable markings [161]. I calculated the speaking rate in syllables per second for

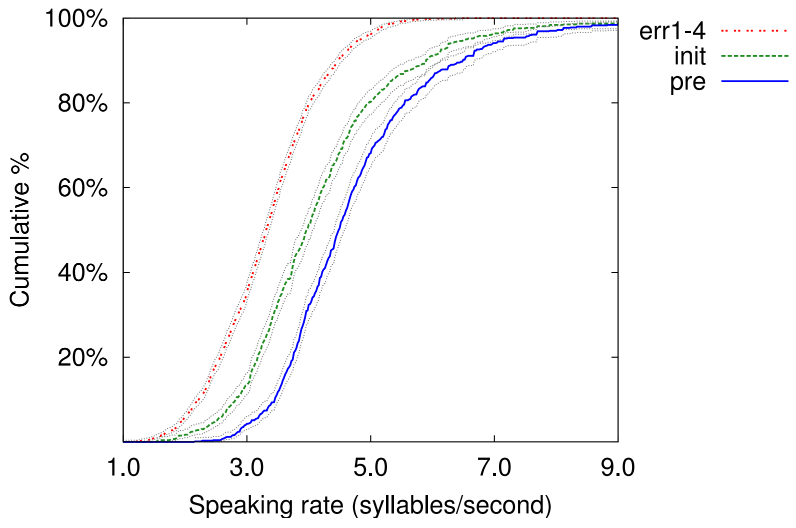


Figure 3.3: Cumulative distribution of speaking rate on `pre`, `init` and `err1-4` utterances (for all participants). Dotted gray lines denote two-sigma error bars.

each utterance, removing starting and ending silence using the forced alignment. As shown in figure 3.3, even before errors occurred on a particular sentence, the `init` utterances were slowed in comparison to the `pre` utterances. A further speaking rate reduction was seen on the `err1-4` error corrections. I found speaking rate did not differ significantly as users repeated the same correction in the `err1`, `err2`, `err3` and `err4` utterances.

I also calculated the amount of inter-word silence in each utterance by summing the durations of all silence phones which appeared between words in the forced alignment. This showed increasing amounts of inter-word silence in the `init` and `err1-4` utterances (figure 3.4).

3.3.3 Pitch, Intensity, and Formant Frequencies

I also calculated cumulative distributions for the following:

- **Pitch** - Minimum, maximum, mean and slope of pitch.
- **Intensity** - Minimum, maximum, and mean of intensity.
- **Formants** - Mean of the first five formants (F1–F5).

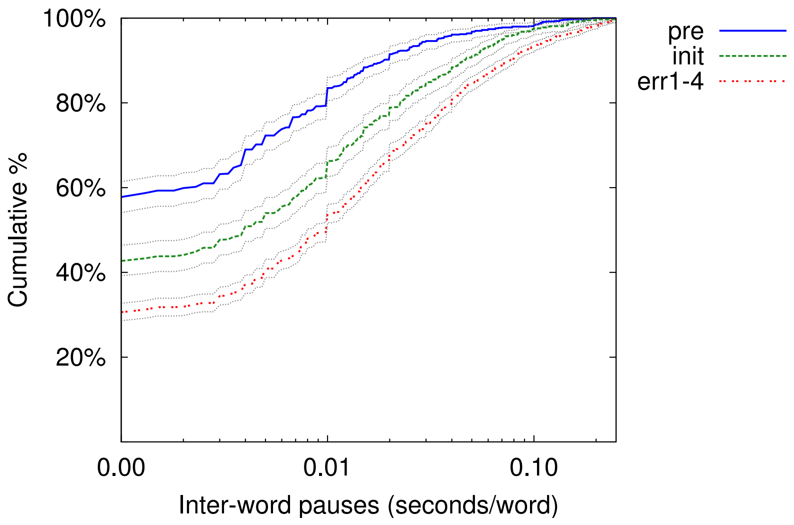


Figure 3.4: Cumulative distribution of inter-word pausing on `pre`, `init` and `err1-4` utterances. Dotted gray lines denote two-sigma error bars. Note the x-axis is on a log scale.

For brevity, the distributions in which I found significant differences are summarized here by their quartiles (figure 3.5). The main effects I saw for error correction attempts (`err1-4`) were a widening in the range of both pitch and intensity (lower minima and higher maxima). I also saw a decrease in the average frequencies of the first three formants (F1–F3). The differences reported were significant in the sense that the two-sigma error bars of the cumulative distributions were non-overlapping. See appendix B.2 for graphs of all the distributions I analyzed.

3.3.4 Correction Strategies

Aside from the initial Dragon instructional text, no intervention or advice was given to the participants on how to speak during the experiment. I observed that participants employed a wide variety of strategies in their efforts to obtain correct recognition. Some participants consistently hyperarticulated corrections while others spoke normally throughout. Some explored different strategies, changing between normal, hyperarticulated and *hypo*articulated (reduced) speaking styles.

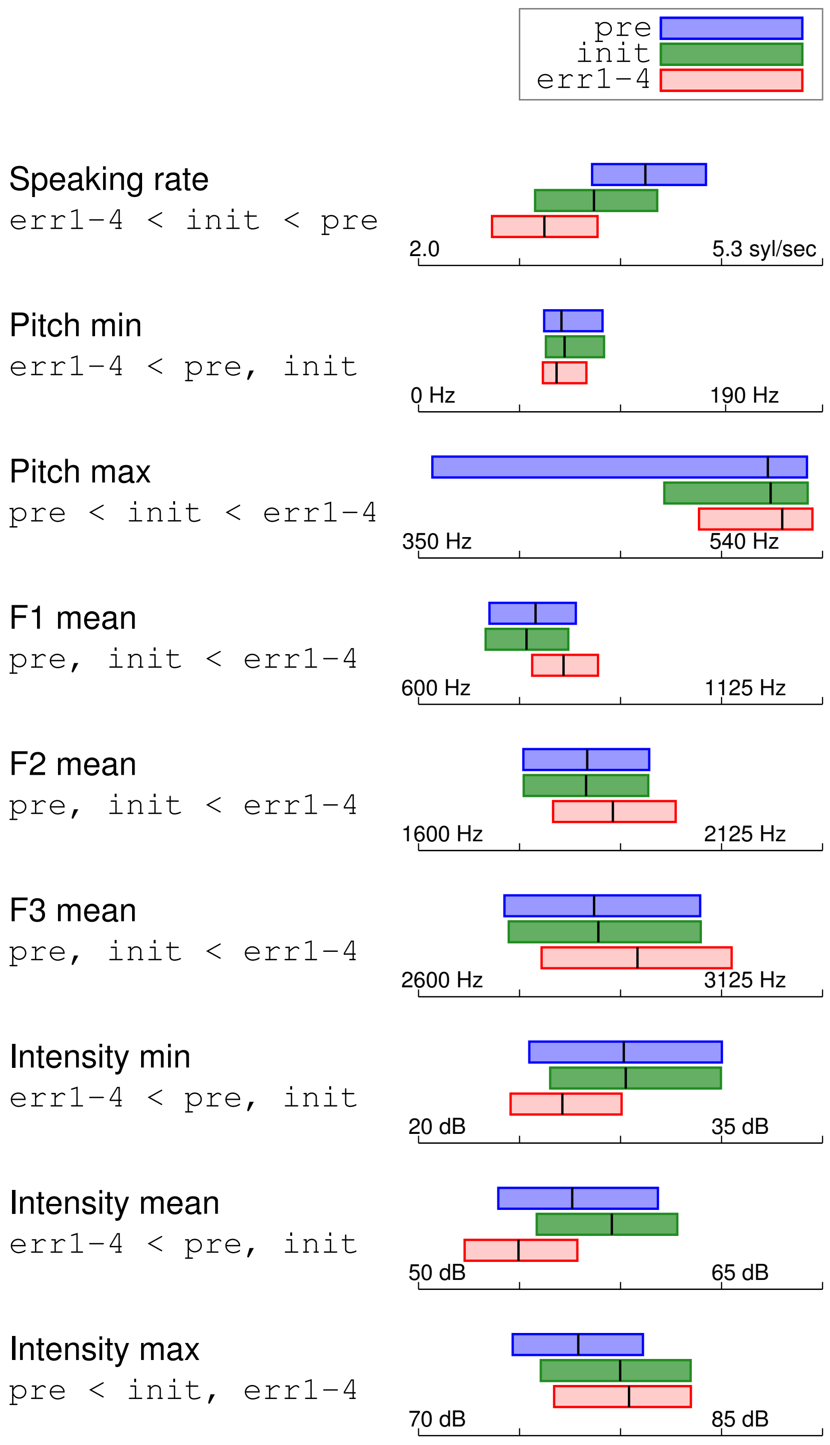


Figure 3.5: On the left, the significant acoustic differences between the `pre`, `init`, and `err1-4` utterances are listed. On the right, the lower and upper quartiles of each utterance type is shown with a central line denoting the median.

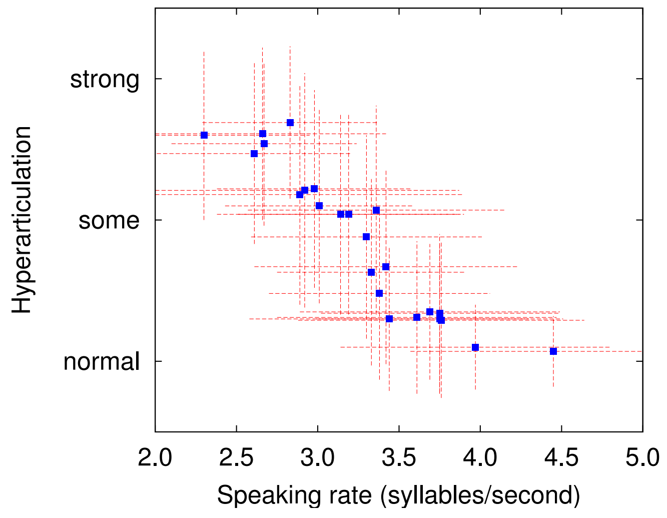


Figure 3.6: Speaking rate and judged hyperarticulation on spoken corrections (`err1-4`). Each dot represents a different participant. The dotted lines show the one-sigma variability within that participant's utterances.

A number also tried isolated speech, inserting long pauses between every word.

The spectrum of participants' correction strategies is depicted in figure 3.6. As expected, the quantitative measure of speaking rate and the qualitative measure of judged hyperarticulation were correlated.

3.4 Recognition Experiments

In this section, I conduct recognition experiments on the audio from my corpus. The goal was to ascertain what effect the speech changes observed during corrections had on recognition accuracy.

3.4.1 Recognizer Setup

For these experiments, I used two commercial speech recognizers and one research recognizer:

- **Microsoft** - I used the recognizer provided in the Microsoft Speech SDK v5.1 [106]. I accessed the engine via the Speech Application Programming Interface (SAPI). I set the Microsoft engine to its maximum accuracy setting, disabled background adaptation, and used an untrained user profile.
- **Dragon** - I used Nuance Dragon NaturallySpeaking v8.1 [111]. I accessed the engine via the C++ SDK. Except where noted, I set Dragon to its default accuracy setting and used an untrained user profile.
- **HTK** - I used the research recognizer HTK v3.3 [169; 170]. The details of the HTK setup are described below.

To ensure Microsoft and Dragon were not doing unsupervised adaptation during the experiment, I performed recognition one utterance at a time. I started each engine with an untrained user profile, performed recognition on a single utterance, then shutdown the engine (without updating the user profile). For the Microsoft recognizer, it was also necessary to replace the user profile files with copies made before the start of recognition. This procedure resulted in Microsoft and Dragon returning identical recognition results regardless of the order of utterances. I report error bars using a one standard error (σ_{se}) confidence interval obtained via per-utterance bootstrap resampling [17].

I trained the HTK acoustic model following my HTK training recipe [147]. I parameterized audio into a 39-dimensional feature vector consisting of 12 Mel-frequency cepstral coefficients plus the 0$^{\text{th}}$ cepstral, deltas and delta deltas, normalized using cepstral mean subtraction. I used the CMU phone set without stress markings (39 phones plus silence). Each phone HMM had three output states and a left-to-right topology with self-loops. For a pronunciation dictionary, I used the CMU dictionary [25].

I trained initial monophone models using the TIMIT corpus [58]. I then trained cross-word triphones using the SI-284 training subset (66 hours) of the WSJ0 [57] and WSJ1 [5] corpora. I used 16 continuous Gaussians per output state and 32 Gaussians for silence states. All Gaussians used diagonal covariance matrices. Output states were tied using a phonetic decision tree. This yielded

Acoustic model training	WSJ0 si_et_05 (Nov'92)	WSJ1 si_et_h2 (Nov'93)
My recipe	5.25	7.36
Woodland et al.	5.14	6.91

Table 3.1: Word error rates on two test sets using my acoustic model training recipe [147] and the results reported in Woodland et al. [165].

a speaker-independent, gender-independent model with approximately 7400 tied-states and 9.3 million parameters.

Table 3.1 shows the error rates of my trained acoustic model compared to the error rates reported in Woodland et al. [165]. Both our experiments used the SI-284 acoustic training set, the standard WSJ 5K bigram language model, cross-word triphones, and a similar numbers of tied-states. We both used WSJ0 si_et_05 as a development test set and WSJ1 si_et_h2 as an evaluation test set. My models performed slightly worse than Woodland et al. A possible reason for the difference is that Woodland et al. trained gender-dependent acoustic models while I trained only a gender-independent acoustic model.

I trained bigram and trigram language models on newswire text from the CSR-III text corpus [64] (222M words). The language model's vocabulary was the top 64K words appearing in the corpus and used non-verbalized punctuation. The language model was trained with interpolated modified Kneser-Ney smoothing and no count cutoffs. I entropy-pruned [137] the language model using SRILM [138] and a threshold of 1×10^{-8}. This resulted in a language model with 8.7M bigrams and 12.4M trigrams. I used the HDecode decoder from HTK v3.4 [167; 168]. Recognition used the bigram language model, expanding and rescoring the bigram word lattices with the trigram language model.

3.4.2 Baseline Recognition

Before analyzing recognition performance on the spoken corrections in my corpus, I first wanted to test how well the three different recognizers performed on

Recognizer	$\times$RT	WER $\pm$ σ_{se}
Microsoft	0.5	32.8 $\pm$ 1.6
Dragon	0.4	31.6 $\pm$ 1.4
HTK	5.2	15.6 $\pm$ 0.8

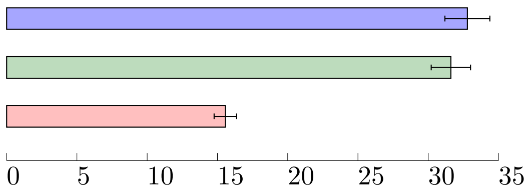


Table 3.2: Real-time factor ($\times$RT) and word error rate (WER) of the three recognizers on the San Jose Mercury sentences in the WSJ1 si_dt_s2 test set.

a standard acoustic test set consisting of naturally-read full sentences. To my knowledge, no one has compared recognition performance of the Dragon, Microsoft and HTK recognizers on a standard test set. I ran this experiment using the San Jose Mercury newswire sentences from the WSJ1 si_dt_s2 test set (207 sentences). This is the same source from which I drew the text for my participants' sentences. Using the same sentences also allowed me to check how well my recordings compared to the recording collected in the original WSJ test set. I found Dragon did not work well with the low volume audio in the original test set utterances. I normalized the volume in the utterances to 30% root mean square (RMS) using Vox Studio v3.0 [3]. I used the volume normalized audio files for the experiments with all three recognizers.

Table 3.2 shows the real-time factor (how long recognition took compared to the length of the audio) and the word error rate (WER). Results were on a 2.2 GHz computer. While HTK took longer for recognition, it provided much better accuracy. This may reflect the close match between the WSJ test set and the WSJ data that trained the HTK acoustic and language models.

Table 3.3 shows the WER on the subset of the WSJ1 si_dt_s2 test set that I recorded from my participants. Microsoft and HTK had similar WER on the original corpus versus the `pre` utterances recorded by my participants (Microsoft 33% versus 30%, HTK 16% versus 17%). Dragon performed worse on the original corpus compared to the `pre` utterances (32% versus 23%).

Recognizer	Utt type	WER $\pm\sigma_{se}$
Microsoft	`pre`	29.5 ± 0.8
Microsoft	`init`	29.0 ± 0.7
Dragon	`pre`	23.1 ± 0.7
Dragon	`init`	20.7 ± 0.6
HTK	`pre`	17.3 ± 0.5
HTK	`init`	16.2 ± 0.5

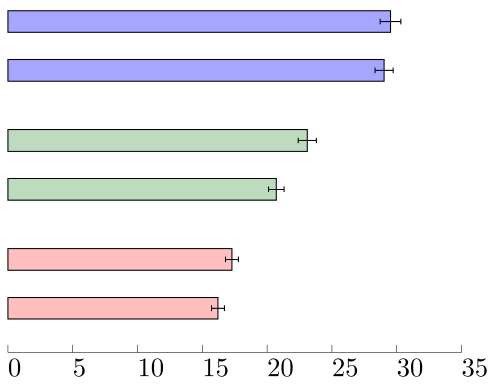


Table 3.3: WER on normally read `pre` utterances and somewhat hyperarticulated `init` utterances.

	`pre`	`init`	`err1`	`err2`	`err3`
Judged score[†]	0.17	0.73	0.99	0.89	0.96
Syllables/sec	4.30	3.78	3.57	3.62	3.60
Pause sec/word	0.006	0.016	0.023	0.025	0.025

Table 3.4: Three measures of hyperarticulation on utterances where corrections where complete sentences. [†] Hyperarticulation was judged on a three point scale: 0 = normal, 1 = somewhat hyperarticulate, 2 = strongly hyperarticulate.

3.4.3 Whole Sentence Recognition

I compared the initial reading of a sentence (`pre`) to the second reading (`init`) (960 utterances per type). I found participants' utterances increased in length by 18% on average in the `init` reading. Participants also tended to hyperarticulate more on `init` utterances with the judged hyperarticulation score increasing from 0.09 to 0.58. Despite the increased hyperarticulation, all three recognizers showed reduced WER on the `init` utterances compared to the `pre` utterances (table 3.3).

For eight sentences, after the two initial readings (`pre`, `init`), three full-sentence error corrections (`err1-3`) were recorded (8 sentences $\times$ 24 speakers = 192 utterances per type). As shown in table 3.4, both quantitative and qualitative

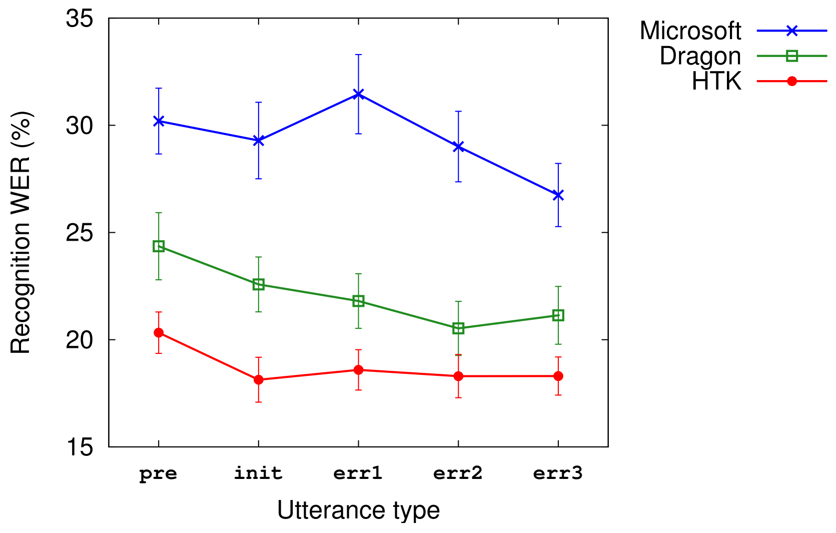


Figure 3.7: WER on sentences where corrections involved reading the whole sentence repeatedly.

measures of hyperarticulation increased on `init` and `err1-3` utterances. However, this did not adversely affect recognition. Error rates for all three recognizers remained similar or decreased on repeated whole sentence utterances (figure 3.7).

3.4.4 Partial Sentence Corrections

For 12 sentences, after the two initial readings of the full sentence (`pre`, `init`), three word- or partial-sentence corrections (`err1-3`) were recorded (288 utterances per type). The word or partial-sentence corrections showed increased hyperarticulation with the judged score increasing from 0.08 on `pre` to 0.80 on `err1-3`. The `err1-3` utterances also had a 29% slower speaking rate.

The error rate of each recognizer on the corresponding fragments of the `pre` and `init` utterances was found by aligning the full sentence recognition results with the reference transcripts. Errors increased for the word and partial-sentence corrections in isolation as compared to when carried within a full sentence (figure 3.8). Note that no surrounding context (i.e. words appearing before the correction location) was used during `err1-3` recognition. This made the recognizer's job harder than strictly necessary. Despite this, Dragon coped reasonably

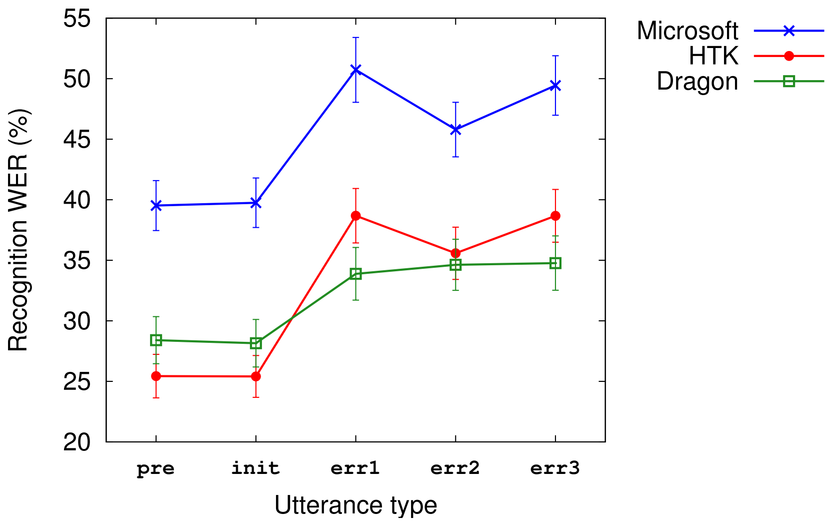


Figure 3.8: WER on corrections that were single-words or partial-sentences. The error rates reported for `pre` and `init` were on the corresponding partial part of the result obtained recognizing the full `pre` and `init` sentences.

well while HTK degraded markedly, showing a large 48% relative increase in WER. Dragon's accuracy on short corrections suggest (as might be expected) it has been specifically designed to handle spoken corrections. Perhaps Dragon has short correction-like examples in its acoustic or language model training data.

3.4.5 Matched Utterance Pairs

I compared the WER of pairs of utterances where each pair came from the same speaker and contained identical lexical content. I paired utterances which were judged normally-spoken with utterances judged somewhat- or strongly-hyperarticulated. Microsoft and HTK did slightly worse recognizing the speech judged hyperarticulate while Dragon did slightly better (figure 3.9a, 698 utterance pairs). I found similar results when I compared normally-spoken and strongly-hyperarticulated utterances (figure 3.9b, 217 utterance pairs) and when I compared the shortest and longest utterances (figure 3.9c, 1666 utterance pairs).

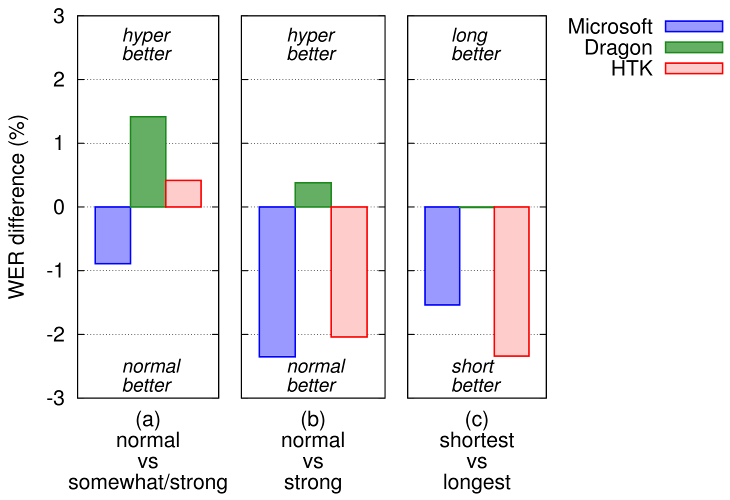


Figure 3.9: Difference in WER between participants' normal and hyperarticulate utterances. I paired utterances based on judged hyperarticulation (a and b) and overall utterance length (c).

3.4.6 Effect of Dragon Adaptation

To this point, I have used speaker-independent acoustic models for all recognizers. In real-world usage, users would usually adapt the recognizer to their voice. I wanted to see if adaptation had any influence on the recognition accuracy of spoken corrections.

During my experiment, I had participants complete Dragon's "Talking to your computer" enrollment session. This provided a speaker-dependent model for each participant. I ran experiments to investigate how effective Dragon adaptation was. In addition to adaptation, I also tested Dragon's accuracy setting. This setting controls the trade-off between recognition accuracy and the time it takes to return recognition results.

3.4.6.1 Per Participant Adaptation Results

First, I validated that adaptation improved recognition accuracy on the normally spoken (`pre`) utterances. As shown in figure 3.10, accuracy did indeed improve

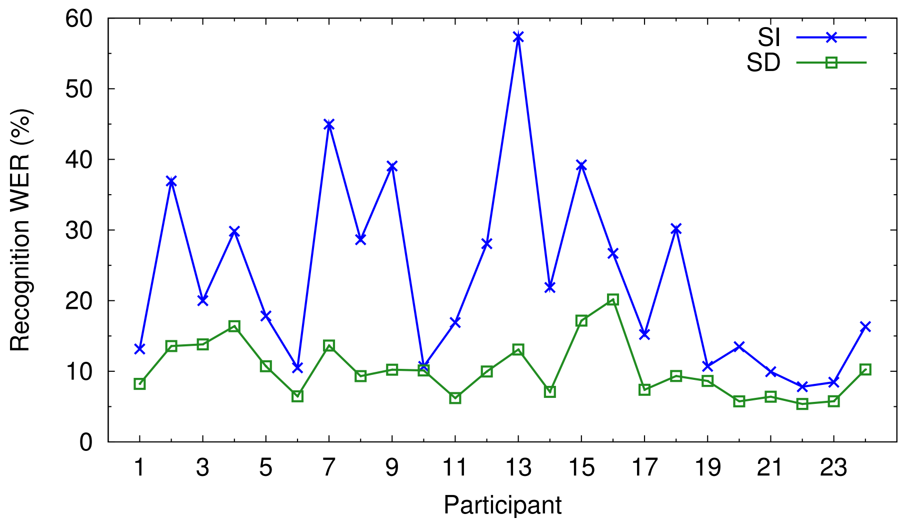


Figure 3.10: WER of each participant on normally spoken `pre` sentences. I tested each participant using a speaker-independent (SI) and a speaker-dependent (SD) model (both using Dragon's default accuracy setting).

for all participants. Some participants experienced very large gains, for example participant 13 went from 57% WER to 13%.

3.4.6.2 Adaptation and Hyperarticulate Speech

Next, I tested what effect adaptation had on my set of repeated full sentence corrections. As expected, the speaker-dependent models substantially reduced WER for every utterance type (figure 3.11). Averaged over all utterance types, WER was reduced from 22% to 11% at the default accuracy setting. Using the maximum accuracy setting further reduced WER to 10%.

Across all Dragon configurations tested, the full sentence corrections (`err1-3`) always had a lower WER compared with the initial (`pre`) reading. Looking at the trends in figure 3.11, the WER of the adapted models leveled out or increased slightly as corrections were repeated (`err1-3`). The unadapted model tended to have lower WER as corrections were repeated. In all configurations, Dragon was robust to the hyperarticulate speech typical of spoken corrections.

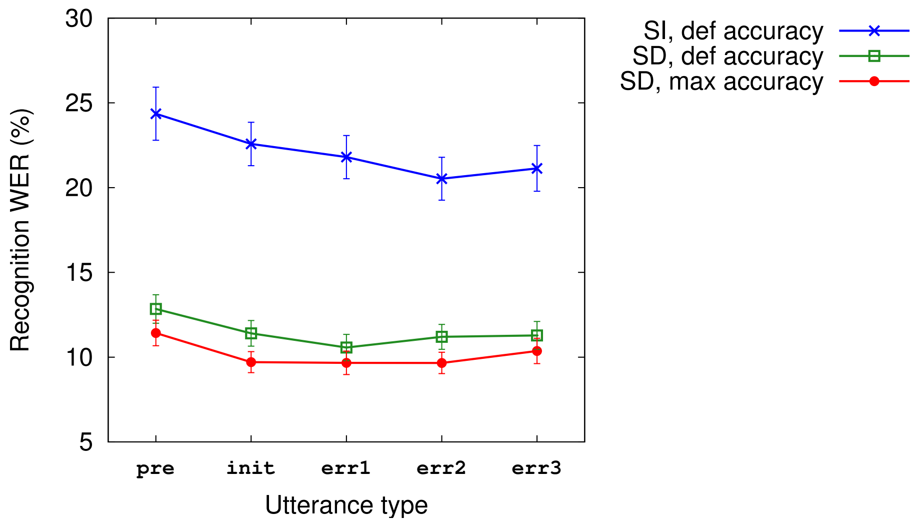


Figure 3.11: WER on sentences in which participants read an entire sentence repeatedly. Recognition used Dragon with either a speaker-independent (SI) or a speaker-dependent (SD) model. I also varied Dragon's accuracy setting between the default and the maximum setting.

3.5 Improving Recognition of Short Corrections

For all three recognizers, the WER on short utterances consisting of single-word or partial-sentence corrections (figure 3.8) was higher than the WER on full sentences (figure 3.7). One explanation for this is that I chose corrections to be in locations that would be difficult for a recognizer. For example, I chose locations that included proper names or uncommon words (e.g. "but Ingram added" and "the eccentric old hermit"). I also chose locations in which I observed recognition errors on my own audio. Additionally, the short corrections were recognized without providing surrounding context information to the language model. In this section, I look at improving the accuracy on these short correction utterances using the HTK recognizer.

3.5.1 Improvement Method

There are two main methods available for improving recognition of short corrections. The first is to improve the language model. An obvious way to improve the language model is to use the context surrounding the correction to bias recognition to things consistent with the surrounding context. This has been studied before and was shown to be effective [140].

The second method is to improve the acoustic model. This could be done by tuning an acoustic model for the hyperarticulate speech typical of corrections. The model might also be tuned for the speech typical of single-word or partial-sentence corrections. It may be necessary to tune the model to short, partial-sentence corrections as other researchers have found a model trained on continuous speech does not perform well on isolated words [10].

Given the unique nature of the corpus I collected, as well as the acoustic differences seen in my participants' corrections, I focused on improving the acoustic model.

3.5.2 Experimental Setup

I used speaker adaptation techniques to see if the WSJ acoustic model could be adapted to better recognize single word or partial-sentence corrections. I did this in a supervised, speaker-independent fashion. I created a correction-specific acoustic model using audio with known transcriptions from a set of training speakers. This adapted model was used to recognize corrections from the test set speakers.

As I needed precise control of the recognition process, the experiments in this section used only the HTK recognizer. For details of the HTK setup, see section 3.4.1. I split my corpus into a training set of 16 speakers and a test set of 8 speakers. Both sets were gender-balanced.

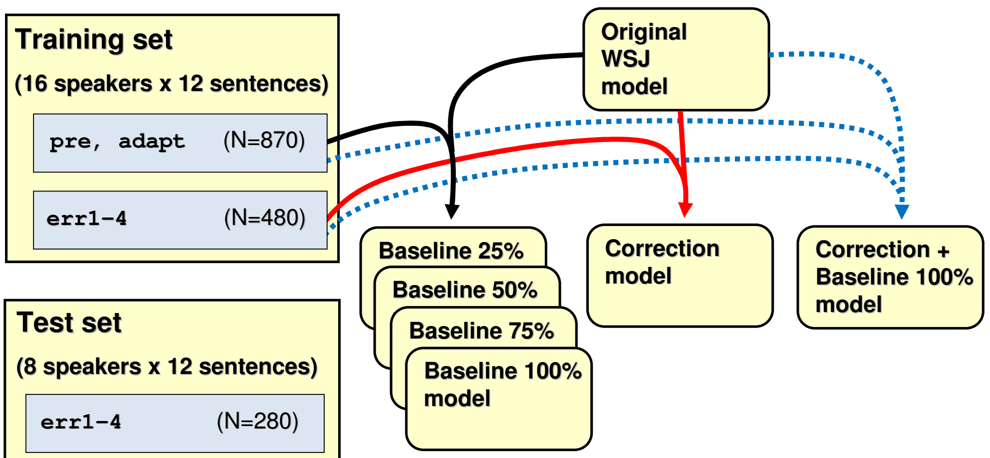


Figure 3.12: The corpus was split into training and test sets for the adaptation experiment. The normally spoken `pre` and `adapt` utterances were used to create baseline models. The `err1-4` utterances were used to create correction-specific models.

3.5.3 Adaptation Data

I used 24 of the 40 sentence tasks that had single word or partial-sentence corrections. This excluded sentences with whole-sentence corrections and sentences with no corrections.

As shown in figure 3.12, the training set consisted of two parts. The first *normal speech* part had the Dragon adaptation utterances (`adapt`) plus the `pre` utterances (870 utterances). The *correction speech* part had the `err1-4` utterances (480 utterances). I did not include the `init` utterances as they were neither true corrections attempts nor naturally read initial dictations. The `pre` and `err1-4` utterances were taken from 12 of the 24 available sentence tasks.

The test set consisted entirely of `err1-4` utterances (280 utterances). These utterances came from 12 of the sentence tasks, but did not overlap with the sentences chosen for the training set. This ensured that there was no overlap in the sentence texts between the training and test utterances.

3.5.4 Adaptation Procedure

I trained a set of baseline and correction-specific acoustic models (figure 3.12). The baseline models were adapted on increasing amounts (25%–100%) of the `pre` and `adapt` utterances. This set of baseline models provided a check on how much adaptation gain was the result of adapting to the microphone and acoustic environment. A correction acoustic model was adapted on just the `err1-4` utterances. I also adapted a model using 100% of the `pre` and `adapt` utterances plus the `err1-4` utterances.

I adapted the original WSJ model's means and variances using maximum likelihood linear regression (MLLR) [96]. MLLR adaptation used an automatically derived 16-class regression tree.

3.5.5 Recognition Results

On the `err1-4` utterances in my test set, the original WSJ acoustic model had a WER of 48.6% (figure 3.13). Using 25% of the normally spoken training data, a substantial reduction of 5.4% absolute (11% relative) was made. Increasing the amount of normally spoken training data did not further reduce WER. Thus there was a gain from adapting to the microphone and acoustic environment. But this gain was quickly realized with a small amount of adaptation data.

The model adapted on the correction speech improved accuracy even further, reducing WER by 11.2% absolute (26% relative) compared to the original WSJ model. This was an additional 5.8% absolute (13% relative) better than the models adapted on the normally spoken speech.

The model adapted on the normally spoken and correction speech did slightly worse than the model adapted on just the correction speech (39.8% WER versus 37.4%). This may have resulted from the normally spoken full sentences (`pre` and `adapt`) hindering adaptation to the correction utterances (`err1-4`).

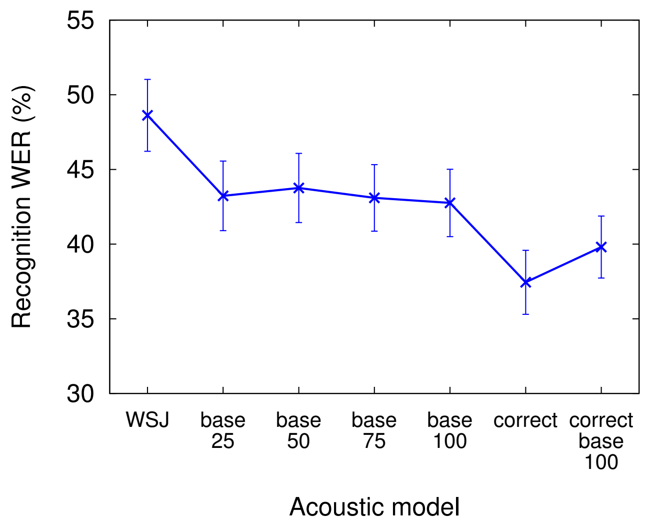


Figure 3.13: WER of speaker-independent acoustic models on spoken corrections. The "base" models were adapted on increasing amounts of normally spoken utterances. The "correct" model was adapted only on spoken corrections. The "correct+base100" model was adapted on both spoken corrections and normal utterances

3.6 Related Work

In this section, I review prior work related to speech changes during corrections, recognition of hyperarticulate speech, and the use of spoken corrections by dictation users.

There are numerous studies describing speech changes during corrections. I will highlight just a few. Levow [97] compared acoustic properties of 300 pairs of original and repeat correction utterances collected from a telephone dialog system. Each utterance pair was from the same speaker and contained the identical lexical content. Similar to my findings, repeat utterances were found to have increased duration, pausing, and changes in pitch. A decision tree using duration, pitch and amplitude features was able to classify 77% of utterances correctly as original or repeat. While Levow indicates her results "suggest different error rates after correct and after erroneous recognitions are due to a change in speaking style", no recognition experiments were conducted to confirm this.

In Oviatt et al. [115], 20 participants used a tablet PC to interact with a dialog system for conference registration and car rental. Similar to my study, they simulated recognition and displayed fake recognition errors. They analyzed the original and first repeat utterances for acoustic differences. Similar to my findings, they found participants increased their utterance duration, pause duration, number of pauses, and pitch minimum. While they speculate hyperarticulate speech may degrade performance, no recognition experiments were reported. A further study by Oviatt et al. [117] provided additional results showing that participants changed their speaking rate and pausing during correction attempts, but did not appreciably change their pitch or amplitude.

In a series of studies, Hirschberg, Litman and Swerts [68; 69; 70; 71] analyzed utterances from a dialog system for train reservations and a telephone-based dialog system for conference registration. They compared acoustic properties of utterances correctly and incorrectly recognized by the system. Utterances incorrectly recognized were higher in pitch, louder, and longer in duration. Unlike my hyperarticulate utterances, their misrecognitions had less internal silence. Utterances which they judged as hyperarticulate were found to be more likely to be misrecognized. These findings could suggest there is a basis for hyperarticulation causing increased recognition errors. However, I urge caution as there could be other explanations for the correlation. For example, maybe loud non-speech noises in utterances were causing misrecognitions and these noises in turn drove the acoustic changes seen in the misrecognized utterances. Choularton [31] also studied acoustic differences in misrecognized utterances and found correlations. Choularton also cautions that these correlations could be due to things besides hyperarticulation such as: pronunciation variation, people with colds, dysarthric speech, children's speech, or noise in the signal.

One study that actually tested recognition accuracy on hyperarticulate speech was Shriberg et al. [131]. In this study, they had participants use a dialog system for airline reservations. Utterances were judged on a three-point scale of hyperarticulation. When they performed recognition on utterances from 13 participants, strongly hyperarticulated utterances had a statistically significantly higher WER than normally spoken utterances. During the experiment, half the participants were given the instruction to speak naturally and not to over-enunciate, while the other half did not receive this instruction. Participants receiving the instruction showed a decrease in judged hyperarticulation and a lower WER, but the WER difference was not statistically significant (18% WER for those given the instruction, 23% for the rest). In further analysis in Wade et al. [155], they showed that for 20 participants, utterances judged normally spoken had a statistically significantly lower WER than utterances judged somewhat or strongly hyperarticulated (14% WER normal versus 25% hyperarticulated).

Another study that tested recognition accuracy was Bell and Gustafson [13]. In this study, they analyzed utterances collected using a Swedish spoken dialog system. They used 200 original utterances and 252 lexically identical repetitions. Similar to my findings, repetition utterances showed an increase in duration and pausing with a decrease in speaking rate. They scored utterances according to their degree of articulation (hypoarticulated, normal, or hyperarticulated). In a recognition experiment, they found *hypo*articulated utterance had the highest *sentence* error rate of 69%. They found normal utterances had a sentence error rate of 40% and hyperarticulated utterances had a sentence error rate of 45% (error rates estimated from figure 6 in [13]).

Soltau and Waibel [133] found that recognition accuracy suffered on isolated, hyperarticulated words. In this study, they elicited normal and hyperarticulate isolated words in German from 81 participants. They adapted a continuous speech recognizer using the participants' normally spoken isolated words. On 20 unseen test speakers, they compared recognition accuracy on normal and hyperarticulate words. While a few participants (2 out of 20) saw a 7% absolute decrease in WER on hyperarticulate speech, the majority (12 out of 20) saw an increase of 11% absolute. Further experiments in [135] showed the initial recognizer trained on normally spoken words had a WER of 20% on normal words and 25% on hyperarticulated words. Soltau and Waibel also collected an English corpus of isolated words [134]. Similar to their German Corpus, they report the English corpus had a lower WER on normal words than on hyperarticulated words (23% WER normal versus 30% hyperarticulated).

In contrast to other studies, Stent et al. [136] showed that hyperarticulate speech actually *improved* accuracy. In this study, 16 participants composed responses to a series of 66 questions. While participants thought the computer was recognizing their responses, in fact an unseen human was transcribing their speech. Preplanned recognition errors were inserted at fixed locations in the dialog. They found participants slowed their speaking rate after simulated errors and that this slowing persisted for a period of time after the error (despite subsequent correct recognitions). This "memory effect" of past errors causing persisting hyperarticulation is similar to what I saw on my participants' `init` utterances. Stent et al. went on to compare recognition accuracy on 387 pairs of original and repair (hyperarticulated) utterances. They used the Sphinx-3 recognizer trained on either broadcast news data or spoken dialog data. They tested using word list, unigram, bigram and trigram language models. Across all language models and both training sets, the repair utterances always had a lower WER. They also tested grammar-based recognition using a "state-of-the-art speaker-independent commercial speech recognizer". The grammar-based recognizer had the same average WER for original and repair utterances.

In the context of a dictation application, recognition of spoken corrections is only important if users are actually correcting errors using this strategy. A number of studies have looked at how dictation users correct errors. In a study by Karat et al. [80], 24 participants used commercial recognizers from Dragon, IBM, and L&H. In the speech condition, participants dictated text and then corrected any errors. Participants could use either speech or the keyboard and mouse to correct errors. In 61% of correction episodes, participants either selected or deleted text and then reentered it. Unfortunately, they do not provide data on how often users used speech correction versus keyboard correction. In a questionnaire, participants indicated a high-level of dissatisfaction with recognition accuracy and the correction process.

Further details about the Karat et al. study [80] appear in Halverson et al. [66]. Similar to my findings, they found spoken corrections were particularly hard to recognize with a WER of 53%. Despite the low accuracy on spoken corrections, their participants tended to be "fixated on re-dictation", trying repeatedly to obtain correct recognition on a particular word. 50% of the time, participants tried to re-dictate a word three times, 25% of the time they tried four times.

Horstmann [86] presents a study in which 24 expert dictation users with physical disabilities used speech recognition to enter text and perform command-and-control tasks. All participants had substantial (6+ months) of experience using their particular commercial recognizer (primarily Dragon). When correcting errors, participants could use a correction dialog which allowed selection of alternate hypotheses, spelling by voice, or spelling by keyboard. Horstmann's participants preferred the recognizer's correction dialog, using it for 38% of corrections. This level of usage of the correction dialog was much higher than the 8% seen of novices in Karat's study [80]. The increased correction dialog usage might be because expert users had learned to avoid error-prone correction methods such as respeaking. It could also be that the users were avoiding keyboard-based corrections due to physical inability or discomfort. Despite preferring to use the correction dialog, Horstmann's participants still used spoken corrections in 25% of their correction episodes. This demonstrates that even for experienced users, spoken corrections are an important part of the correction process.

3.7 Conclusions

In this chapter, I described a corpus I collected using a dictation-style interface. Users first spoke full sentences and then they spoke corrections based on a series of simulated recognition errors. Using this corpus, I showed participants had a strong tendency to change their speech. Compared to naturally read speech, speech during error corrections showed a slowed speaking rate, increased inter-word pausing, expanded pitch range, expanded intensity range, and increased formant frequencies. I found human-judged levels of hyperarticulation increased during error correction episodes. I also found judged hyperarticulation increased even before an error had occurred, possibly as a response to errors encountered on previous sentences.

3.7 Conclusions

While I had expected an increase in recognition errors on hyperarticulated speech, my experiments showed otherwise. Despite increasing levels of hyperarticulation on repeated full sentences, recognition error rates remained similar or even decreased. Within-subject pairings of word, partial sentence, or full sentence utterances did show some differences between normal and hyperarticulate speech. But these differences were small and Dragon actually showed improved recognition on hyperarticulate speech. It seems that the exact influence of hyperarticulation on recognition accuracy is variable. Most prior work reports increased errors on hyperarticulate speech in the domain of spoken dialog systems [131; 155] and isolated word recognition [133; 134; 135]. However, one prior study showed decreased errors on hyperarticulate sentences [136]. Perhaps the exact nature of the task and the recognizer used influences whether hyperarticulation is helpful or harmful.

I found recognition of word or partial-sentence corrections was particularly hard. I investigated improving recognition on these types of corrections by adapting the acoustic model to short corrections. I found this adaptation reduced the word error rate by 13% relative.

Chapter 4

Speech Dasher

4.1 Overview

Speech offers a potentially very fast way to enter text into the computer. Users have been measured dictating text to a computer at 102 (uncorrected) words per minute (wpm) [93]. But speech recognition isn't perfect and mistakes need to be corrected. Previous research has shown correction time dominates the entry process and significantly slows entry rates (e.g. 14 wpm [80], 17 wpm [86], 17 wpm [93]). A further problem is that correction interfaces often assume the user can control a non-speech input device such as a mouse, keyboard, or stylus. This makes correction difficult for people who have limited or non-existent use of their hands.

In this chapter, I investigate Speech Dasher, a novel correction interface for speech recognition. Speech Dasher works by allowing the user to easily navigate the rich hypothesis space available to the speech recognizer. I will show that expert users were able to use Speech Dasher to write at 40 (corrected) words per minute. They did this despite a recognition word error rate of 22%. Furthermore, they did this by using only speech and the direction of their gaze (obtained via a gaze tracker).

Speech Dasher is built upon the existing text entry interface Dasher [157]. In

Dasher, users write by zooming through a world of boxes. As shown in figure 4.1, Dasher's main navigation display consists of a collection of nested boxes with each box labeled by a letter. The vertical size of each letter's box is proportional to the letter's probability under a letter-based language model. As more letters are written, Dasher makes stronger and stronger predictions about the upcoming letter. This enables the user to write common words quickly. All letters appear inside every box in alphabetical order, from top to bottom. Even if the user's desired text is not well predicted by the language model, the text can still be written by navigating to the correct location in the alphabetical ordering.

Dasher can be controlled by any type of pointing device. The rate of zooming is controlled by the horizontal position of the mouse. If the user points to the right of the screen midpoint (the vertical line in figure 4.1), Dasher zooms in on letters. The speed of zooming is controlled by how close the mouse is to the right side. As a letter box passes through the midpoint, that box's letter is outputted. The maximum rate of zooming is controlled by a user selected speed setting. If the user makes a mistake, pointing to the left side of the screen reverses the zooming direction and allows previously written letters to be erased. After an hour of practice, users can write at 20 words per minute (wpm) with a mouse [157] or 16–26 wpm using a gaze tracker [158].

In this chapter, I augment Dasher by adding information from a speech recognizer. This enables users to write faster than using Dasher alone. In Speech Dasher, users navigate as in normal Dasher, but can quickly zoom through complete words and parts of sentences. Even if the recognition error rate is very high, Speech Dasher allows easy correction via a letter-by-letter spelling of the desired word.

This chapter is structured as follows. First, I describe the principles that guided my design of Speech Dasher. Second, I provide an overview of the Speech Dasher interface. Third, I describe the probability model that Speech Dasher uses to navigate the recognition hypothesis space. Fourth, I describe results of a user study in which participants used Speech Dasher to write with a gaze tracker.

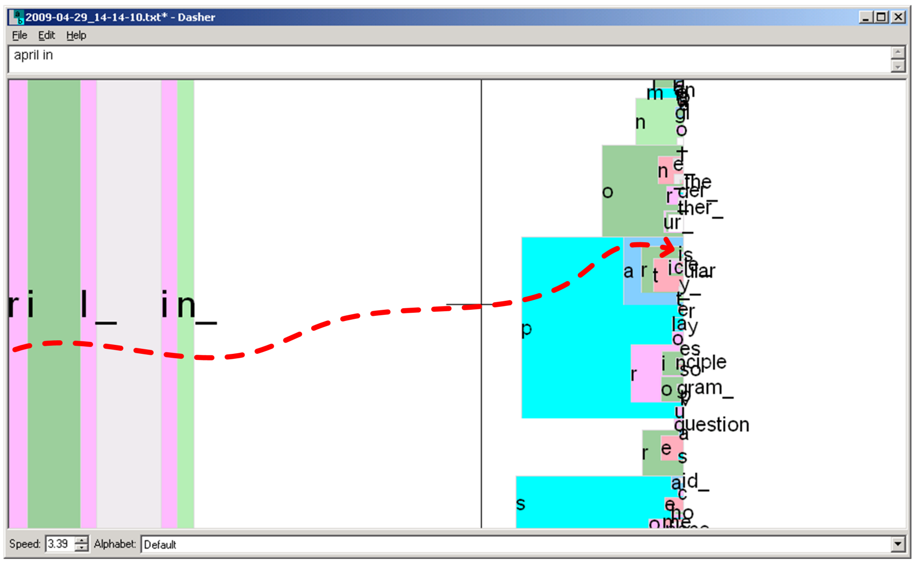


Figure 4.1: The standard Dasher interface after the user has written "april in". The dashed red line shows the path a user might take to write "april in paris".

4.1.1 Publication Note

The work presented in this chapter extends the work in my M.Phil thesis "Efficient Computer Interfaces Using Continuous Gestures, Language Models, and Speech" [146]. I repeat from [146] the basic formulation of the lattice probability model (sections 4.4.2 and 4.4.3). The model has been extended in numerous ways in this work, including the use of a fully probabilistic recognition lattice, optimizing the model and display for word-level predictions, and adding a special "escape" box.

4.2 Design Principles

In this section, I discuss some of the key principles that guided my design of Speech Dasher. These principles leveraged existing results in the literature as well as user feedback I obtained from early prototypes.

4.2.1 Visualize the Hypothesis Space

Standard speech recognition interfaces like Nuance Dragon NaturallySpeaking first present only the recognizer's top hypothesis. In order to see other possibilities, the user highlights part of the top hypothesis and issues a command. But doing this only exposes a small number of other hypotheses. The recognizer has a much larger set of hypotheses within the lattice generated during the speech recognition search. I designed Speech Dasher to allow the user to navigate the entire hypothesis space of the recognizer. By exposing this space to the user, the user can easily and efficiently arrive at one of the recognizer's alternative hypothesis. This allows the user to correct a large number of errors without resorting to entering the information from scratch.

4.2.2 Avoid Cascading Errors

Previous research has shown that using speech to correct speech recognition errors can lead to time-consuming and frustrating cascades of errors [66; 80]. I designed Speech Dasher so that correction could be achieved via transparent mechanisms that did not rely on further use of error-prone recognition technologies. As I will describe shortly, I added a highly visible "escape" box. By going to the escape box, the user can expect a consistent correction experience similar to conventional Dasher.

4.2.3 Efficient when Correct

Given reasonably accurate speech recognition, the best recognition hypothesis will often be completely, or nearly completely, correct. A design choice was whether the user should proofread the best recognition result as normal, static text, or whether they should proofread the text by zooming in Dasher. In my opinion, proofreading static text requires less time and mental effort than proofreading the same text in Dasher. For this reason, Speech Dasher first presents the best hypothesis as simple text. The user can then choose to accept the result, selectively correct portions of the result, or correct the entire sentence.

This also makes it easier to imagine how Speech Dasher might integrate into a real-world text creation environment. A user could add text to a document by speaking and having the best recognition added to the document (as in conventional dictation interfaces like Dragon). When an error needs to be corrected, Speech Dasher would be invoked by selecting the desired text and issuing a command. The user could then correct just the erroneous portion of the text using Speech Dasher. In this way, the user could focus on their primary task of document creation and not devote attention or screen real estate to Speech Dasher until it was actually needed.

4.2.4 Present Entire Words

While Dasher is normally letter-based, it can in principle predict any type of lexical unit. I discovered in early prototype versions that users disliked a letter-by-letter approach to correcting speech results. I also found that allowing the user too many choices made navigation difficult. This was because all but the most probable choices tended to be small in the Dasher display. This made it difficult for the user to successfully navigate to the less probable choices. I updated my design to present results at the word level. This allows users to zoom through the majority of their desired sentence, escaping out to letter-by-letter spelling only when the model's most likely predictions are incorrect.

4.3 Interface Description

In this section, I provide an overview of how the Speech Dasher interface works from a user perspective. Figure 4.2 shows the Speech Dasher interface. The main area of the window is the Dasher display. It displays words and letters that are used to correct the recognition result. The buttons along the bottom row allow

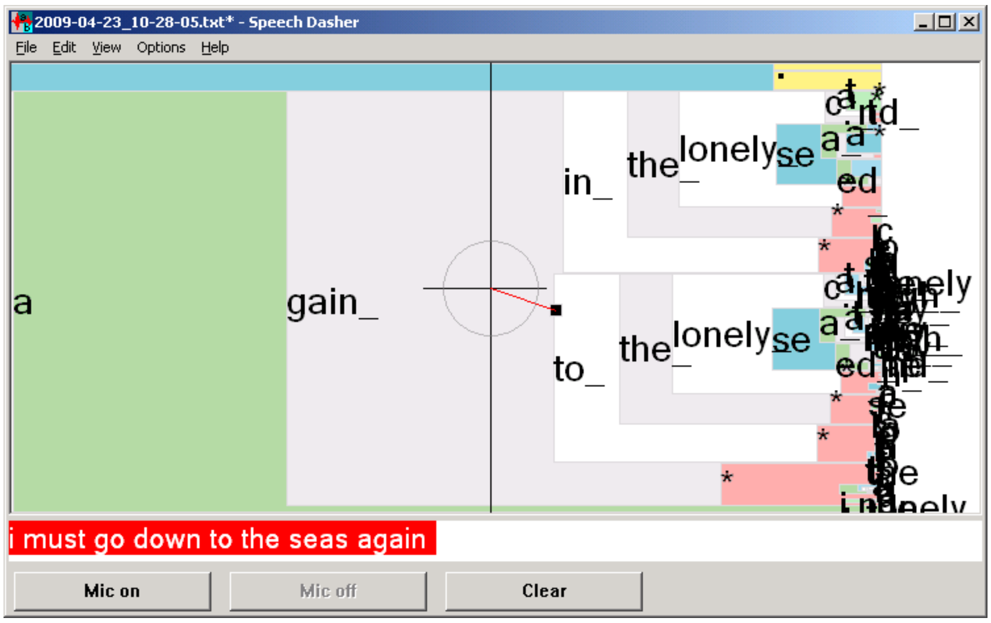

2009-04-23_10-28-05.txt* - Speech Dasher
File Edit View Options Help
a
gain_
in_
the
lonely
se
ed
to_
the
lonely
se
ed
i must go down to the seas again
Mic on
Mic off
Clear

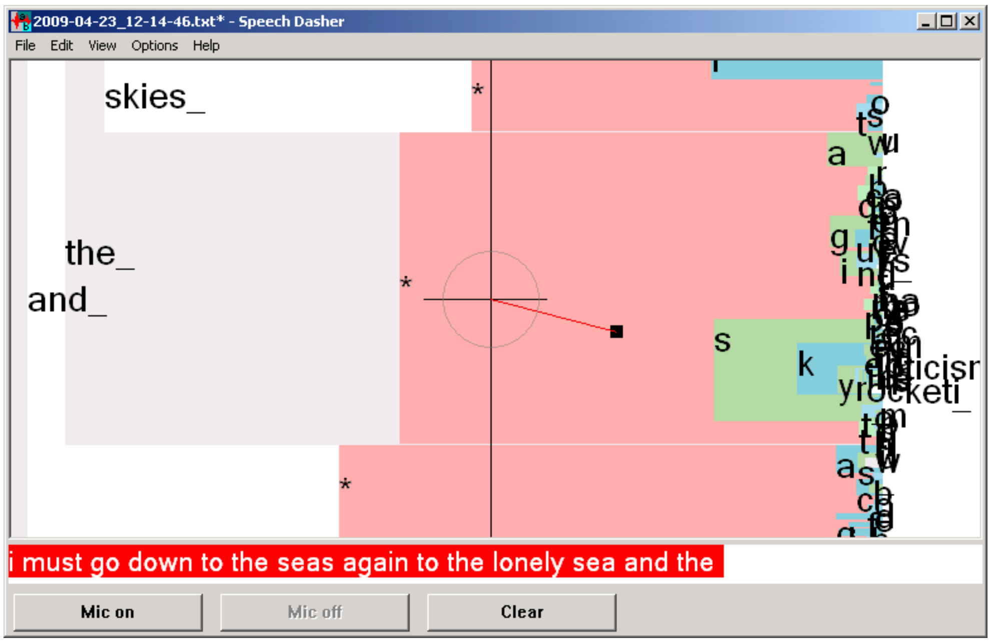
Figure 4.2: The Speech Dasher interface. The user is currently midway through the sentence "i must go down to the seas again to the lonely sea and the sky". The user must now choose between the word "in" or "to".

control of the microphone and allow the current result to be cleared. The text box above the buttons displays the current text output of the correction process.

4.3.1 Basic Navigation

Writing in Speech Dasher is done by zooming through the alternative recognition hypotheses in the results returned by the recognizer. This is best illustrated by an example. Figure 4.2 shows a user correcting the sentence "I must go down to the seas again to the lonely sea and the sky". At the point shown in figure 4.2, the recognizer was unsure if the user said "in the lonely sea", "in the lonely seed", "to the lonely sea", or "to the lonely seed". The user must now choose which branch to take within the recognition hypothesis space.

Primary predictions are the words that Speech Dasher thinks are most probable at the current location. Primary predictions appear in alphabetical order in the Dasher display. In figure 4.2, the words "to" and "in" are the primary


2009-04-23_12-14-46.txt* - Speech Dasher
File Edit View Options Help
skies_
the_
and_
*
Mic on
Mic off
Clear

i must go down to the seas again to the lonely sea and the

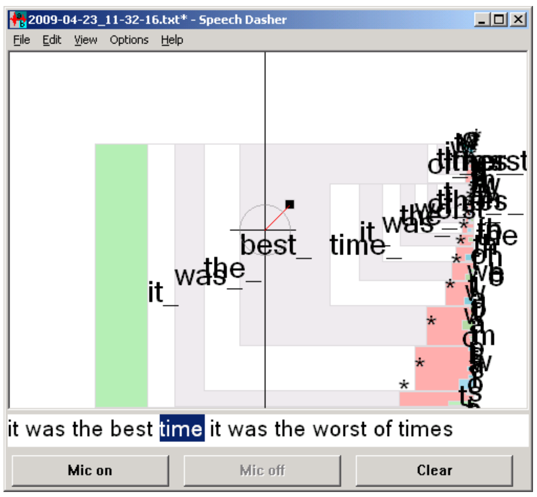
Figure 4.3: The user wants "sky" but the primary prediction was "skies". By going into the escape box, the user can spell "sky" using a combination of information from the speech recognition result and from a general model of English.

predictions. The primary predictions are always big and easy to navigate to.

The recognizer may also have a set of much less probable word predictions. These *secondary predictions* always appear below the primary predictions inside the *escape box*. The escape box is a red box labeled with an asterisk that appears at word boundaries. The user navigates to the escape box when Speech Dasher's primary predictions are wrong. Once inside the escape box, the Speech Dasher model offers the secondary predictions as well as all the other letters of the alphabet. This makes it possible to spell any word, regardless of whether it appeared in the recognition lattice. Figure 4.3 shows an example of using the escape box to perform a correction.

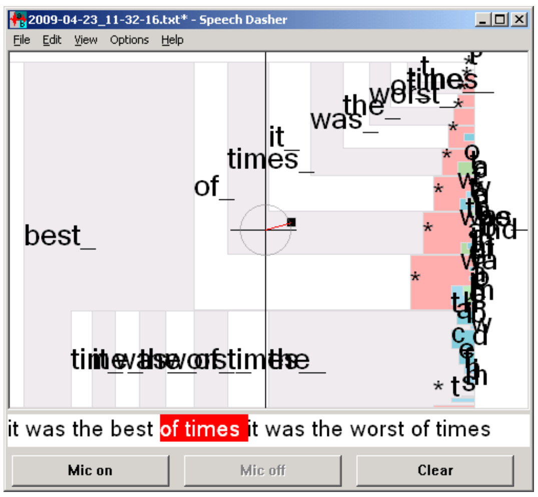
4.3.2 Correction Styles

Speech Dasher supports two different styles of performing corrections:


2009-04-23_11-32-16.txt* - Speech Dasher
File Edit View Options Help
Mic on
Mic off
Clear

it was the best time it was the worst of times

(a) Selecting the error


2009-04-23_11-32-16.txt* - Speech Dasher
File Edit View Options Help
Mic on
Mic off
Clear

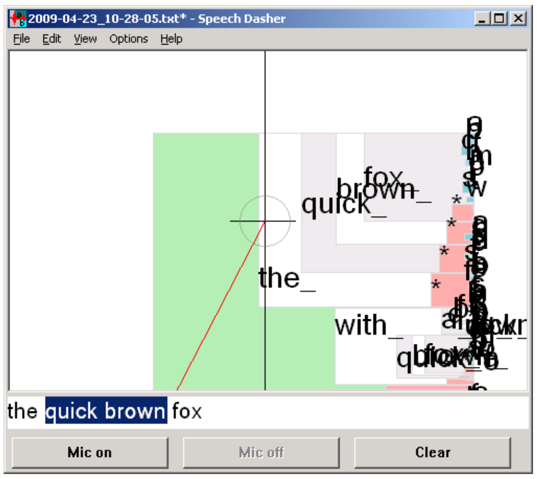
it was the best of times it was the worst of times

(b) After correction

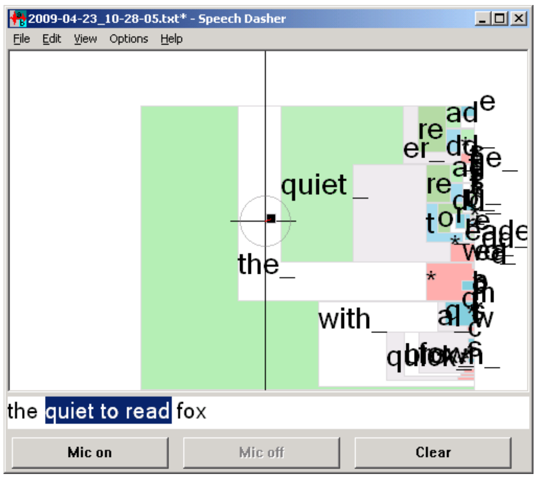
Figure 4.4: Example of using selective correction. The user has selected "time" in the text box using the mouse (left). The Dasher display is updated to show this portion of the hypothesis space. The blue highlighting of the text indicates that this text will be replaced by anything written via Dasher navigation. The user then navigates in the Dasher display to the alternate hypothesis "of times" (right). The blue highlighted text is replaced with the red highlighted text during navigation.

- **Full navigation** – In full navigation, the user simply navigates through the entire hypothesis space. This method can be thought of as combining proofreading and correction into one procedure. This style of correction is good when the best recognition result has many errors. Figure 4.2 shows an example of the full navigation style.

- **Selective correction** – In selective correction, the user highlights errors in the text result box using the mouse. The Dasher display is then updated to show that part of the hypothesis space. Whatever the user then writes via Dasher navigation is used to replace the highlighted text in the text result box. This style of correction is good when the best recognition result has only a few errors. Figure 4.4 shows an example of the selective correction style.


2009-04-23_10-28-05.txt* - Speech Dasher
File Edit View Options Help
the_
quick_
brown_
fox_
with_
the quick brown fox
Mic on
Mic off
Clear

(a) Before spoken correction


2009-04-23_10-28-05.txt* - Speech Dasher
File Edit View Options Help
the_
quiet _
with_
the quiet to read fox
Mic on
Mic off
Clear

(b) After spoken correction

Figure 4.5: Speech Dasher after the user speaks "the quick brown fox" (left). The user decides to change the text to "the *quiet red* fox". The user highlights "quick brown" with the mouse, turns on the microphone, and says "quiet red". The spoken correction is integrated into the original hypothesis space (right). While the best hypothesis of the spoken correction was incorrectly recognized as "quiet to read", the user can select the correct text by navigating in Dasher.

4.3.3 Spoken Corrections

In either correction style, users can use speech to help correct their text. The user does this by speaking the words he or she wants at a certain point in the text. The spoken correction is then integrated into the Dasher display. If the user has selected text in the result box, the selected text is replaced with the best result from the spoken correction. The user turns the microphone on or off using either the right mouse button or by using the buttons at the bottom of the interface. Figure 4.5 shows an example of a spoken correction.

4.4 Probability Model

In this section, I give the details of the probability model used by Speech Dasher. This model generates the Dasher display showing the various words in the recognition. In prior work [146], I focused on making Speech Dasher's model work with a commercial recognizer that output only a ranked n-best list. Here I instead use a lattice that contains all the probabilistic information available from the recognition process.

4.4.1 Lattice Processing

Before its use in the Speech Dasher model, the recognition lattice undergoes several processing steps. These steps serve to keep the lattice to a manageable size and also to convert the language and acoustic likelihoods in the lattice to posterior probabilities. The posterior probabilities are used by the Speech Dasher model to assign penalties to the various recognition hypotheses. The lattice processing steps are as follows:

- **Forward/backward reduction** – Redundant lattice nodes were combined with a single forward and backward reduction pass [159]. This process maintains the same lattice paths and probabilities while making the lattice smaller.
- **Posterior pruning** – Lattice nodes with a posterior probability less than a fixed constant times the posterior probability of the best path were removed. This removed low-probability hypotheses resulting in a smaller lattice.
- **Compact trigram expansion** – The lattice was expanded and rescored using a compact trigram expansion algorithm [159]. This allowed the proper trigram language model probabilities to be assigned to all paths in the lattice. This was necessary because Sphinx, despite decoding with a trigram, only exposed a bigram recognition lattice (see chapter 5, section 5.5.5).
- **More posterior pruning** – Another round of posterior pruning was performed to reduce the size of the expanded lattice.

After the above steps, the recognition lattice is simplified to contain just the information used by the Speech Dasher probability model. This removes timing information, pronunciation probabilities, acoustic likelihoods, and language

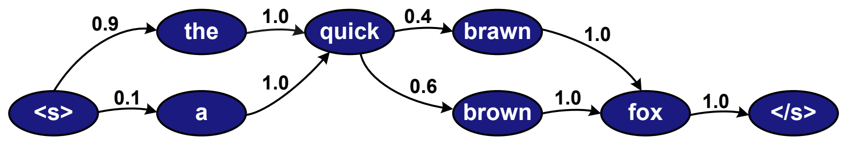


Figure 4.6: Lattice after simplification and the addition of the edge penalties used by the Speech Dasher model. The penalties are based on the posterior probabilities of the nodes in the lattice.

model likelihoods. The simplified lattice consists of a set of nodes where each node has a word label, a posterior probability, and a set of nodes reachable from that node.

In order to compute the symbol probabilities required by Dasher, Speech Dasher compares different paths in the lattice to see how likely they are. A path in the Speech Dasher model incurs a penalty for every edge it traverses. Initially, the penalty on each edge of the lattice is set to the posterior probability of the node the edge goes to. An example lattice after simplification and the addition of edge penalties is shown in figure 4.6.

After the above processing steps, I perform an additional step in which edges are added that skip over words. By adding these skip edges, extra hypotheses are added to the lattice that cover all one-word recognition insertion errors. So for example, if the recognition lattice has the hypothesis "the quick brown fox", Speech Dasher would allow the user to write "the brown fox“, "the quick fox", and so on. I found in previous work [146], that this technique helped reduce the amount of information required to perform corrections. The penalty of each added edge was set to be a constant α_{ins} multiplied by the penalties for all edges skipped over in the original lattice. So for example, in figure 4.7 the penalty for the new edge from "quick" to "fox" is $\alpha_{\text{ins}} \cdot (0.4 \cdot 1.0 + 0.6 \cdot 1.0)$.

4.4.2 Computing the Symbol Probabilities

Each box in Dasher requires a probability distribution over all symbols in the alphabet. The symbol alphabet $\boldsymbol{S}$ contains both letters and the word boundary

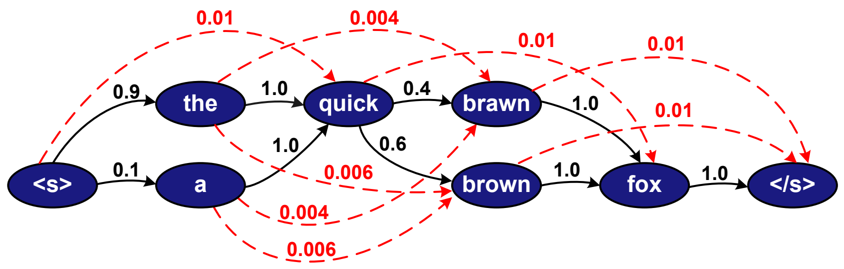


Figure 4.7: Lattice after the addition of new edges that cover all one-word recognition insertion errors. The new edges are shown as dashed red lines. The calculation for the new edges' penalty values used $\alpha_{\text{ins}} = 0.01$.

symbol (denoted here by an underscore). The goal of the Speech Dasher model is to compute:

$$P(s_{t+1} \mid \text{lattice}, s_1, ..., s_t) \tag{4.1}$$

where $s_1, ..., s_t \in \boldsymbol{S}$ are the t symbols that have been written thus far in Dasher.

Speech Dasher's model computes this probability by first finding the set of paths in the lattice that are consistent with the symbol history. For the moment, I will assume that at least one such path exists. So for example, given the lattice in figure 4.7, if the symbol history is "the_quick_br", then there are two possible paths. One path goes to the "brawn" node and the other goes to the "brown" node. Given these two paths, Speech Dasher predicts that the next symbol would be either "a" or "o". The model assigns probabilities to the two letters based on the total penalties incurred by each path as it traversed edges in the lattice.

I will denote a path by ρ and the penalty value of a path by $V(\rho)$. Let $\boldsymbol{C}(s_1, ..., s_{t+1})$ be the collection of paths that are consistent with the symbol history $s_1, ..., s_{t+1}$. The probability of the next symbol is:

$$P(s_{t+1} \mid \text{lattice}, s_1, ..., s_t) = \frac{\sum_{\rho \in C(s_1,...,s_{t+1})} V(\rho)}{\sum_{s_x \in \boldsymbol{S}} \sum_{\rho \in \boldsymbol{C}(s_1,...,s_t,s_x)} V(\rho)}. \tag{4.2}$$

4.4.3 Backing Off

A particular sequence of symbols may not always be in the lattice. For example, in figure 4.7 there are no paths consistent with "the_quie". This results in the model predicting zero probability for all symbols in the alphabet. This prevents words not in the lattice from appearing in the primary predictions in Speech Dasher. Such out-of-lattice words can be written via the secondary predictions that appear inside the escape box. In section 4.4.6, I will describe how symbol probabilities are calculated in the escape box.

For the time being, assume the user has used the escape box to write an out-of-lattice word. After completing the word in the escape box, Speech Dasher tries to get the user back on track somewhere in the lattice. It does this by assuming the recognizer has committed a deletion or substitution error somewhere in the lattice. A new search for paths is initiated, but this time each path is allowed to make a single deletion or substitution error.

During the new search, paths are assessed a penalty when they use their allowed error. A deletion error is assessed a penalty of α_{del} and substitution errors are assessed a penalty of α_{sub}. Using the collection of paths that are allowed to make an error, the probability distribution over symbols can be calculated using (4.2). If no paths are found using a single error, all paths are allowed to make two errors and the search is repeated. The number of errors continues to be increased until at least one path is found matching the lattice.

Figure 4.8 show an example of the backing off procedure. Once a non-empty set of paths is found, the model can once again make symbol predictions based on the penalties of the set of paths. A path's total penalty is the multiplication of all penalties incurred traversing edges in the lattice plus any α_{sub} or α_{del} penalties associated with the path making substitution or deletion errors.

4.4.4 Pruning

There are two types of pruning used by the Speech Dasher model. The first type is used to keep the search for possible paths tractable. As just described, every time the model backs off, paths are allowed an additional word error. This causes a large growth in the number of possible paths in the lattice. During the path

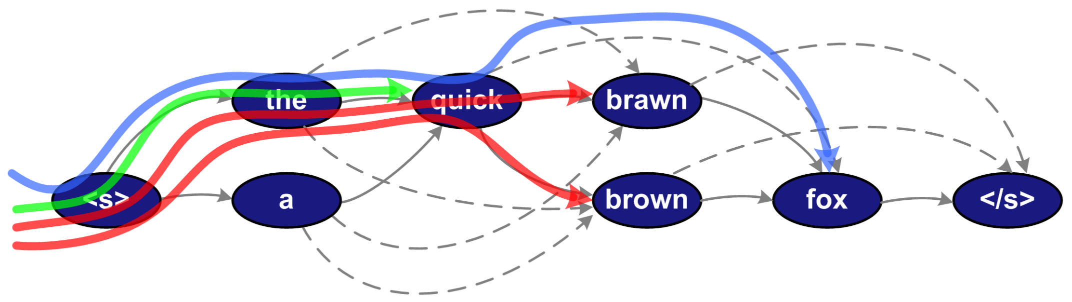


Figure 4.8: The user has written the "the_quiet_". A substitution error at the node "quick" allows the two red paths to reach the "brawn" and "brown" nodes. A substitution at the "quick" node also allows the blue path to reach "fox" via the insertion edge between "quick" and "fox". Using a deletion error after "the", the green path can reach the "quick" node. So at this point, Speech Dasher would predict the user is going to write the letter "b", "f" or "q".

search, I prune out all but the top β_{prune} paths. The paths are pruned based on their penalty value.

The second type of pruning is used to remove unlikely hypotheses from the primary prediction part of the display. This pruning uses a threshold value on the symbol probability from (4.2). Symbols with a probability less than α_{min} are set to zero. The symbol probabilities are then renormalized. This pruning only applies in the primary prediction part of the Dasher display. It does not apply when inside the escape box. This pruning has the effect of causing the primary predictions to be highly peaked at only the most probable word hypotheses.

4.4.5 PPM Language Model

As I will describe next, the escape box in Speech Dasher makes use of a letter-based language model. This language model is based on the text compression algorithm prediction by partial match (PPM) [32]. Speech Dasher uses PPM to generate the probability distribution over letters based on the previous letter history. In PPM terminology, the number of previous symbols conditioned on is called the *order*. An order-1 model conditions on one previous letter (a bi-gram language model), an order-2 conditions on two previous letters (a trigram), etc. Information is shared between different context lengths according to PPM's escape mechanism. Dasher uses the PPM-D escape mechanism [72] which is a slight modification of PPM-C [107]. The PPM language model can adapt to the user's writing by updating its counts based on the text seen. For further details about PPM, see [14; 143].

4.4.6 Secondary Predictions

As previously discussed, some words may be missing from the lattice. Speech Dasher allows writing of any word in the secondary predictions. These predictions appear inside the special escape box. The escape box appears only at word boundaries and is allocated a fixed amount of conditional probability. In this work, I used 20% of the probability mass for the escape box. Once inside the escape box, the model interpolates between four different models:

- **Lattice paths** – A search for paths is conducted based on the symbol history. Inside the escape box, no minimum probability threshold is used. Additionally, the highly probable paths that appeared outside the escape box are given zero probability inside the escape box. The interpolation weight for this model is λ_{lat}.
- **Uniform lattice paths** – The search for paths is done as normal, but the penalty of each path is set uniformly. This smooths the distribution over paths, allowing low scoring paths to still have an appreciable size in the Dasher display. The interpolation weight for this model is λ_{uni}.
- **PPM** – The PPM language model is used to predict symbols based on the previous symbol history. The interpolation weight for this model is λ_{ppm}.
- **Shortened PPM** – The PPM language model makes predictions using a context that stops at the first word boundary symbol. This makes the PPM prediction less sharply peaked since only the word currently being spelled out is used to predict upcoming letters. The interpolation weight for this model is λ_{short}.

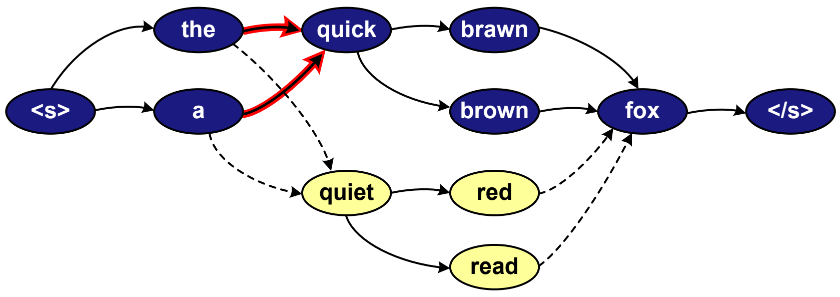


Figure 4.9: Example of the spoken correction "quiet red" being spliced into an existing lattice. The blue nodes are the original lattice. The yellow nodes are from the spoken correction. The dotted edges show the edges that splice the new lattice into the original. The edges highlighted in red show the original lattice edges that were assessed a penalty.

A symbol's probability inside the escape box is the sum of its probability under each of these four models multiplied by the λ interpolation weights. The interpolation weights sum to one. In this work, the interpolation weights were set to be equal. This resulted in half the probability mass in the escape box being allocated according to the general language model. In testing, I found this was a good compromise that provided a relatively consistent user experience while still providing the ability to utilize less probable lattice paths.

4.4.7 Spoken Corrections

Spoken corrections are accomplished by splicing the lattice of the spoken correction into the original lattice. The combined lattice then becomes the basis for Speech Dasher's probability estimates. The lattice is spliced in at a location determined by the currently selected text (in the case of selective correction) or by the current location in the Dasher display (in the case of navigation correction). At the location where the lattice is spliced in, all other outgoing edges are assessed a penalty of α_{respeak}. This penalty causes the old predictions to be deemphasized. Figure 4.9 shows an example of splicing a correction lattice into an existing lattice.

4.5 Speech Recognition

For speech recognition, I used CMU Sphinx and the PocketSphinx decoder [74]. While PocketSphinx was designed for mobile devices, I found it also provided fast and accurate recognition on a desktop computer. In this section, I describe the details of the recognition setup I used for my Speech Dasher experiment.

4.5.1 Acoustic Model

I trained both US and UK English acoustic models following the recipe described in [147]. I used an HMM topology of 5 states with skip transitions. I trained cross-word triphones using 39 CMU phones without stress markings plus silence. I parameterized audio into a 51-dimensional feature vector consisting of 12 Mel-frequency cepstral coefficients plus their short-term deltas, long-term deltas, delta deltas, and three 0^{th} cepstral power terms.

My US English model was trained on 211 hours of WSJ [5; 57] training data. I used a semi-continuous model with 1024 codebook Gaussians, 8000 tied-states, and the CMU pronunciation dictionary.

My UK English model was trained on 16 hours of WSJCAM0 training data. I used a semi-continuous model with 256 codebook Gaussians, 4000 tied-states, and the BEEP pronunciation dictionary. I mapped the BEEP phone set to the CMU phone set and added missing words from the CMU pronunciation dictionary. From the original gender-independent model, I create gender-dependent models using maximum likelihood linear regression (MLLR) adaptation [96] of the means followed by maximum a-posteriori (MAP) adaptation [59] of the means, mixture weights and transition matrices.

After recognition, lattices were post-processed as described in section 4.4.1. This reduced their size and prepared them for use in the Speech Dasher model. During the posterior pruning step, I pruned lattice nodes that had a posterior probability less than 1×10^{-3} times the posterior probability of the most probable lattice path. In the user study (to be described in section 4.6), recognition was performed on a total of 1008 utterances. Before post-processing, the original recognition lattices had an average density of 416 words per second, a 1-best WER of 25%, and an oracle WER (the lowest WER of any lattice path) of 7.5%. After post-processing, the pruned lattices had an average density of 61 words per second, a 1-best WER of 23%, and an oracle WER of 9.3%. The original lattices had a higher 1-best WER because the lattices returned by Sphinx had only bigram language model probabilities. In the post-processing phase, I rescored the lattice with a trigram language model and this improved the 1-best WER.

4.5.2 Audio Capture and Normalization

Audio was recorded at 16 kHz using a wired Sennheiser PC 166 microphone. The audio was streamed to the recognizer as soon as the microphone was enabled. Audio was normalized using cepstral mean normalization/subtraction [99] based on a prior window of audio.

4.5.3 Language Model

I trained a trigram language model using: newswire text from the CSR-III text corpus (222M words), interpolated modified Kneser-Ney smoothing, and a vocabulary of the top 64K words in the corpus. I trained the initial language model with no count cutoffs and then performed entropy-pruning [137] using a threshold of 1×10^{-8}. The resulting language model had 8.7M bigrams and 12.4M trigrams.

4.6 User Study

In this section, I describe the longitudinal user study I conducted to investigate whether speech improved users' ability to write with Dasher. In particular, I investigated people controlling Dasher using only their gaze (via a gaze tracking device). This combination of gaze-tracking and speech might be useful for people with disabilities that have both eye and speech control but cannot use a conventional keyboard or mouse.

I used a within-subjects experimental design with two conditions:

1. **Dasher** – Participants used conventional Dasher (without speech input) to write sentences.
2. **Speech Dasher** – Participants spoke sentences to a speech recognizer and then corrected any errors using Speech Dasher.

4.6.1 Software Setup and Parameters

The experimental software used in both conditions was based on Dasher v3.2.0. The eye tracker mode of Dasher was enabled. This mode enables users to reverse by looking to the extreme top or bottom of the *right* side of the display. This allows users to reverse while continuing to look for their desired string.

The PPM language model used in both conditions was trained on 25M words of newswire text from the CSR newswire corpus. As the participants were writing (unseen) newswire sentences, this setup tests the situation where Dasher's language model is well adapted to the user's writing style. The PPM model made predictions based on the prior 7 letters of context. I used an alphabet of 26 lower case letters plus apostrophe and period. An implicit context of a period followed by a space was assumed at the start of every sentence. The adaptation of PPM was turned off for the study. Adaptation would have had little effect as the model was well trained on newswire text and the participants were writing text in a similar genre. Adaptation also would have made it more difficult to separate gains made by proficiency using gaze tracking versus changes due to adaptation of the language model.

The Speech Dasher probability model has a number of free parameters. Table 4.1 lists the settings I used for the user study.

Parameter	Value	Description
α_{sub}	0.08	Path penalty for substitution errors
α_{ins}	0.01	Path penalty for insertion errors
α_{del}	0.01	Path penalty for deletion errors
α_{min}	0.20	Minimum probability for primary predictions
λ_{lat}	0.25	Lattice model escape box interpolation weight
λ_{uni}	0.25	Uniform lattice model escape box interpolation weight
λ_{ppm}	0.25	PPM model escape box interpolation weight
λ_{short}	0.25	Shortened PPM model escape box interpolation weight
β_{prune}	32	Maximum paths to keep during search

Table 4.1: Parameter settings of the Speech Dasher model as used in the user study.

4.6.2 Participants and Apparatus

I recruited 3 participants. All participants were able-bodied and had no (uncorrected) visual impairment. Here are the relevant details about each participant:

- **US1** – This participant was a native speaker of North American English and used the US acoustic model. This participant had no prior experience using a gaze tracker and had used Dasher with a mouse for about an hour prior to the study.
- **UK1** – This participant was a native speaker of British English and used the UK acoustic model. This participant had significant experience both with gaze tracking and with Dasher.
- **DE1** – This participant was a non-native speaker of English and used the US acoustic model. This participant had no prior experience using a gaze tracker and had never used Dasher.

Participants used a MyTobii P10 gaze tracker (figure 4.10). The P10 is a single unit that combines the gaze tracking camera, an LCD monitor, and a Windows XP computer running at 1.5 GHz. The interface was presented at a resolution of 1024 × 768 and occupied the entire 15" screen.

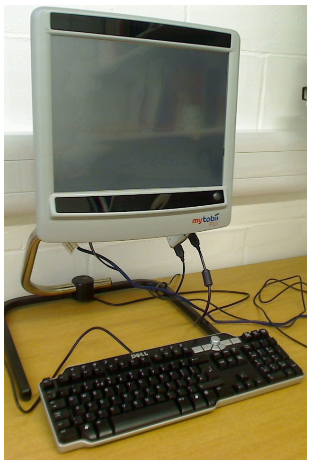


Figure 4.10: The MyTobii P10 gaze tracker used in the study. The tracking camera is built into the screen.

4.6.3 Sessions

Each participant took part in a series of sessions which occurred over a two week period. No more than two sessions were conducted on any one day. If two sessions occurred on the same day, they were separated by at least 4 hours. There were four types of sessions:

- **Introduction** – In the very first session, the participant was given about 15 minutes to play gaze-controlled games such as tic-tac-toe. This served to familiarize the participant with using a gaze tracker. At the end of the session, the participant was shown the Dasher interface and allowed to write by gaze for about 5 minutes. In the introductory session, the participant did not use speech and was free to write anything.
- **Gaze tracking, phase 1** – Participants wrote for 15 minutes in one condition (either Dasher or Speech Dasher). After a break of at least 15 minutes, participants wrote for 15 minutes in the other condition. The order of conditions was swapped after each session. The gaze tracker was calibrated at the start of each condition. Participants completed between 6 and 8 phase-1 gaze tracking sessions. This phase of sessions served to get all participants up to an expert level of performance using Dasher and the gaze

tracker. Also during this phase, various changes were made to the gaze tracker settings to improve Dasher navigation (see section 4.6.4).

- **Gaze tracking, phase 2** – Participants did a final three gaze tracking sessions. The procedure was the same as in the phase 1 sessions. These sessions represent expert level performance and used the optimum gaze tracking settings found during phase 1.
- **Mouse** – After the conclusion of all gaze tracking sessions, participants did one additional session using a conventional mouse instead of the gaze tracker.

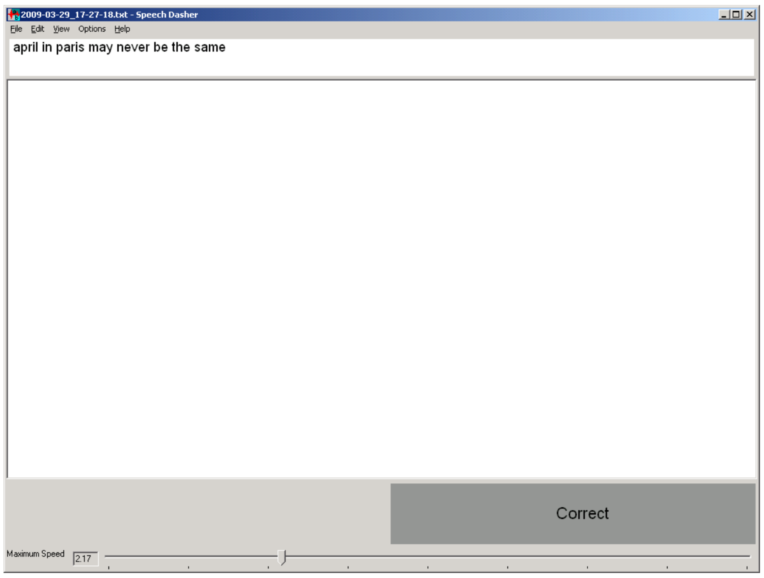
4.6.4 Mouse Smoothing

The MyTobii P10 gaze tracker had two settings that affected the movement of the mouse pointer, *speed* and *fixation sensitivity*. For the first few sessions of US1 and DE1, both settings were set to normal. After this point, the speed setting was changed to “very fast”. All of UK1’s sessions used a speed of “very fast”. The fixation sensitivity was set to normal for most of phase 1’s sessions and then changed to “high” at the end of phase 1. I found these final settings allowed experienced Dasher users to navigate the fastest with the gaze tracker.

4.6.5 Interface Modifications

I made a number of modifications to facilitate the user study and to make Dasher easy to use with a gaze tracker:

- **Target sentence display** – The target sentence the user was suppose to write was displayed at the top of the window (figure 4.11). To encourage participants to read the sentence, I displayed the target sentence in teleprompter style (displaying each letter after a short delay). The sentence was also read out to the participant using the Microsoft text-to-speech (TTS) engine. At any point, the participant could press the space bar to replay the TTS audio for the target sentence.



2009-03-29_17-27-18.txt - Speech Dasher

april in paris may never be the same

Maximum Speed 2.17

Figure 4.11: The experimental interface after initial presentation of the target sentence (displayed in the top text box). The button in the lower right is activated by dwelling over it for one second. The bottom slider allowed the user to control the speed of Dasher zooming. The speed slider was changed using the mouse.

- **Dwell button** – I implemented a large dwell button in the lower part of the interface (figure 4.11). The dwell button was "clicked" by dwelling on it for one second. During the one second period, 85% of gaze locations had to be inside the button. As the user dwelled on the button, it gradually changed from gray to red. When successfully activated, the dwell button flashed blue and played an audible sound. After a "click", the dwell button was deactivated until the user looked away from the button.

  In phase one of the gaze tracker sessions, the dwell button was in the lower right-corner. I found this made it more difficult for users to use Dasher because extreme gaze directions were sometimes confused with dwelling on the button. In phase 2, the button was moved to the lower-left corner to make it easier for users to reverse without accidentally activating the button.

- **Audio feedback of words** – After writing a new word in Dasher, the word was played back to the user using text-to-speech. This provided users with feedback about what had just been written without requiring the user to divert their gaze away from navigating to the next word.

- **Slow-down region** – I added a feature that allowed navigation to be progressively slowed by dwelling in a circle at the center of the navigation display (figure 4.12). If a user looked inside the circle, navigation speed was progressively dampened. After 1.25 seconds, navigation stopped completely. If a user looked outside the circle, zooming speed was progressively increased. This slow-down region allowed the user to stop temporarily in order to rest, look at the target sentence, look at what had been written, or move to the dwell button. Feedback about the current level of speed dampening was provided by filling the circle with a semi-transparent red color. The fill became progressively more red as speed was dampened. If the user looked off the top or bottom of the Dasher navigation area, the speed dampening was accelerated, stopping navigation completely after only 0.3 seconds.

  In phase one of the gaze tracker sessions, the slow-down region was 100 pixels in diameter. In phase 2, the slow-down region was increased in size to 150 pixels. The larger size made it easier to activate using the unsmoothed mouse settings used in phase 2. In addition, in phase 2 the slow-down region was not activated if the user looked off the top or bottom of the Dasher display area in the right 50% of the screen. This made it easier for users to reverse using Dasher's eye tracker mode.

4.6.6 Target Sentences

The target sentences were newswire sentences from the set-aside directory of the CSR-III corpus. I chose sentences with between 8 and 12 words, removing all punctuation except for apostrophes, and making each sentence lower case. Using the trigram language model used for recognition, the average per-word perplexity was 97. Using the PPM language model used in Dasher, the information content was 1.9 bits per symbol. The sentences had an out-of-vocabulary rate of 0.7% using the 64K vocabulary. The target sentences were excluded from the data

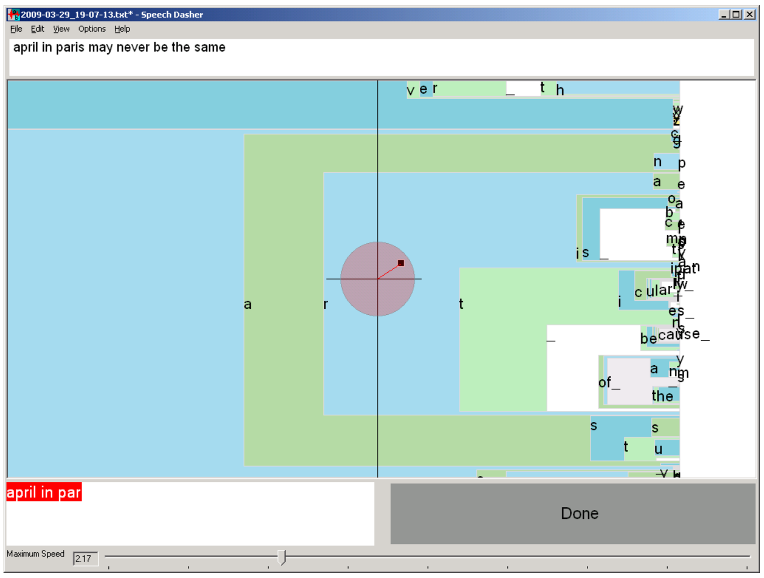

2009-03-29_19-07-13.txt* - Speech Dasher
File Edit View Options Help
april in paris may never be the same
april in par
Done
Maximum Speed
2.17

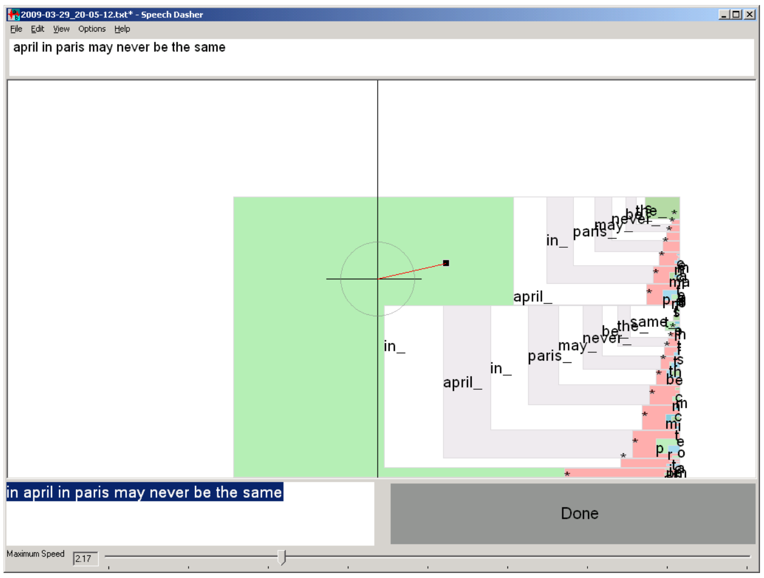
Figure 4.12: The user after writing “april in par” in the Dasher condition. As letters are completed, they are added to the text box in the lower left. By dwelling inside the red circle in the center, navigation could be temporarily stopped.

used to train the PPM and recognition language models. Participants received the target sentences in random order.

4.6.7 Method

In the Dasher condition, after presentation of the target sentence, dwelling on a CORRECT button caused the main navigation display to appear and zooming to automatically begin. As the user wrote, letters were output in the text box in the lower left (figure 4.12). Once the user had completed the sentence, dwelling on the DONE button moved to the next sentence.

In the Speech Dasher condition, after presentation of the target sentence, dwelling on a MIC ON button turned on the microphone. After speaking the sentence, dwelling on a MIC OFF button turned off the microphone. I allowed only a single attempt at recognition for a particular target sentence. If a participant misspoke, correction had to be done via navigation in the main display.


2009-03-29_20-05-12.txt* - Speech Dasher

april in paris may never be the same

in april in paris may never be the same

Figure 4.13: User at the start of navigation in the Speech Dasher condition. The best recognition hypothesis is "in april in paris may never be the same". The recognition result includes an insertion error of the word "in". By navigating to the top set of boxes in the main display, the correct sentence can be written.

After a short recognition delay (2.0 s ± 1.1 s), a beep signaled recognition was complete and the navigation display would appear. The best recognition hypothesis was initially displayed in the text box in the lower left (figure 4.13). If the recognition was completely correct, the user could dwell on the DONE button to move immediately to the next sentence. Otherwise, the user could begin navigation in the main display and the best hypothesis would be replaced by whatever the user wrote in Dasher. Once the user had completed the sentence, dwelling on the DONE button moved to the next sentence.

In both conditions, the participant was told to proceed "quickly and accurately". The participant could adjust the speed of zooming via a slider at the bottom of the window. The speed slider was controlled using a conventional mouse and could be adjusted between sentence tasks as the participant saw fit. I encouraged each participant to increase the speed setting as he or she progressed through the study.

4.6.7.1 Collection Errata

In participant DE1's second session, the Dasher condition was accidentally omitted. In participant US1's final gaze tracking session, Dasher's eye tracker mode was mistakenly turned off. This session was replaced with a new session conducted the next day.

4.6.8 Overall Results

In this section, I provide summary statistics and plots for all the gaze tracking sessions (both phase 1 and phase 2).

4.6.8.1 Error Rate

The *recognition error rate* is the error rate of the speech recognizer's best hypothesis (applies only to the Speech Dasher condition). Recognition error rate was measured using the word error rate (WER). Over all gaze tracking sessions, the participant average WER was 22% and recognition took 1.4 $\times$ real-time. Figure 4.14 shows the WER for each participant. The three participants saw different levels of recognition errors. US1 had the best recognition of 10%, UK1 had a WER of 15%, and DE1 a very high WER of 42%.

The *user error rate* is the error rate of the final text written by the participants. I measured the user error rate using both word error rate (WER) and the character error rate (CER). The CER is the character edit distance between the target sentence and what the user wrote divided by the number of letters in the target sentence. In both conditions, participants wrote the target sentences with very few errors (figure 4.15 and 4.16). Over all gaze tracking sessions, the user error rate was 1.3% WER (0.59% CER) in Dasher and 1.7% WER (0.70% CER) in Speech Dasher.

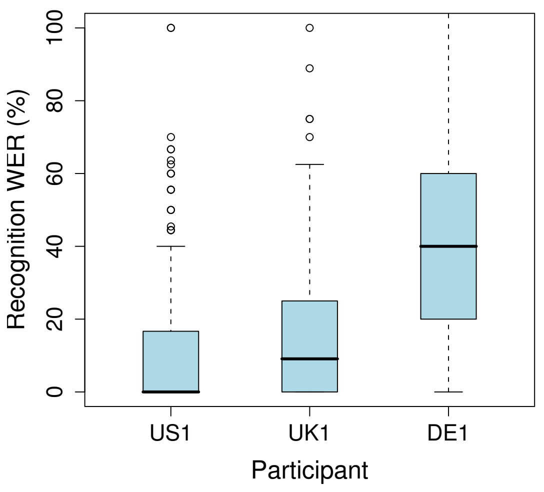
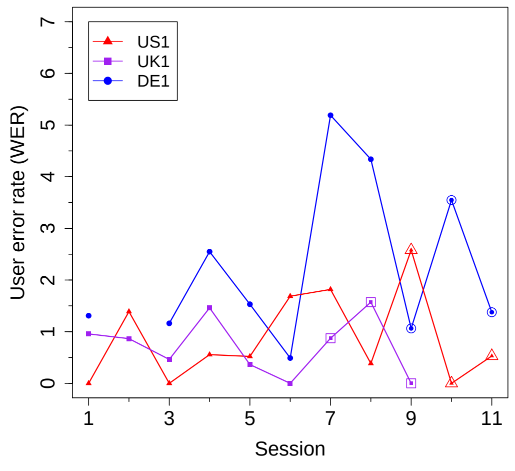


Figure 4.14: Recognition word error rate on sentences spoke by each participant during the gaze tracking sessions.

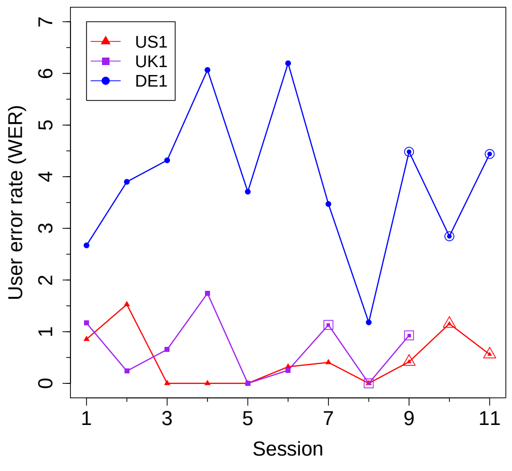
4.6.8.2 Entry Rate

The *entry rate* was calculated in words per minute (wpm). I used the standard convention defining a "word" as five consecutive characters. In the Dasher condition, I used the time interval between the end of the dwell on the CORRECT button and the start of the dwell on the DONE button. In the Speech Dasher condition, I used the time interval between the end of the dwell on the MIC ON button and the start of the dwell on the DONE button. Note that I excluded the overhead associated with the dwell button delay in both conditions. The entry rates included correction time, audio recording, and recognition delays.

I also report the *information rate*. This measures how much information content was in the text written by the participant. Uncommon words have a higher information content. Information rate was measured in bits per second (bps) and was calculated using the PPM language model used in the study. The information rate allows easier comparison between the results presented here and any future experiments using other language model training material, languages, etc.

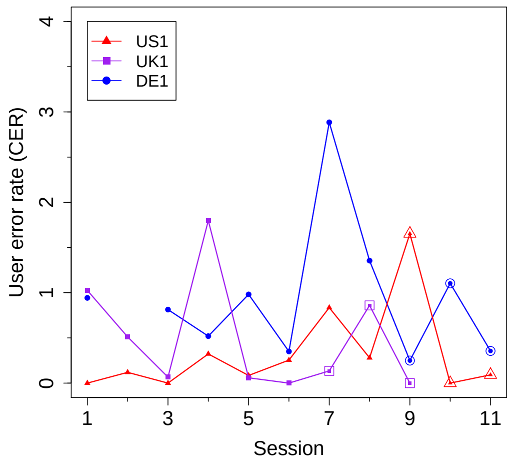


(a) Dasher

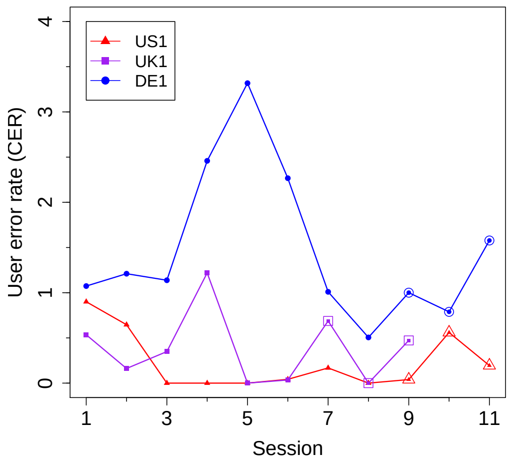


(b) Speech Dasher

Figure 4.15: User error rate (measured in word error rate) during all gaze tracking sessions in the study. The phase 2 sessions are shown with a double symbol.



(a) Dasher



(b) Speech Dasher

Figure 4.16: User error rate (measured in character error rate) during all gaze tracking sessions in the study. The phase 2 sessions are shown with a double symbol.

In each 15-minute session, participants wrote on average 20 sentences in Dasher and 33 sentences in Speech Dasher. Over all gaze tracking sessions, participants' entry rate was 16 wpm (2.5 bps) in Dasher and 37 wpm (5.6 bps) in Speech Dasher. In the Speech Dasher condition, 40% of sentences were recognized completely correctly. These sentences obviously had a very fast entry rate. Removing sentences recognized completely correctly, participants still had an entry rate of 26 wpm (4.0 bps).

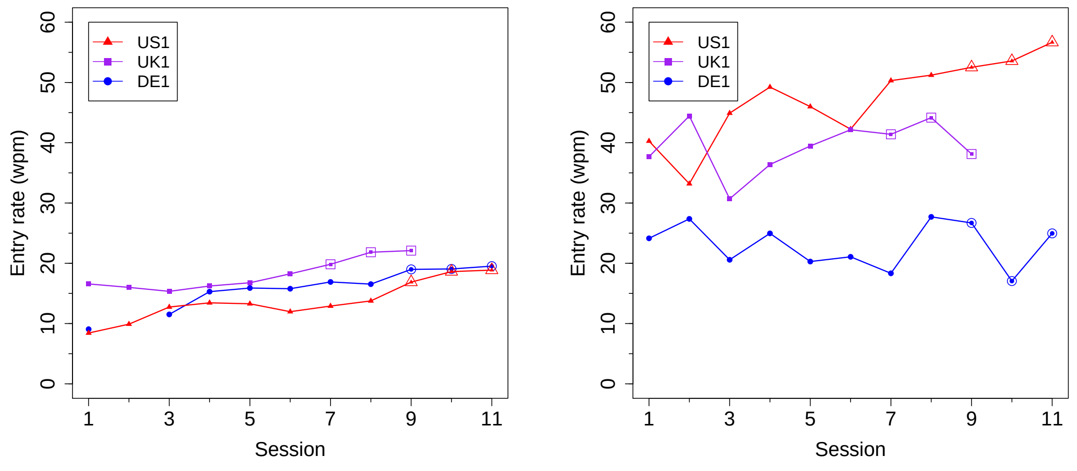
Participants became faster in both conditions as the study progressed (figure 4.17 and 4.18). Participants' entry rate in Speech Dasher was much more variable between sessions. This was presumably due to the large effect that recognition accuracy had on entry rate.

4.6.8.3 Influence of WER on Performance

I also wanted to see how performance degraded at different recognition word error rates. Figure 4.19 shows the recognition WER and entry rate for each of the participants' sentences. As expected, the lower the recognition WER, the faster participants wrote. The Speech Dasher interface performs well even for fairly high recognition word error rates. Participants' average entry speed in the Dasher condition was 16 wpm. Given the fit line in figure 4.19, participants would write faster using Speech Dasher as long as WER was below 50%.

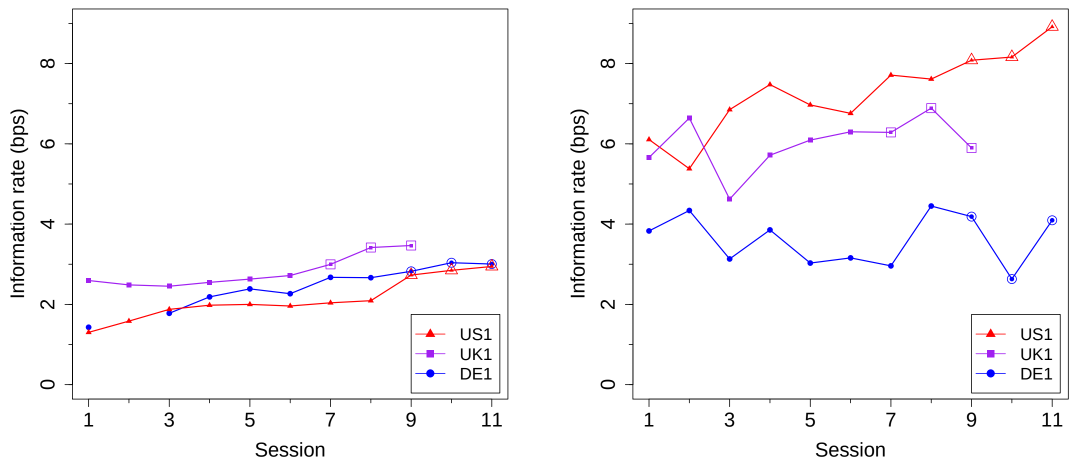
4.6.8.4 Speed Setting

Participants were allowed to set Dasher's zooming speed. As shown in figure 4.20, participants increased their setting in both conditions as the study progressed. All participants eventually increased their speed setting to 4 bits per second or faster in both conditions. In the Dasher condition, on sentences in which participants used higher speed settings, they tended to write faster (figure 4.21).



(a) Dasher (b) Speech Dasher

Figure 4.17: Entry rate (measured in words per minute) for each participant during all gaze tracking sessions. The phase 2 sessions are shown with a double symbol.



(a) Dasher (b) Speech Dasher

Figure 4.18: Information rate (measured in bits per second) for each participant during the gaze tracking sessions. The phase 2 sessions are shown with a double symbol. The Speech Dasher information rate (right) was the information rate under the PPM language model.

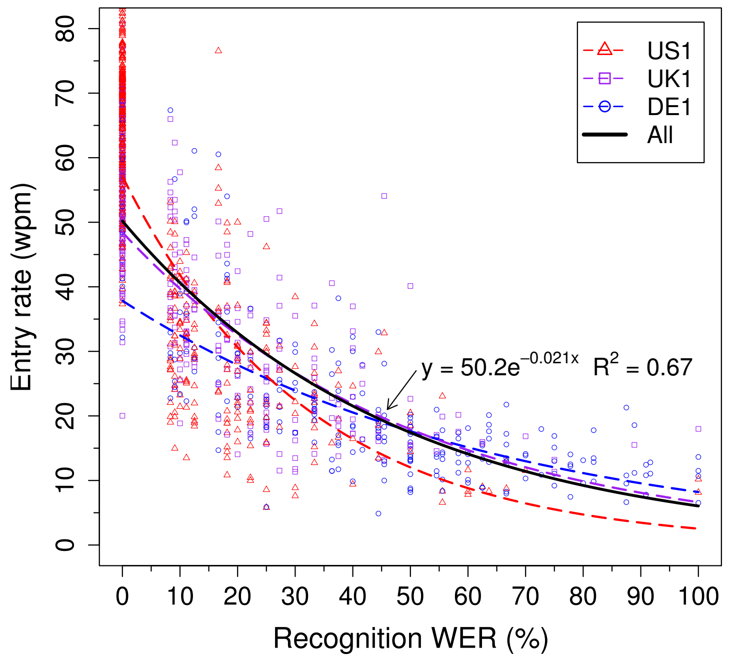
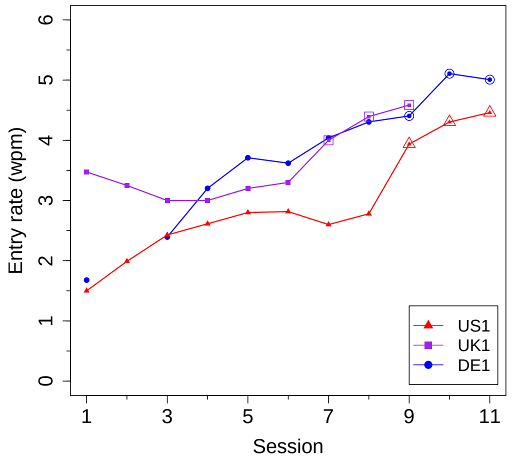


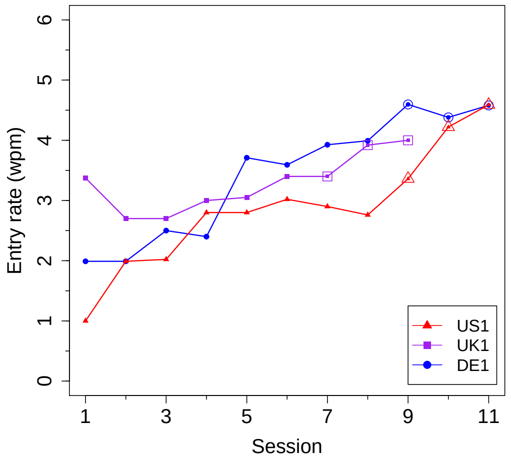
Figure 4.19: Participants' sentences by entry rate and recognition WER. The solid black line was fit to the data from all participants. The dotted colored lines were fit to each participant's data.

4.6.8.5 Gaze Location

During the experiment, I logged the users' gaze location about 30 times per second. Using this data, I calculated where participants were looking in each condition (figure 4.22). The right side and center of the Dasher navigation display were the most popular gaze locations. The main difference was that users in Speech Dasher spent more time looking at the text box in the lower left. They also spent more time looking at the target sentence in the top text box. This was probably due to users comparing the recognition result with the target sentence to determine if the result was completely correct.



(a) Dasher



(b) Speech Dasher

Figure 4.20: The speed setting (in bits per second) chosen by participants in each session. The phase 2 sessions are shown with a double symbol.

4.6.9 Expert Performance

Here I provide analysis of the final three gaze tracking sessions (phase 2 only). These sessions used the final gaze tracking setting and improved interface layout. In addition, by this point in the study, participants were experts at both interfaces.

4.6.9.1 Expert Error Rate

In total, participants wrote 324 sentences in the last three Speech Dasher sessions. The average participant recognition error rate was 22%. The WER varied significantly between participants with a WER of 7.8% for US1, 12.4% for UK1, and 46.7% for DE1.

Figure 4.23 shows the user error rates of participants in the last three gaze tracking sessions. User error rates were low overall, with an average CER of 0.5% in Dasher and 0.6% in Speech Dasher.

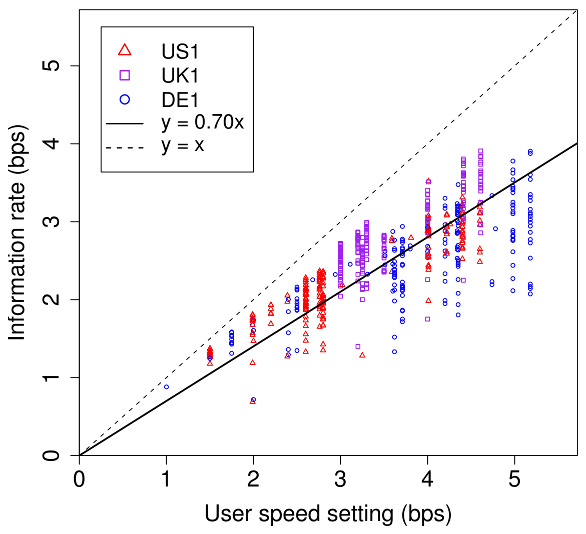


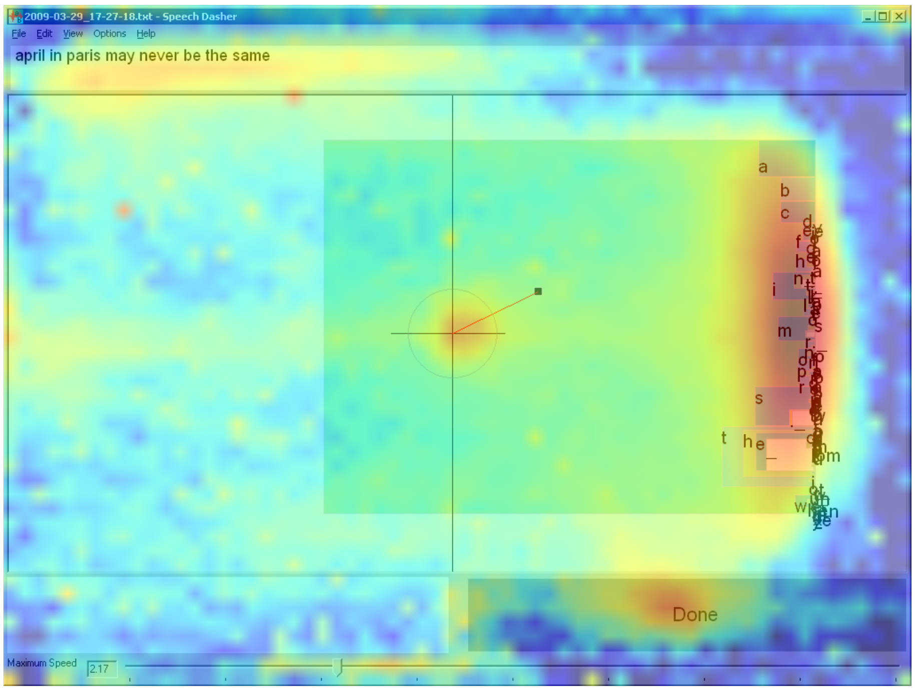
Figure 4.21: Sentences plotted by user's speed setting and the information rate of the user's writing in the Dasher condition. A point on the diagonal line $y = x$ would occur if a user was zooming at maximum speed without reversing or slowing down, and there was no overhead to starting or stopping.

4.6.9.2 Expert Entry Rate

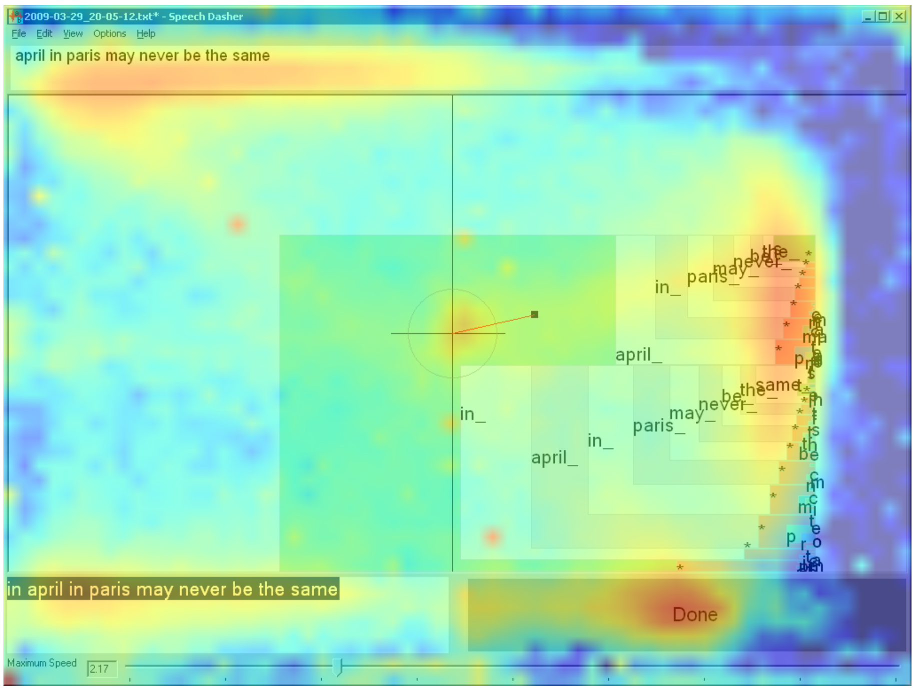
Figure 4.24 shows the entry rates of participants in the last three gaze tracking sessions. The average participant entry rate was 20 wpm in Dasher and 40 wpm in Speech Dasher. Compared to Dasher, participants wrote twice as fast using Speech Dasher. In Speech Dasher, participants showed a wide range of entry rates, presumably due to their differing recognition error rates. US1 was the fastest at 54 wpm, UK1 was next at 42 wpm, and DE1 was at 23 wpm. On sentences with at least one recognition error, participants still wrote at 30 wpm in Speech Dasher (figure 4.24c).

4.6.9.3 Performance at High WER

Participant DE1 had the slowest Speech Dasher entry rate in figure 4.23b. His slow entry rate was to be expected considering his high WER of 47%. I analyzed

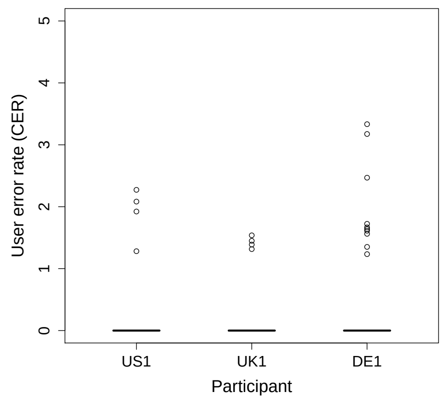

2009-03-29_17-27-18.txt - Speech Dasher
File Edit View Options Help
april in paris may never be the same
Done
Maximum Speed
2.17

(a) Dasher

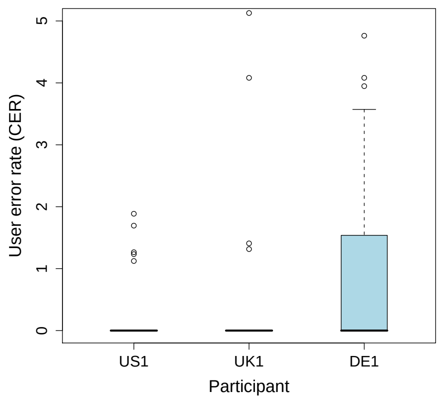

2009-03-29_20-05-12.txt* - Speech Dasher
File Edit View Options Help
april in paris may never be the same
april_
in_
paris_
may_
never_
the
same
in april in paris may never be the same
Done
Maximum Speed
2.17

(b) Speech Dasher

Figure 4.22: Heat map showing how frequently participants were looking at various parts of the interface in each condition. The blue areas were the least looked at areas and the red areas were the most looked at areas.



(a) Dasher



(b) Speech Dasher

Figure 4.23: The user error rate of participants in the last three gaze tracking sessions. Error rate was measured using the character error rate (CER).

his data separately to see if he was seeing any benefit from speech. Table 4.2 shows the average writing speed of DE1. Even with his high recognition error rate, DE1 still got a small benefit from Speech Dasher, writing 20% faster. This demonstrates that Speech Dasher degrades gracefully in the face of inaccurate recognition.

4.6.9.4 Gaze Tracking versus Mouse

Table 4.3 shows the average entry rates of participants in their last gaze tracking session compared to their final mouse session. In Dasher, participants were actually slower using the mouse than using gaze tracking. In Speech Dasher, participants performed about the same using the mouse or using gaze tracking. There could be several reasons for participants' slower mouse performance in Dasher. The participants may have required more than one session to become accustomed to mouse control. In addition, the mouse speed setting may have needed to be faster for optimal Dasher use. Nonetheless, it is encouraging that Dasher with gaze tracking was competitive with an input device that users had much more experience with.

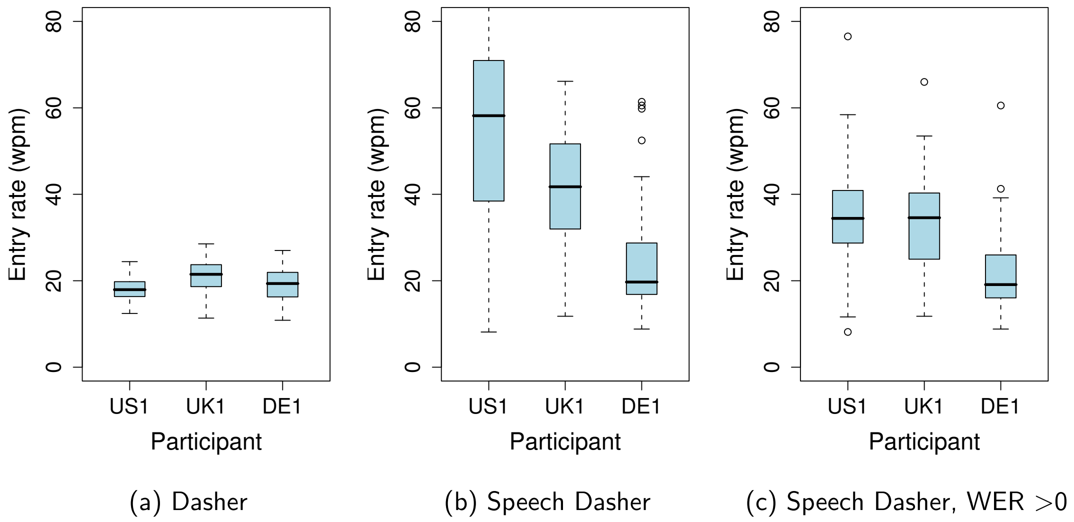


Figure 4.24: The entry rate of participants in the last three gaze tracking sessions. The rightmost graph (c) shows performance on sentences with recognition errors.

4.7 Discussion

In this section, I discuss some of the limitations of my study. I also discuss practical improvements to Speech Dasher based on lessons learned during the study.

4.7.1 Limitations

4.7.1.1 Number of Participants

My user study used three participants. But even with this small number of participants, the difference in speed between Speech Dasher and Dasher was clear.

Condition	Subset	Entry rate wpm ± 95% CI
Dasher	-	19.2 ± 0.9
Speech Dasher	-	23.4 ± 2.9
Speech Dasher	WER > 0	21.2 ± 2.2

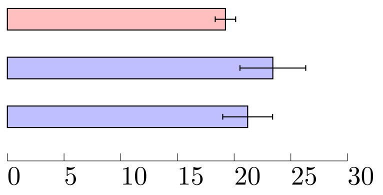


Table 4.2: The entry rate of participant DE1 in the last three gaze tracking sessions. The third row shows performance on sentences with recognition errors. The 95% confidence intervals show the individual's variance.

Condition	Input	Entry rate wpm ± 95% CI
Dasher	gaze	20.2 ± 1.9
Dasher	mouse	16.8 ± 0.5
Speech Dasher	gaze	39.9 ± 18.0
Speech Dasher	mouse	41.4 ± 12.2

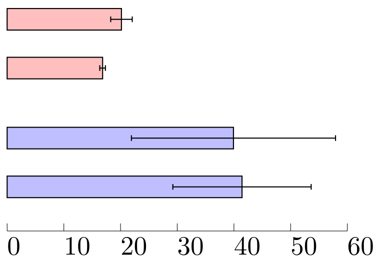


Table 4.3: The entry rate of all three participants in their final gaze and final mouse sessions. The 95% confidence intervals shows variance between individuals.

4.7.1.2 Pointing Device

I tested users using a gaze tracker to control Dasher. Speech Dasher may also work well with other more conventional pointing devices. In particular, participants in the user study could not take advantage of Speech Dasher features such as selective correction and spoken corrections.

4.7.1.3 Comparison with Speech Correction

I did not test a correction interface that used speech as the primary correction mechanism. Such an interface could be a competitive option for people writing via speech and gaze tracking alone. In my own testing with Nuance Dragon NaturallySpeaking v9, I found that while speech-only corrections were sometimes successful, often the mouse was required. The interface elements in Dragon were too small for practical use with a gaze tracker. It would be interesting to investigate a correction interface using primarily speech-based corrections but where non-speech interface elements were optimized for gaze tracking use.

4.7.1.4 Transcription Task

In the study, users wrote transcribed sentences I presented. This made it easy to measure the correctness of their writing. But transcribing newswire text is not a normal user activity. Users may have had more trouble remembering the sentence then if they had composed their own sentence. While users could replay the target sentence via TTS, sometimes it was not possible to guess a word's spelling from the TTS audio. In such cases, navigation had to be interrupted to consult the target sentence. It would be interesting to test Dasher and Speech Dasher in a more realistic composition setting.

4.7.2 Design Improvements

4.7.2.1 Eliminate Shared Boxes

Participants sometimes found they navigated into a word that looked correct, only to discover later that the word's ending was wrong. This was a problem when several competing hypotheses shared an initial prefix. This caused a split in the Dasher box hierarchy, which made the entire word difficult or impossible to see before navigating into the prefix. For example, in figure 4.25, the user wants to write "all administrations" but it is difficult to currently tell if that option is inside the "admin" box or not. Another problem with shared prefixes is that the

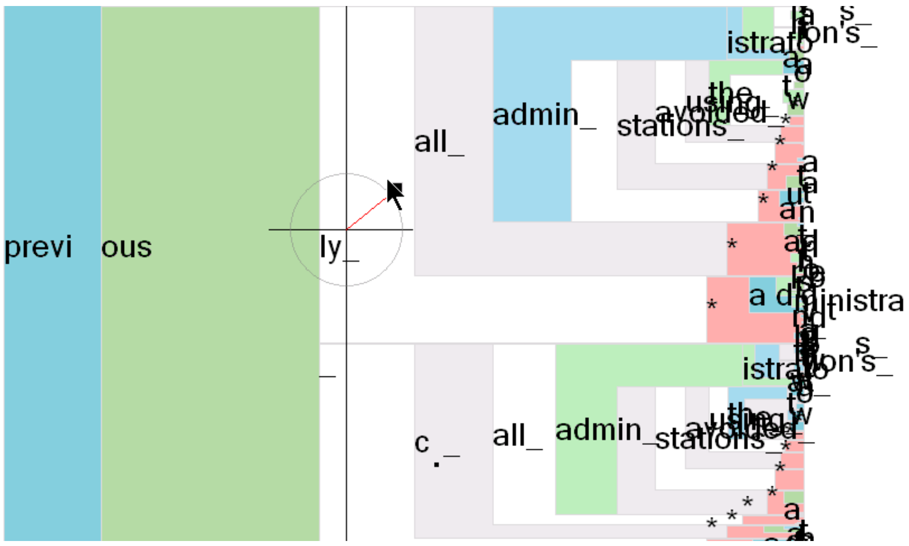


Figure 4.25: After writing the word "previously", the user wants to write "all administrations". It is difficult for the user to tell if the word "administrations" is present inside the "admin" box.

letters of the word get split over a wide horizontal space and are not necessarily aligned vertically. This makes comprehension of the complete word difficult. A solution would be to predict full words in the non-escape area and not to allow words with the same prefix to share an initial box.

4.7.2.2 Correspondence with Best Hypothesis

Speech Dasher displayed the best recognition hypothesis in the lower left text box. This best hypothesis was the most probable path through the recognition word lattice. This usually, but not always, corresponds to the words displayed as primary predictions using Speech Dasher's probability model. The primary prediction could be different if, for example, the best lattice path's first word had a low probability compared to other paths. This low probability word would get put in the escape box and not with the primary predictions. The probability model should be changed to prevent this mismatch.

4.7.2.3 Layout of Words

The letters in the Dasher display were aligned so letters in the same word appeared on the same vertical baseline. But the horizontal width of words sometimes caused them to visually overlap and obscure each another (e.g. “avoid” and “using” in the top-right of figure 4.25). In many cases, this could be avoided by intelligently offsetting words in close proximity.

4.7.2.4 More Responsive Model

The current probability model relies on tracking a large number of paths within the recognition lattice. When the user diverges from the lattice, the model expends considerable computational effort trying to calculate where the user might reenter the lattice. Despite optimization efforts, this occasionally caused noticeable performance lags. Using a more compact representation for the recognition result such as a word confusion network [104] might help alleviate this problem.

4.7.2.5 Variable Zooming Speed

Speech Dasher currently uses a single user-selected zooming speed. This may be suboptimal. When navigating through a recognition hypothesis well predicted by the lattice, even a slow zooming speed results in fast writing. In fact, slow zooming is needed in order to read the words that are being presented in close visual proximity. On the other hand, when navigating outside the lattice, the letter-based predictions are typically much less strongly predicted and are more visually separated. In this case, a faster zooming speed is needed for efficient writing. Speech Dasher might benefit from different zooming speeds depending on whether the user is writing something well predicted or not.

4.8 Related Work

There has been only limited work in interfaces that combine speech and some sort of continuous pointing or gesture interface. Huggins-Daines and Rudnicky [75] developed a touch-screen interface that allowed user to interact with the alternative hypotheses within a recognition lattice. In their interface, users could "pull apart" sections of the best recognition result to reveal other alternative hypotheses in a given time region of the lattice. No user-trial results were reported.

In Kurihara et al. [91], an instructor's handwriting was combined with speech recognition to provide a confusion network correction interface. Their interface displayed a static view of the hypothesis space compared to Speech Dasher's dynamic zooming display. But similar to Speech Dasher, users could use a continuous gesture to select between the alternative hypotheses. Their system resulted in instructors needing fewer pen strokes to correct their text.

Akman [9] took the Speech Dasher idea and applied it to transcribing audio files in Turkish. Akman developed a probability model for use in Dasher that was based on the weighted finite state automata output of a Turkish research recognizer. As in the Speech Dasher's model, Akman's model supports various strategies for handling cases when what the user wanted was not in the speech result. On two Turkish acoustic test sets, he compared his methods with my original Speech Dasher model (based on n-best lists). His best model obtained lower cross entropies of 0.26 and 0.54 bits per symbol compared to the Speech Dasher model which obtained 0.45 and 0.60 bits per symbol. This shows the advantage of using the probability information in the recognition lattice. No user-trial results were reported.

While my focus here was not to improve normal Dasher's gaze writing speed, I highlight several studies that show the performance in my study was consistent with past Dasher gaze tracking studies. I caution against drawing strong conclusions from comparing the entry rates between these Dasher experiments. The entry rate (in wpm) in Dasher is related to how well the text is predicted by the language model. This can vary depending on the language, training text, and

PPM settings. In addition, differences in participants and experimental method make comparison between studies tenuous.

In Ward et al. [158], 2 experts and 2 novice users wrote text using Dasher and a gaze tracker. After an hour of practice, users wrote at 16–26 wpm (estimated from figure 1b in [158]). After 3-8 hours of practice, my users wrote at 18–22 wpm. The results in [158] are a superset of those appearing in Ward's thesis [156]. In [156], the sentences used in the study had an information content of 1.7 bits per symbol. In my study, sentences had a slightly higher information content of 1.9 bits per symbol.

Tuisku et al. [144] did the largest Dasher gaze tracking study to date. They had 12 participants transcribe Finnish text using Dasher in a series of ten 15-minute sessions. By the last session, participants wrote at 17 wpm. They showed almost linear increases in participants entry rate over 1.5 hours of use. In my study, I found users entry rate gains started to diminish by their final sessions (after 3–8 hours of practice). It is possible their participants had not yet reached their entry rate limit. Tuisku et al. had also allowed the PPM model to adapt during the study. Since their training text was a Finnish novel and their test set was a corpus of short phrases, I expect part of the entry rate increase reflects adaptation of the language model as the study progressed. They did not report the information content of their test phrases.

There have been numerous studies investigating correcting speech recognition with traditional input devices such as keyboard, mouse and stylus. In Karat et al. [80], 12 novice and 4 expert users dictated text from a book. They were allowed to correct errors via speech, mouse or keyboard. Novice users' entry rate was 14 wpm while experts' entry rate was 25 wpm. They do not report the recognition error rate.

Shuhm et al. [139] used handwriting to correct recognition errors. Novices had an entry rate of 5 wpm using pen correction and 7 wpm using mouse and keyboard correction. In their study, the recognition WER was 25%.

In Horstmann [86], 24 expert speech recognition users with physical disabilities were given text transcription and composition tasks. The majority of the users relied on the keyboard for correction (17 out of 24). The rest used speech or an on screen keyboard to correct recognition errors. The users entry rate was 17 wpm at a recognition WER of 15%.

Larson and Mowatt [93] had 12 users dictate text from a book. Users corrected errors using speech, speech and a mouse, an alternates list, or a soft keyboard. Using the reported average correction times and number of words per task, I calculated the entry rate for each entry method: 5 wpm for speech, 13 wpm for speech and mouse, 13 wpm for alternates list, and 17 wpm for soft keyboard. They do not report the recognition error rate. In a second experiment, 6 users were allowed to use any of the different correction methods. In this experiment, users entry rate was 17 wpm at a WER of 13% (estimated from the reported average words per task and total recognition errors).

4.9 Conclusions

Speech Dasher is a novel interface that allows users to correct recognition errors by navigating through a speech recognizer's many alternative hypotheses. Using Speech Dasher, users were able to correct many recognition errors with only minimal effort. When the recognizer's search space didn't contain the right answer, Speech Dasher allowed seamless fallback to efficient letter-by-letter predictive text entry.

I showed that Speech Dasher could be a useful and efficient entry method for people using only speech and their eyes. After four hours of practice, users were able to write at 40 wpm despite a recognition WER of 22%. Even on sentences that had a least one recognition error, users were still able to write at 30 wpm. Using Dasher without speech, users were significantly slower, writing at 20 wpm. This shows that Speech Dasher successfully leveraged recognition information to greatly improve users' writing efficiency.

Chapter 5

Touch-Screen Mobile Correction

5.1 Overview

Using speech recognition to enter text on a mobile device is an attractive alternative to conventional entry methods. This is because mobile devices typically lack a keyboard. Without a keyboard, the device must rely on techniques such as predictive text, multi-tap, isolated character recognition, or handwriting recognition. For an overview of mobile input methods, see [102; 173]. But key- and stylus-based input methods all fall far short of the speed at which people can speak. For example, users can dictate to a computer at 102 (uncorrected) words per minute [93]. Besides its inherent input speed, speech is also a naturally acquired skill that requires little practice from users.

However, speech recognition remains to prove itself as a competitive mobile text entry method. There are a number of problems with using speech for mobile text entry. The most serious problem is how to deal with recognition errors. Mobile speech recognition is likely to experience even more errors than desktop speech recognition due to limited processing power, poor quality microphones, and background noise. Just as on the desktop, mobile speech recognition performance is also bound to suffer due to the process of correcting recognition errors [80]. While in [93] users dictated at 102 words per minute (wpm), after correction

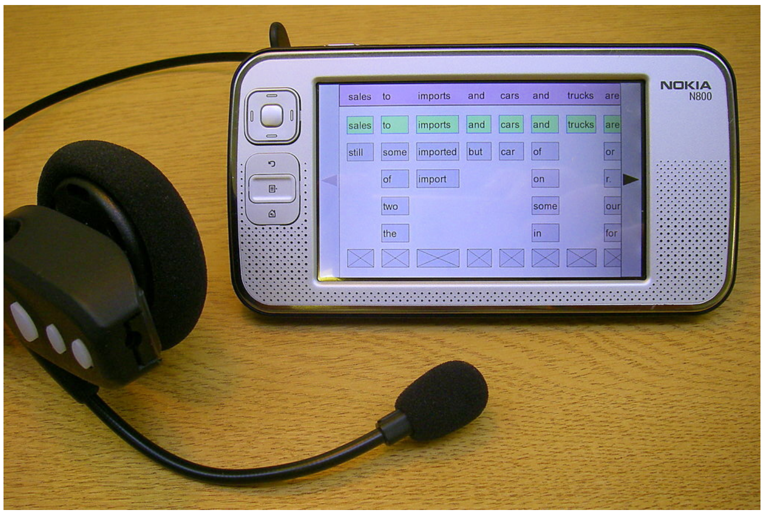

NOKIA
N800
sales to imports and cars and trucks are
sales to imports and cars and trucks are
still some imported but car of or
of import on r
two some our
the in for

Figure 5.1: The Nokia N800 device and a Bluetooth headset. The Parakeet continuous speech recognition user interface is shown running on the device.

the actual writing throughput was only 10 wpm.

Another problem is the obvious privacy implications of using speech in a mobile and possibly public setting. Finally, it has been argued that speech input makes demands on a human's memory and processing capabilities which could adversely affect task completion [130]. Despite these caveats, the potential speed and naturalness of mobile speech text entry make it an entry method worth exploring.

Recently, researchers have made progress in developing continuous speech recognition engines for embedded and mobile devices [33; 110]. Projects have been undertaken to enable mobile use of research recognizers such as Sphinx [74], SUMMIT [67] and Janus [87]. These developments help pave the way for my exploration.

In this chapter, I describe Parakeet: a touch-screen interface I designed for efficient mobile text entry using speech. Users enter text into their mobile Linux device (such as the Nokia N800) by speaking into a Bluetooth headset microphone (figure 5.1). The user then reviews and corrects the recognizer's output using a touch-screen interface.

The development of Parakeet followed several design cycles. My design was guided to a large extent by computational experiments on recorded speech data. These experiments helped from two perspectives. First, they helped guide my decisions about design choices. For example, they helped me decide which edit operations were most useful to include in the correction interface. Second, they helped me find the optimal parameter settings for my user interface. For example, I used experimental results to determine how many word alternatives to display.

The rest of this chapter is structured as follows. First, I discuss the principles that guided my design. Second, I describe my interface design and detail the experiments that helped shape that design. Third, I describe the details of my mobile speech recognition system. Fourth, I present a user study investigating how well the new interface worked in practice. Fifth, I present results from an expert pilot study demonstrating the promising potential speech has as a mobile text entry solution. Finally, I discuss limitations and implications of my findings and then conclude.

5.1.1 Publication Note

The work presented in this chapter was joint work with Per Ola Kristensson. The information contained in this chapter was published in part in the papers "Parakeet: A Touch-Screen Interface for Continuous Speech Recognition on Mobile Devices" [152] and "Parakeet: A Demonstration of Speech Recognition on a Mobile Touch-Screen Device" [153] at the International Conference on Intelligent User Interfaces (IUI'2009). This research was funded in part by Nokia.

5.2 Design Principles

Parakeet's development was guided by five key design principles. In this section, I describe these principles and how they relate to prior work.

5.2.1 Avoid Cascading Errors

Speech recognition is imperfect and recognition errors are ultimately unavoidable. Thus, error correction is a crucial part of any speech recognition interface. As previous research has shown (e.g. Karat et al. [80]), users have difficultly correcting speech recognition errors using only speech. This is partly because errors may cascade – recognition errors in the correction phase may require further corrections, and so on. A possible solution is to use a different modality than speech for correction. For example, Suhm et al. [139], investigated correcting speech using pen gestures. For an in-depth review of multimodal speech interfaces, see Oviatt et al. [116].

To avoid cascading errors, I also decided to create a multimodal speech interface. In my interface, recognition is first performed on the user's utterance. The recognition result is then corrected using a method that is simple, direct and transparent to the user. This correction method does not rely on any further use of error-prone recognition technologies.

5.2.2 Visualize the Hypothesis Space

In a typical speech recognition interface such as Nuance Dragon NaturallySpeaking, only the best recognition hypothesis is presented to the user. In order to explore alternatives to this best hypothesis, the user must take explicit action. For example, the user might highlight some text and issue a command to bring up a list of alternative words. As observed by Kristensson and Zhai [89], this style of error correction interface introduces a degree of uncertainty. The users has to hope his or her action will expose the desired correction. If it does not, the user has wasted effort. Such interfaces may lead to user frustration, which is a common problem in intelligent user interfaces in general.

In designing my interface, I wanted to avoid forcing users to blindly search the hypothesis space. I wanted the user to be able to see immediately, and without explicit action, whether the desired correction was available. If the desired text was not easily available, a more costly corrective measure could be used that was guaranteed to succeed. For example, rather than providing a long list of word alternatives that requires scrolling, I would rather present a small number of alternatives. If the alternatives do not contain the correct word, the user can invoke a software keyboard that is guaranteed to allow entry of the desired word.

5.2.3 Usable by Touch

I designed Parakeet for touch-screens for three reasons. First, touch-screens do not require a stylus and are hence easier for users to interact with while on the go. Second, one-handed usage is impossible with a stylus. There is evidence that indicates users prefer mobile interfaces that can be operated with one hand [82]. Third, a well-designed touch-screen interface can also be used with a stylus, but the converse does not necessarily hold. By creating a good touch-screen user interface, I am also creating an interface suitable for stylus use.

5.2.4 Support Fragmented Interaction

I wanted my interface to be usable while walking around. In such a mobile setting, users would need to divide their attention between interacting with the device and dealing with their surroundings. Oulasvirta et al. [114] found that users interacting with a mobile web browser while walking attended to the device in 4 to 8 second bursts. This finding has two implications. First, my interface needs to enable users to quickly process and respond. Second, my interface needs to be designed so that it is easy for users to pick up from where they left off after an interruption.

Therefore, I designed Parakeet to minimize attention demands on the user. For example, after recognition is completed, Parakeet flashes the entire display screen and plays a short beep. This simple feedback frees the user to attend to their surrounding almost entirely while waiting for recognition to complete.

5.2.5 Minimize Physical Actions

I designed Parakeet to help minimize the physical actions required by the user. In a mobile setting, a large sequence of precise touch gestures is likely to go wrong. I designed towards an interface that presents more visual information rather than less. This may require several bursts of visual attention from the user, but hopefully will require fewer motor actions. For example, a user might enter a few letters of a word and then scan a list of predictions which allow completion of their word with a single further physical action.

5.3 Interface Description

The main screen of Parakeet displays the recognizer's best hypothesis along a single line at the top (figure 5.2). If the best hypothesis cannot fit on the screen, the user can scroll using the buttons on the left and right sides. In addition to the best hypothesis, likely alternatives for each word in the best hypothesis are displayed. For example, in figure 5.2 the word "worked" has three other competing words ("work", "were", and "we're"). The currently selected word in each column is highlighted in green.

This display is based upon a word confusion network [65]. A word confusion network is a time-ordered set of clusters where each cluster contains competing word hypotheses along with their posterior probabilities. The word confusion network is built from the lattice generated during the speech recognizer's search.

By displaying more than a 1-best result, Parakeet allows the user to quickly scan other likely word alternatives. This is done without requiring explicit user action (as in standard interfaces such as Dragon).

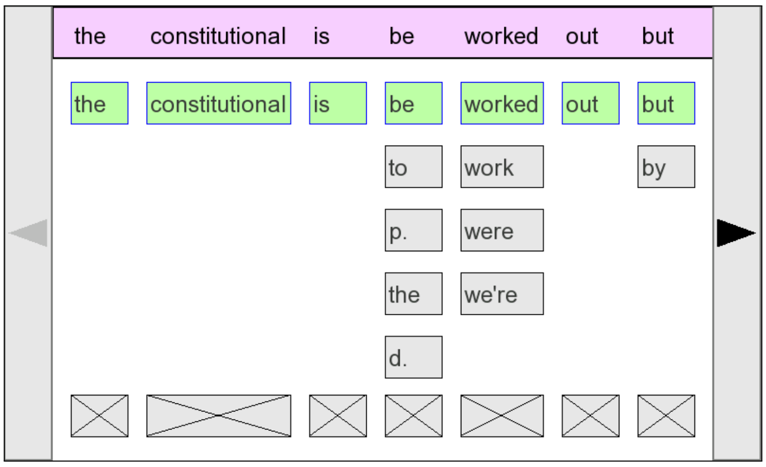


Figure 5.2: The word confusion network correction interface. The user can touch the word buttons to choose alternative words. The "X" button at the bottom allows words to be deleted.

5.3.1 Available User Actions

5.3.1.1 Substituting Words

To substitute a word, the user can use several methods. The most direct method is to simply touch an alternative word in the confusion network. This causes the selected word to change color and updates the word displayed in the top row. Sometimes several desired substitutions are in adjacent columns. In this case, the user can slide his or her finger across each desired word to perform multiple substitutions with one gesture.

5.3.1.2 Editing Words

The user's desired word may not be one of the displayed alternatives. By touching a word in the top row or by double-tapping any word, the user is brought to a separate edit screen. Here they can use a predictive software keyboard to either edit the selected word or enter an entirely new word (figure 5.3). The software keyboard will be described in detail in section 5.4.

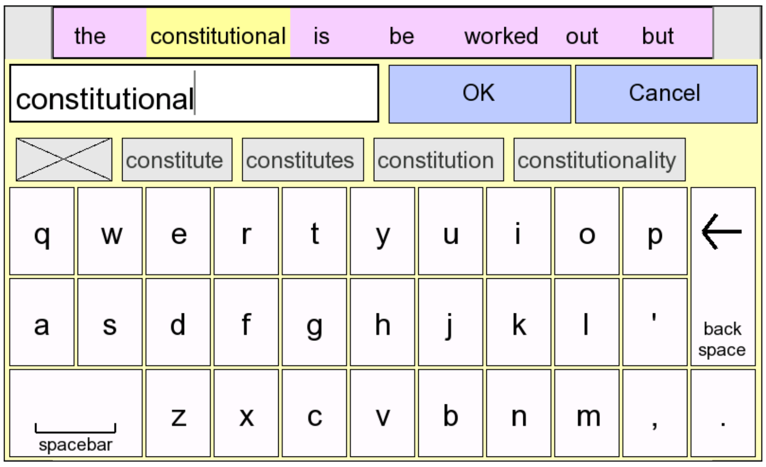


Figure 5.3: After touching the word "constitutional" in the word confusion network, the user is brought to the software keyboard interface. The morphological variants for "constitutional" are shown in a row above the keyboard.

5.3.1.3 Deleting Words

To delete words, the user touches the delete button (a box with a diagonal X, cf. figure 5.2). If the user wants to delete several words at once, the user can slide his or her finger across adjacent delete buttons. Often in speech recognition, the recognizer gets off track and outputs several erroneous words in a row. In such instances, it is particularly useful to be able to cross several delete buttons at once. To make such contiguous delete actions easy, I aligned all the delete buttons at the bottom.

5.3.1.4 Inserting Words

To insert a word, the user can touch the area between two columns. This brings up a keyboard interface that allows the user to choose from a set of word candidates or type a new word (figure 5.3). A second option is to touch a preceding word and type a space and then the new word.

Sometimes, the user's desired word may appear in a different column from where the word is required. A third way to insert a new word is to touch and hold a word for a moment. The background of the word then changes and the

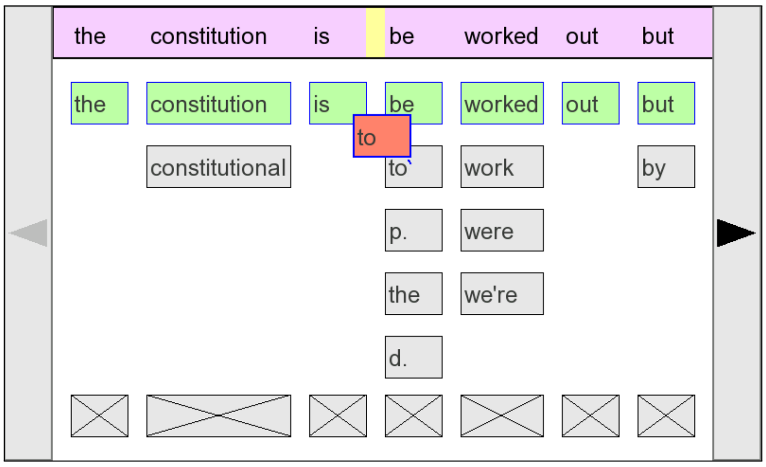


Figure 5.4: The user is inserting the word "to" between the words "is" and "be" by dragging the word "to" from the fourth column to the desired location.

word can be copied to another cluster (figure 5.4). During the copy, the current destination cluster is highlighted in yellow in the top row of the display.

5.3.1.5 Correcting by Crossing

Similar to [91], I allowed users to correct errors by a continuous crossing gesture. Figure 5.5 shows one fluid gesture changing "to" to "of", changing "imports" to "imported", and deleting "and". This crossing-based interaction method is possible because in each column, only one item can be selected. Therefore, the detection algorithm only needs to track the last word or delete button crossed in each column. For example, in figure 5.5 the user has crossed both "but" and the delete button in column 4. The detection algorithm will select the delete button in column 4 since it was the last thing crossed. Users can start crossing anywhere on the display and can cross in any direction.

The theoretical performance of crossing interfaces is of the same mathematical form as Fitts' law [7; 52]. At the same index of difficulty [52], crossing is more efficient or on par with selecting items individually [7].

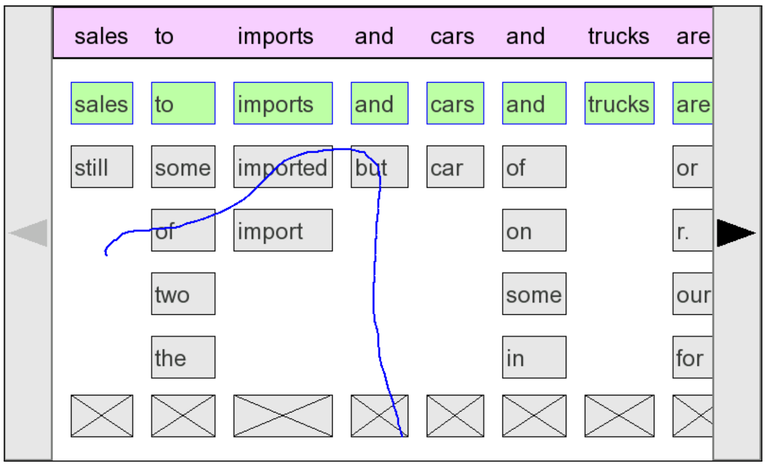


Figure 5.5: Selecting several words and the delete box in one crossing action.

5.3.2 Finding Useful Actions with a Simulated User

I used computational experiments to guide my decisions on what interface actions to include. I also used the experiments to decide how many word alternatives to use in each cluster. The experiments were done by performing recognition on utterances from three standard acoustic test sets (WSJ1 si_dt_05, WSJ0 si_et_05, WSJ0 si_dt_05, 1253 total utterances). After recognition, I created a word confusion network for each utterance. The recognition setup was as described in section 5.5 except I used a vocabulary of the top 5K words in the CSR-III text corpus [64]. This vocabulary resulted in an out-of-vocabulary (OOV) rate of 3.2% on the WSJ test data.

Overall, the test utterances had a 1-best WER of 18%. This is higher than might be expected on these relatively "easy" test utterances. This reflects compromises made to the recognition setup to keep memory and processing demands reasonable for a mobile device. It also reflects the inclusion of OOV words in the test data. I compare mobile and desktop recognition setups in section 5.5.4.

I assumed an "oracle" user, that is, a simulated user who made optimal use of a given confusion network and set of interface actions to correct as many errors as possible in the recognition result. While all errors can be corrected in Parakeet via the software keyboard, in these experiments, the simulated user was assumed

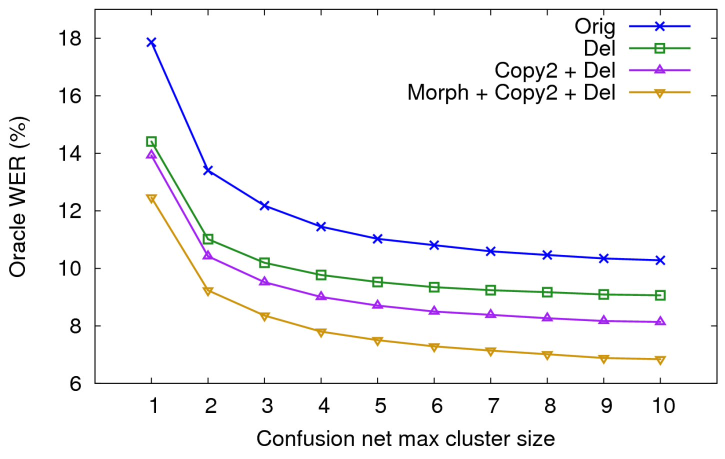


Figure 5.6: Oracle word error rate (WER) as a function of cluster size in the confusion network. The top line is using the original confusion network with no modifications. The other lines show how error rate decreased when more correction features were added.

not to type letters using the software keyboard.

As shown in figure 5.6, increasing the number of words in each cluster allowed more errors to be corrected. The majority of gains were seen by adding the first few alternatives. This guided my decision to use a small cluster size of five. By minimizing the cluster size, I was able to use larger and easier to touch buttons. Adding a delete button to every cluster was shown to substantially reduce errors (*Del* line, figure 5.6). This makes sense as every recognition insertion error can be corrected.

I tested allowing copying words from clusters within two words of each other (*Del+Copy2* line, figure 5.6). This provided a small gain. As shown in figure 5.7, bigger gains were possible when I allowed copying across longer distances. But I doubt users would be likely to copy over such long distances. This is because they would first have to notice their desired word in a distant cluster (possibly off the screen). They would then need to undertake a long drag operation while maintaining constant touch pressure.

Finally, as will be detailed next, I tested a feature that allowed a word to be easily replaced by one of its morphological variants (e.g. replacing “accept” with “accepted” or “acceptance”). This provided further error reductions (*Morph+*

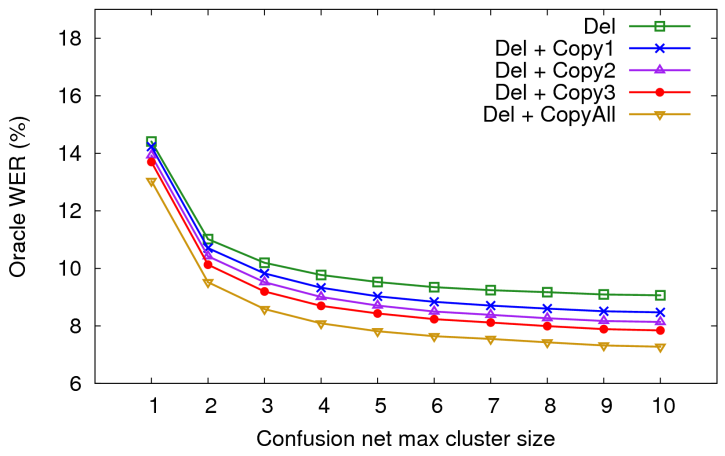


Figure 5.7: Oracle word error rate (WER) as a function of cluster size in the confusion network. The lines show improvement depending on how far the user is assumed to copy words between clusters. *Del* assume no copying (same as *Del* in figure 5.6), *Del+Copy1* assumes copying up to one cluster away, and so on. *Del+CopyAll* assumes copying between any clusters.

Copy2+Del line, figure 5.6).

5.3.3 Word Substitution Prediction

When a user double-taps a word in the confusion network, the keyboard interface opens. In order to try and minimize the need to type a word, I decided to try and predict likely alternative words based on the word the user touched. For example, if the user touched the word "constitutional", the interface might propose morphological variants of the word such as: "constitute", "constitutes", etc. (figure 5.3).

Before I settled on displaying morphological variants, I considered several possibilities. One possibility was to predict words that are acoustically close. Acoustically close words have similar, but not identical, phone sequences in a pronunciation dictionary. Another possibility was to propose word candidates based on the preceding word (using a forward language model), or based on the following word (using a backward language model). For details, on how I

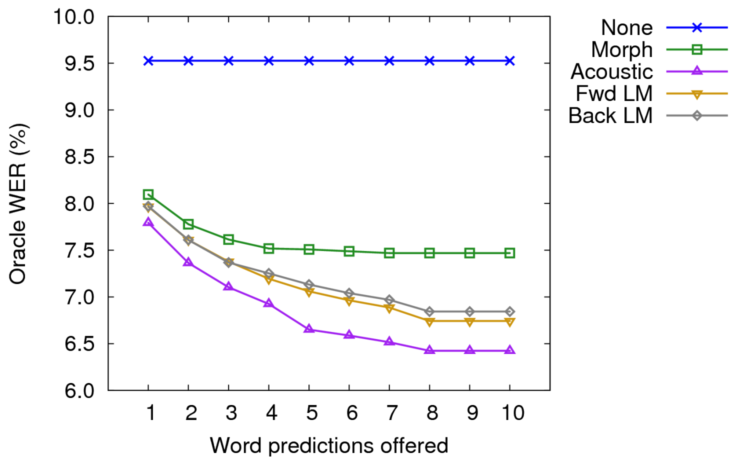


Figure 5.8: Oracle word error rate (WER) as a function of how many word predictions were given by the software keyboard. The different lines show the performance of using different types of word predictions. Prior to entering the software keyboard, the confusion network interface was assumed to have displayed five word alternatives plus the delete box.

generated these word alternatives, see [146].

I again simulated a perfect user's performance using my set of confusion networks. Note that in the actual interface, word predictions are shown only after the user touches a word in the main display. The oracle is assumed to know which of the various words in the main display to touch in order to best correct the recognition result.

As shown in figure 5.8, providing more predictions provided greater error reductions. The majority of the gains were seen by 5 predictions. This led to my decision to use a single prediction row placed above the software keyboard (figure 5.3). The prediction row always includes a delete button on the left side. This button allows the current word in the text entry box to be deleted. To the right of delete, I include as many predicted words as would fit on the screen (typically around 4 or 5 words).

While acoustic predictions performed best, they are also highly unintuitive to users. As an example, "aback" is acoustically similar to "attack". It would be difficult for a user to know they should touch "aback" if they wanted "attack".

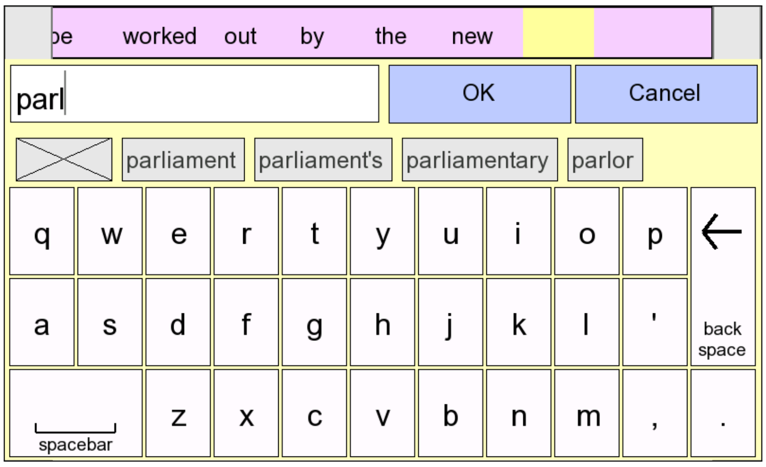


Figure 5.9: The predictive software keyboard. The user has typed "parl" and the most likely ways to complete the word are displayed in a row above the QWERTY keyboard.

Language model predictions also suffer from being unintuitive. They depend on surrounding context rather than the actual word the user touched. For these reasons, I decided to use morphological variants. It is straightforward to explain to users that a variant of a word which differs in ending, possessiveness, or grammatical number, can usually be obtained by touching the word and checking the predictions.

5.4 Predictive Software Keyboard

Sometimes the user's desired word may not appear anywhere in the interface. To allow entry of arbitrary words, I added a predictive software keyboard.

5.4.1 Software Keyboard Design

Previous research suggests that pointing performance is severely degraded when users are walking [37]. I therefore tried to make the typing keys as big as possible (figure 5.9).

Another explicit design decision was to make the keyboard literal – each key pressed is output immediately (as with an ordinary desktop keyboard). There are some systems proposed in the literature that can improve typing precision by inferring the intended letter keys rather than making a literal interpretation, e.g. [62; 88]. However, these solutions are based on machine learning algorithms and could introduce further recognition errors should their inferences be wrong. Since I wanted to avoid cascading errors, I opted for a traditional keyboard to provide a fallback method of text entry.

When a key was hit, its background color was changed to green and then faded back to white over a 350 ms period. This provided feedback about the last few keys hit without the user needing to look at the text entry box. Without this feedback, it would be difficult for the user to know if a particular attempt to type a letter had been successful or not.

5.4.2 Typing Prediction

I complemented the keyboard with typing prediction [40]. The keyboard suggests the most likely words given what the user has typed so far (figure 5.9). It finds predictions by searching a prefix-tree with 64K words. The prediction display is populated with the most likely words (given a unigram language model) matching the currently typed letters. The displayed predictions are sorted in alphabetical order.

Typing prediction results were displayed on the screen 350 ms after the last key press. This delay was introduced for two reasons. First, the prediction lookup and screen redraw introduces lag which could interfere with users' typing. Second, a dynamically changing graphical interface might distract users from their typing task.

As the user enters text, it is displayed in a text entry box. The user is allowed to move the current cursor position in the box by using the left and right hardware buttons on the N800 (the Left and Right buttons shown in figure 5.13). The cursor position can also be moved by touching the desired position. Moving the cursor allows arbitrary edits to words. For example, in figure 5.9, the user may actually want to write “parole”. In this case, the user can move the cursor to the left and then insert the missing letter “o”.

5.5 Mobile Speech Recognizer

Parakeet’s speech recognition was based on CMU Sphinx and used the PocketSphinx decoder [74]. In this section, I give details of the recognition-related components of Parakeet.

5.5.1 Acoustic Model

For fast performance, I opted for a semi-continuous acoustic model. My acoustic model was trained following the recipe described in [147]. I used an HMM topology of 5 states with skip transitions and 256 codebook Gaussians. I trained cross-word triphones using 39 CMU phones without stress markings plus silence. I parameterized audio into a 51-dimensional feature vector consisting of 12 Mel-frequency cepstral coefficients plus their short-term deltas, long-term deltas, delta deltas, and three 0^{th} cepstral power terms.

My US English model was trained on 211 hours of WSJ [5; 57] training data, downsampled to 8 kHz. I used 8000 tied-states and the CMU pronunciation dictionary.

My UK English model was trained on 16 hours of WSJCAM0 [125] training data, downsampled to 8 kHz. I used 4000 tied-states and the BEEP pronunciation dictionary. I mapped the BEEP phone set to the CMU phone set and added missing words from the CMU pronunciation dictionary. From the original gender-independent model, I created gender-dependent models using maximum likelihood linear regression (MLLR) adaptation [96] of the means followed by maximum a-posteriori (MAP) adaptation [59] of the means, mixture weights and transition matrices.

5.5.2 Audio Capture and Normalization

I captured audio on the N800 using a Blue Parrot B150 Bluetooth headset. I chose this headset as it has a close-talking boom microphone and a long battery life. Audio from the headset was sampled at 8 kHz and was obtained using the GStreamer framework [2].

Normally, to improve recognition accuracy, the means of the acoustic features observed in an utterance are subtracted from each frame of audio in that utterance. This technique, known as cepstral mean normalization/subtraction [99], normally uses the entire audio of an utterance. But in order to reduce the recognition delay, Parakeet streamed audio to the recognizer as soon as the microphone was enabled. As a consequence, I used cepstral mean normalization based on a prior window of the user's audio. When Parakeet starts, the cepstral mean vector was initialized to a value based on my own audio recorded on the N800. This initial mean vector was then adjusted to the user's voice as Parakeet was used.

5.5.3 Language Model

I trained a trigram language model using: newswire text from the CSR-III text corpus (222M words) [64], interpolated modified Kneser-Ney smoothing, and the WSJ 5K word list (without verbalized punctuation). Since my test sentences were taken from the CSR set-aside directory, I purposely chose the WSJ 5K vocabulary in order to introduce a small number of out-of-vocabulary (OOV) errors. I thought it was important to validate my design in the face of OOV errors as they are typically unavoidable in real-world recognition tasks.

The language model was one of the dominating factors in the memory footprint of my system. Rather than training with n-gram count cutoffs, I instead used no cutoffs and performed entropy-pruning [137] using a threshold of 5×10^{-8}. The resulting language model had 1.4M bigrams and 2.6M trigrams. As in [113], I found entropy-pruning produced compact and well-performing models.

5.5.4 Mobile versus Desktop Recognition Setups

I compared the recognition performance of the US-English acoustic model designed for mobile use (as described in section 5.5.1) versus a model designed for use on a more powerful desktop computer. The desktop model differed in a number of ways. The desktop model used audio sampled at 16 kHz, parameterized into a 39-dimensional feature vector consisting of 12 Mel-frequency cepstral coefficients plus the 0$^{\text{th}}$ cepstral, deltas and delta deltas. The desktop models used a 3-state left-to-right HMM topology, 8000 tied-states and 16 continuous Gaussians per state.

I tested the mobile and desktop acoustic models on a set of utterances taken from three standard test sets (WSJ1 si_dt_05, WSJ0 si_et_05, WSJ0 si_dt_05, 1253 total utterances). For recognition, I used the language model described in section 5.5.3 and a WSJ 5K vocabulary. The test utterances had a very low OOV rate of 0.1% using the WSJ 5K vocabulary. I tested both models with two sets of parameters that controlled the recognizer's search effort. The FAST parameters aggressively pruned the recognizer's search in order to speed recognition. The FAST parameters were those used on the mobile device in the user study (section 5.6). The SLOW parameter set resulted in less pruning and reflect settings that might be used on a desktop computer.

As shown in table 5.1, the mobile acoustic model provided the fastest recognition, but was significantly less accurate than the desktop model. As evidenced by the lower accuracy of the FAST parameter set, significant search errors were incurred in order to speed recognition. Results were on a 3.3 GHz desktop computer.

5.5.5 Lattice Processing

The word confusion networks used in Parakeet's primary correction interface were created from the recognition lattices returned by PocketSphinx [74]. There were several problems with using Sphinx's lattices to create confusion networks. Firstly, while PocketSphinx was using a trigram language model for decoding, the

Acoustic model	Params.	×RT	WER ± 95% CI
Mobile	FAST	0.08	12.34 ± 0.76
Mobile	SLOW	0.25	9.93 ± 0.66
Desktop	FAST	0.76	9.75 ± 0.64
Desktop	SLOW	1.51	7.74 ± 0.55

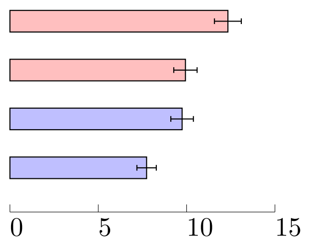


Table 5.1: Recognition performance using acoustic models designed either for mobile or desktop use. I also varied the parameters that controlled pruning during the recognizer's search. Confidence intervals were calculated using per-utterance bootstrap resampling [17].

lattices exposed were only bigram lattices (i.e. edges could have multiple trigram contexts). It is not clear why this is the case, perhaps due to limitations in the history structures maintained by Sphinx.

Secondly, the lattices (particularly in challenging acoustic conditions), could be quite large. Such large lattices, if left unchecked, could create unacceptable memory- or processing-demands on a mobile device.

To address these problems, and also to produce the required confusion network representation, the following processing steps were performed on the lattice returned by PocketSphinx:

- **Forward/backward reduction** – Redundant lattice nodes were combined with a single forward and backward reduction pass [159]. This process maintains the same lattice paths and probabilities while making the lattice smaller.

- **Posterior pruning** – Lattice nodes with a posterior probability less than a fixed constant times the posterior probability of the best path were removed. This removed low-probability hypotheses resulting in a smaller lattice. In the experiments reported here, I pruned nodes with a probability less than 1×10^{-6} times the best path probability.

- **Compact trigram expansion** – The lattice was expanded and rescored using a compact trigram expansion algorithm [159]. This allowed the proper trigram language model probabilities to be assigned to all paths in the lattice.
- **More posterior pruning** – Another round of posterior pruning was performed to reduce the size of the expanded lattice.
- **Create word confusion network** – A word confusion network was created from the expanded lattice. To transform the lattice into a confusion network, I used an algorithm based on the one implemented by SRILM [138].
- **Prune word confusion network** – The maximum size of each confusion network cluster was limited to the number of words that Parakeet intended to display. In addition, normally a confusion network contains numerous clusters in which the "delete" hypothesis dominates (i.e. the recognizer thought the cluster should not generate anything). Including all these delete clusters would greatly increase the number of clusters in Parakeet's display. I removed delete clusters in which the delete word's probability exceeded a threshold of 0.9.

5.5.6 Speaker Adaptation Environment

Since I knew mobile recognition would be challenging, I wanted to improve accuracy by adapting the recognizer's acoustic model to the user's voice and to the Bluetooth microphone. Adaptation was done by having the user read a set of sentences with known transcriptions. Typically, with a desktop dictation package (such as Dragon), users perform this adaptation while in front of their desktop computer.

When taking speech recognition mobile, the question arises whether it is worth adapting the speech recognizer in a mobile, rather than a desktop environment. Perhaps while using a speech application while mobile, a person's voice changes due to heavier breathing or due to competing attentional demands. It might be possible to obtain better recognition accuracy by collecting adaptation data in the target environment.

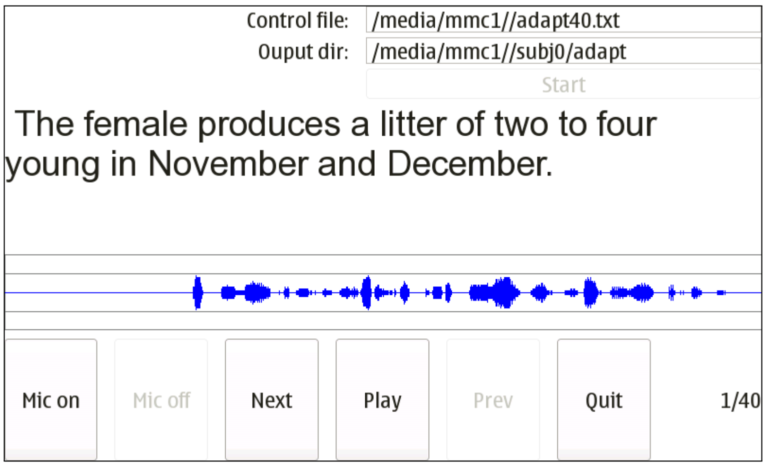
Price and colleagues investigated this question in [120]. They had half of their participants provide adaptation data while walking on a noisy treadmill, while the other half provided data while seated in an office (with simulated treadmill noise playing). While not statistically significant, they found that users who had performed adaptation while on the treadmill had a lower word error rate (WER) both when tested on the treadmill and when seated. They suggest that adapting in a more demanding condition might improve recognition both while seated and while walking (though they provide no insight into why this might be).

To address my concerns about how to best collect adaptation audio, I decided to conduct a small, within-subject experiment. I had four US-English speakers record three identical sets of adaptation data on the N800:

- **Desktop** – Indoors while seated. Recorded at a 16 kHz sampling rate on a desktop with a wired Sennheiser PC 166 microphone.
- **Indoor** – Indoors while seated. Recorded at a 8 kHz sampling rate on the N800 with a wireless Blue Parrot B150 microphone.
- **Outdoor** – Outdoors while walking. Recorded at a 8 kHz sampling rate on the N800 with a wireless Blue Parrot B150 microphone.

For an adaptation set, I used 40 phonetically diverse sentences from the WSJ corpus [118]. As a test set, speakers also recorded 125 sentences from the CSR-III set-aside directory [94]. The test set sentences were recorded both indoors and outdoors immediately after recording the corresponding adaptation set. The test set sentences were not recorded using the desktop setup.

All speakers recorded the desktop adaptation set first (some on a previous day). Speakers then either did the outdoor or indoor recording next (balanced between speakers). Speakers used a custom prompt recording application (figure 5.10). The application presented sentences in sequence in a large font. No recognition was performed during the sessions. Audio was simply recorded and



The female produces a litter of two to four young in November and December.

Figure 5.10: The application for collecting adaptation audio data from users. It displays the text to be read as well as a waveform representation of the user's last recording.

experiments conducted later using a desktop computer. The audio recorded using the desktop setup was downsampled to 8 kHz before using it for adaptation.

As shown in table 5.2, performing any sort of adaptation was better than no adaptation, reducing WER by about 20% relative. The desktop adapted models performed worse than either the indoor or outdoor adapted models. In my subjective judgment, the audio quality of the desktop microphone was superior to that of the Bluetooth microphone. But it appears that adapting to the microphone used to collect the test audio was more important. I also found that recognition outdoors was much harder than indoors, increasing WER by 45% relative.

The error rates of indoor and outdoor adapted models were very similar regardless of whether they were tested on audio recorded in the same environment or not. Since there was no significant accuracy difference, I decided to collect adaptation data indoors. I also feel indoor adaptation is more practical as it allows users to learn to dictate to the computer in a more comfortable and private setting.

Adapt	Test	WER ± 95% CI
none	indoor	15.5 ± 2.2
desktop	indoor	13.0 ± 2.0
indoor	indoor	12.4 ± 2.0
outdoor	indoor	12.5 ± 2.0
none	outdoor	21.2 ± 2.6
desktop	outdoor	19.0 ± 2.4
indoor	outdoor	18.0 ± 2.3
outdoor	outdoor	17.9 ± 2.3

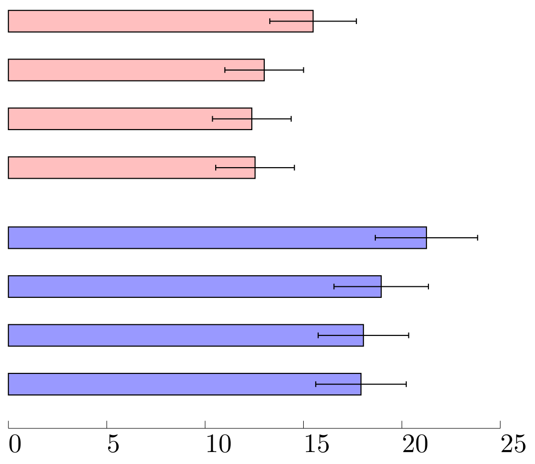


Table 5.2: The effect of adaptation environment on speech recognition word error rate (WER). The bars show means and 95% confidence intervals. Confidence intervals were calculated using per-utterance bootstrap resampling [17].

5.6 User Study

5.6.1 Participants and Apparatus

I recruited four participants (3 males and 1 female) from the university campus. 3 participants used the UK acoustic model and 1 participant used the US model. Their ages ranged from 22 to 39. All participants were novice speech recognition users.

Participants used a Nokia N800 Internet Tablet (figure 5.1). The N800 uses an ARM1136 CPU clocked at 320 MHz. The physical dimensions of the device (length × width × thickness) is 75 × 144 × 13 mm. The screen has a resolution of 800 × 480 pixels and a physical size of 90 × 55 mm.

Audio was recorded using a Blue Parrot B150 wireless headset microphone. The built-in camera of the N800 was used to record video of the participant. From the video, it was possible to tell whether the participant was looking at the device or looking away.

In the outdoor condition, the N800 recorded participants' location via a small BlueNEXT BN909GR global positioning system (GPS) device which was physically affixed to the microphone headset. The GPS data was intended to provide information about the participants' walking speed during the study. I also wanted to analyze if particular parts of the outdoor course were more subject to recognition errors due to localized noise sources.

5.6.2 Method and Setup

Participants took part in a single two-hour session. Participants first trained the speech recognizer for 10 minutes. Training consisted of reading 40 phonetically diverse sentences from the WSJ corpus. They did this while seated in a quiet office. Due to a technical problem, participants' adapted acoustic models were not used in the user study. The results presented here reflect recognition using a speaker independent acoustic model.

After training, participants received a 5 minute introduction to using Parakeet. They were then given 10 minutes to practice (with the experimenter available to answer any questions). On average, participants completed 14 sentences during their practice session.

After the practice session, participants proceeded to either the seated indoor condition or to the walking outdoor condition. In the outdoor condition, participants walked circles around the Mott building of the Cavendish Laboratory. One loop of the course was approximately 0.34 kilometers. Figure 5.11 shows the GPS track of one participant in the outdoor condition. The outdoor condition was carried out on a safe track and under constant supervision. While walking around the track, participants led the way with the experimenter following at about a one meter distance. The order of the conditions and the target sentences received was balanced across participants. Each condition lasted approximately 30 minutes.

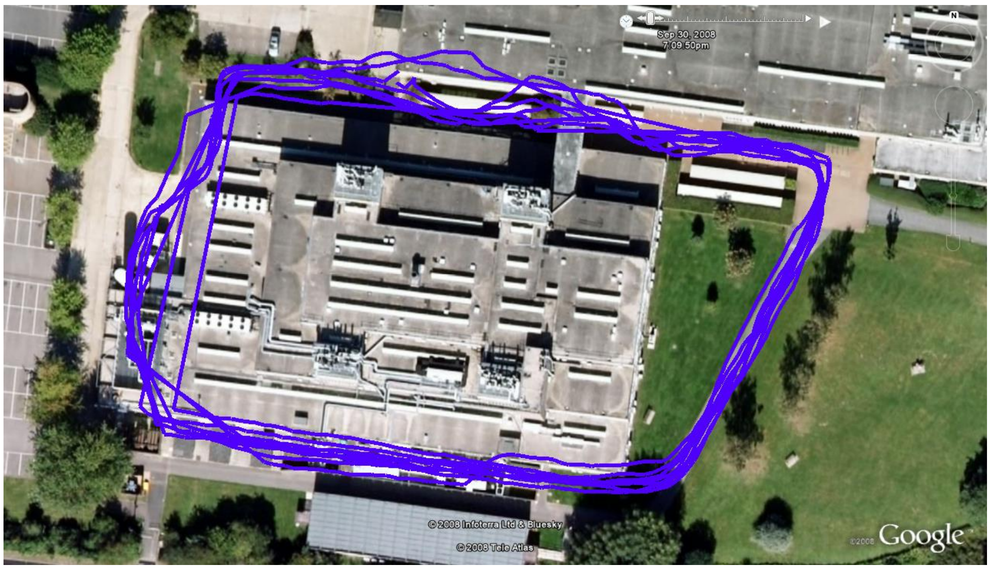


Figure 5.11: One participant's GPS track around the Cavendish Laboratory. This participant made about 8 loops around the building. The image is from Google Earth.

5.6.2.1 Test Sentences Used

I wanted to study the use of my interface given a "usable" level of recognition errors. In my opinion, there is no point in testing speech correction interfaces at high (say $> 40\%$) WER. With so many errors, it would likely be better to not use speech recognition in the first place. My target in these experiments was to have a WER (before correction) of around 15%.

But I knew getting to 15% WER was going to be problematic. First, to obtain reasonable real-time performance, I had to heavily prune the recognizer's search. This heavy pruning resulted in substantial search errors. Second, for practical participant recruitment reasons, I wanted to allow UK-English speakers. Unfortunately, I only had 16 hours of UK-English acoustic training data. This made training an accurate acoustic model difficult. Third, I was using wireless narrowband audio collected in a potentially challenging outdoor acoustic environment.

So to help control recognition error rates, I chose sentences with a low per-word perplexity of 18. These low-perplexity sentences should be easier to recognize since they are well predicted by the language model. The sentences were

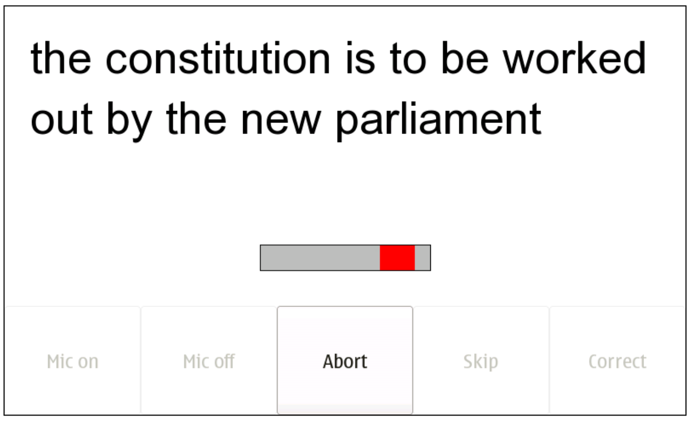

the constitution is to be worked out by the new parliament

Figure 5.12: This screen presents the target sentences to the user. From left to right, the bottom buttons are used to: 1) turn the mic on, 2) turn the mic off, 3) abort a recognition, 4) skip to the next sentence, 5) move to the correction phase. If users needed a reminder of the sentence during correction, they could return to this screen by pressing the center button of the N800's cursor pad.

taken from the set-aside directory of the CSR-III corpus and were excluded from language model training. I chose sentences with between 8 and 16 words (mean = 10.3, sd = 2.1).

I also wanted to test the interface in the face of out-of-vocabulary (OOV) words. So I used a mismatched WSJ 5K vocabulary instead of the top words in the CSR-III corpus. This caused 2.4% of words in my target sentences to be OOV.

5.6.3 User's Task

In both conditions, participants were presented with the target sentences on the screen of the N800 in a large font (figure 5.12). When ready, participants pressed a MIC ON button, read the sentence, and then pressed a MIC OFF button. After a recognition delay, the device beeped and flashed the screen. Participants then hit a CORRECT button to enter the correction interface. After participants corrected the sentence to the best of their ability, the DONE button was pressed to move to the next sentence. The DONE button used a hardware button on the top of

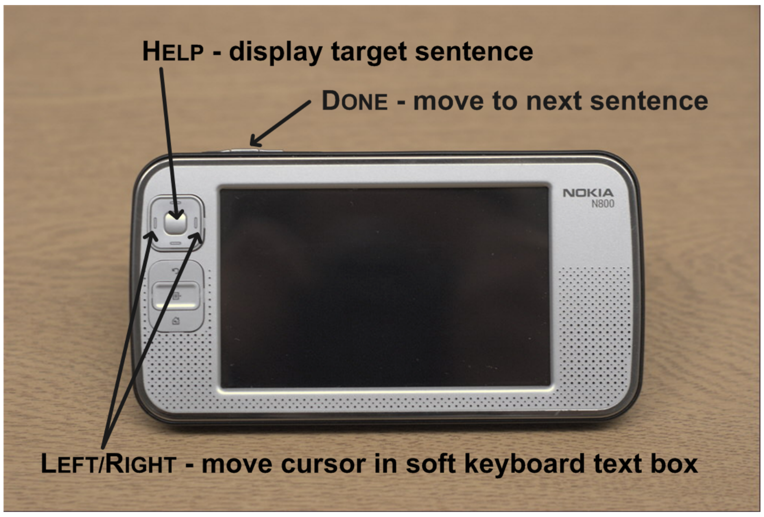

HELP - display target sentence
DONE - move to next sentence
NOKIA N800
LEFT/RIGHT - move cursor in soft keyboard text box

Figure 5.13: Diagram showing the N800 hardware buttons used during the experiment.

the N800 (see figure 5.13). Participants could obtain a reminder of the target sentence by pressing a HELP button which used a hardware button in the center of the N800's cursor pad.

5.6.4 Data Collected

The N800 logged a wide variety of information about participants' interactions. This included low-level information such as touch trace data and high-level information such as what they wrote. In addition, I logged the audio utterances recorded, the recognition lattices, and the word confusion networks.

The N800's built-in camera recorded low resolution 160 × 120 video at 4 frames per second. Using the paired Bluetooth GPS receiver, the N800 also recorded location information every few seconds while outdoors. The video and GPS recording consumed less than 10% of the N800's CPU.

Condition	Text	WER ± 95% CI
indoor	before correction	16.2 ± 4.5
outdoor	before correction	25.6 ± 3.1
indoor	after correction	1.2 ± 1.0
outdoor	after correction	2.2 ± 1.7

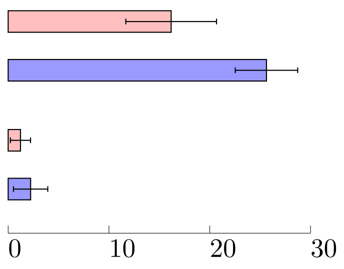


Table 5.3: Novice users' mean word error rates (WER) and 95% confidence intervals. The confidence intervals reflect variance between individuals.

5.6.5 Novice Results

5.6.5.1 Error Rate

Word error rate (WER) was calculated as the word edit distance between the target sentence and the written sentence divided by the number of words in the target sentence. Within their 30-minute time limit (per condition), participants completed about 41 sentences indoors (mean = 40.8, sd = 7.1) and 27 sentences outdoors (mean = 27.3, sd = 4.1).

Table 5.3 shows the mean error rates obtained, indoors and outdoors. The *before correction* error rate is the error rate of the speech recognizer's output. The *after correction* error rate is the error rate after participants corrected the speech recognizer's output. As can be seen in table 5.3, the recognizer's error rate was considerably higher outdoors than indoors. Users corrected most, but not all, recognition errors. Users failed to correct more errors in the outdoor condition.

5.6.5.2 Entry Rate

Entry rate was calculated in words per minute (wpm). I used the standard convention defining a "word" as five consecutive characters. The wpm rates reported include the time spent correcting. For each condition, I computed two different entry rates. The *actual entry rate* was based on the actual time interval between

Condition	Entry rate	wpm ± 95% CI
indoor	actual	18.4 ± 1.8
outdoor	actual	12.8 ± 0.6
indoor	potential	36.6 ± 4.4
outdoor	potential	25.6 ± 2.7

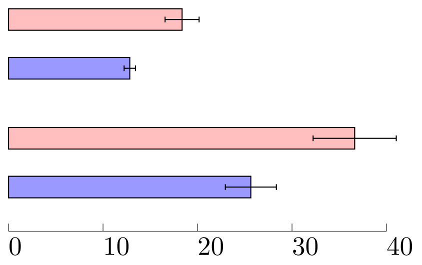


Table 5.4: Novice users' mean entry rates in words per minute (wpm) and 95% confidence intervals. The confidence intervals reflect variance between individuals. The bottom two rows represent an upper-bound on performance assuming no recognition delay.

the user pressing the MIC ON button and pressing the DONE button. The *potential entry rate* subtracted out the recognition delay (the time between pressing the MIC OFF button and the recognition being available). This represents an upper-bound on what users might achieve using a faster device or recognizer.

Table 5.4 shows the mean entry rates. As expected, participants were faster indoors than outdoors. The recognition delays experienced by the novices drove down their actual entry rates. As shown in the bottom two rows of table 5.4, the novices had the potential to write much faster if they had not had to wait for recognition to complete.

5.6.5.3 Correction Method Usage

Participants could correct errors either by using the word confusion network or by using the software keyboard. If participants forgot the sentence they were trying to write, they could invoke a help screen to display the sentence again.

Indoors, participants spent 62% of their correction time in the word confusion network, 32% in the predictive software keyboard, and 6% in help. Outdoors, participants spent 56% of their correction time in the word confusion network, 33% in the software keyboard, and 11% in help. Over a third of correction time

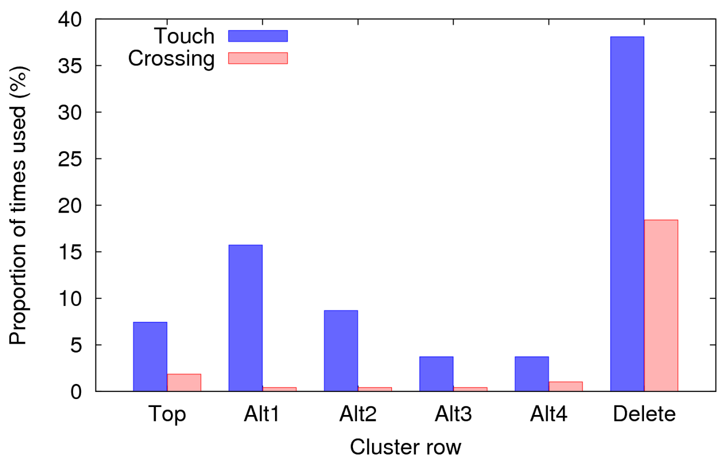


Figure 5.14: Proportion of times novice users touched or crossed a certain row in the confusion network. *Top* indicates the row containing the 1-best words, *Alt1-Alt4* indicate the alternative word rows (*Alt1* being the topmost alternative, *Alt2* being the next alternative, etc.), and *Delete* indicates the delete button row.

was spent using the software keyboard. This demonstrates the importance of implementing a good fallback correction method.

5.6.5.4 Word Confusion Network Usage

Figure 5.14 shows which buttons were used in the word confusion network interface. The most common action was to use the confusion network to delete words (56% of usage). When deleting, touch was used the most frequently (38% of all usage) but crossing was also common (18% of all usage).

When substituting words, selections decreased in frequency as a function of how far away the words were from the 1-best result (*Alt1-Alt4* in figure 5.14). This validated my computational results which showed the first few alternatives were the most useful for corrections. Users most often selected single buttons via touch. When they did select multiple buttons via a crossing gesture, they primarily selected delete buttons. This showed aligning delete buttons in a single row was a useful feature.

In the complete set of all outdoor sessions, novices wrote 273 sentences. Of these, 82 had a completely correct 1-best result. Users completed 80 of these completely correct tasks without making any unnecessary user interface actions (such as touching a word or invoking the keyboard). In 27 of the 273 sentences, the errors in the sentence could be corrected completely using only the word confusion network. Users corrected 26 of these sentences using only the confusion network. This shows that users took advantage of the confusion network whenever possible rather than invoking the software keyboard.

Out of a total of 416 selections in the word confusion network, 374 (90%) were touch actions and 42 (10%) were crossing actions. The feature allowing words to be copied between clusters (figure 5.4) was not popular and was only used three times.

5.6.5.5 Scrolling

Only 2% of recognition results fit completely on one screen. The average width required to display an entire result was 1083 pixels (sd = 266). The display width (minus the scroll buttons) was 700 pixels. So on average, participants needed to scroll right at least once in order to inspect their sentence.

5.6.5.6 Software Keyboard Usage

While indoors, 17% of keyboard presses were the backspace key. While outdoors, 25% of keyboard presses were the backspace key. The increased frequency of backspace outdoors indicates that walking degraded users' ability to type accurately.

In addition to spelling out a word, participants could also make use of the typing prediction. In total, participants wrote 265 words with the keyboard. When typing those words, participants used the typing prediction 54% of the time. On average, participants typed about 3 letters (mean = 3.3) before selecting a prediction. When participants did not use prediction, the desired word had been displayed by the system 70% of the time. In these cases, I found on average the

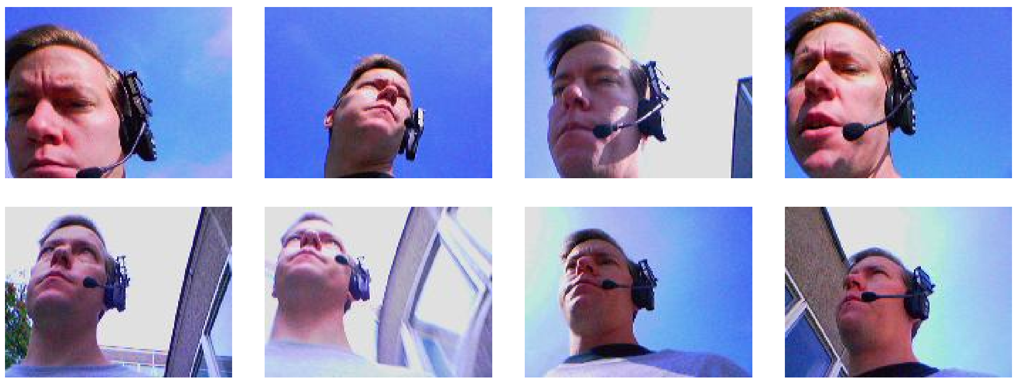


Figure 5.15: Sample video frames from one of my outdoor sessions.

user only needed to type a few additional letters (mean = 1.6) to complete their word. This is probably why they ignored the correct typing prediction.

5.6.5.7 User's Speed and Location

I had hoped to use the GPS tracking data to analyze user's walking speed during the experiment. In addition, I wanted to see if recognition performance was influenced by various point noise sources around the outdoor course. Unfortunately, the GPS tracking information was not accurate or reliable enough. In any future trial, a better way of tracking the user's location would be helpful.

5.6.5.8 Video of User

A detailed analysis of the video taken during the study was not performed. I did however look at the video to assess the camera's usefulness in any future trial. Despite the video's low resolution and frames-per-second, it does appear possible to judge whether the user is looking at the device or not (figure 5.15). This would allow an analysis in which user's attentiveness to the device could be measured (similar to [114]).

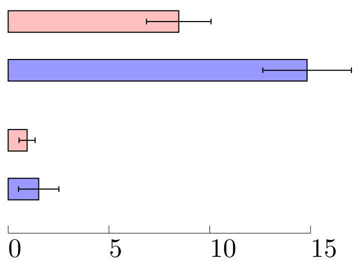
Condition	Text	WER ± 95% CI	
indoor	before correction	8.5 ± 1.6	
outdoor	before correction	14.8 ± 2.2	
indoor	after correction	0.9 ± 0.4	
outdoor	after correction	1.5 ± 1.0	

Table 5.5: Expert user's mean word error rate (WER) and 95% confidence intervals. The confidence intervals reflect the individual's variance.

5.6.6 Expert Pilot Study

To illustrate the potential of speech as a viable mobile text entry method, I tested my own performance using Parakeet. I performed the user study as described previously but over seven sessions. Instead of a 30-minute time limit, I completed 45 sentences per condition. I used the US acoustic model. I have several years of experience using speech recognition systems.

5.6.6.1 Expert Error Rate

Table 5.5 shows the word error rate before and after I finished correction. My recognition WER was much lower than the novice users (indoors 8% versus 16%, outdoors 15% versus 26%). While some of this difference may be due to my speech recognition experience, another factor is my use of the US acoustic model. With substantially more training data, the US model is typically more accurate than the UK model (which 3 of the 4 novice participants used).

5.6.6.2 Expert Entry Rate

Table 5.6 shows my text entry rates in each condition, averaged over all sentences in every session. Despite relatively long recognition delays, my text entry rates were surprisingly good. While, as expected, walking outdoors slowed entry, it

Condition	Entry rate	wpm ± 95% CI
indoor	actual	24.4 ± 0.7
outdoor	actual	19.6 ± 0.7
indoor	potential	53.2 ± 1.9
outdoor	potential	44.8 ± 2.0

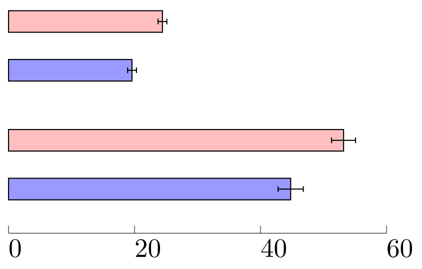


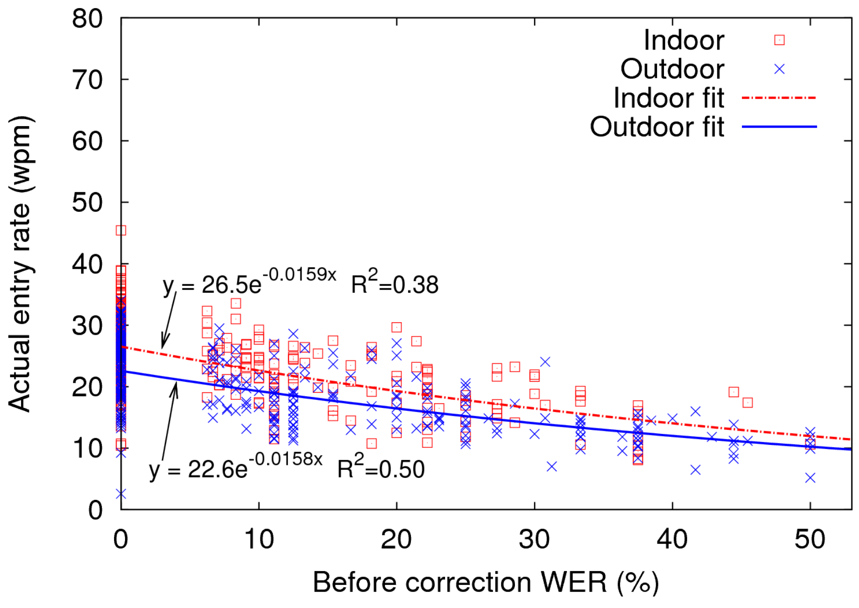
Table 5.6: Expert user's mean entry rates in words per minute (wpm) and 95% confidence intervals. The confidence intervals reflect the individual's variance. The bottom two rows represent an upper-bound on performance assuming no recognition delay.

did so only by about 20%. Even with the recognition delays, I was able to write at almost 20 wpm while walking. If I removed the time I spent waiting for recognition, my writing speed over doubled to 45 wpm. While obviously having no recognition delay is not realistic, some of these gains should be realized as devices and recognizers improve.

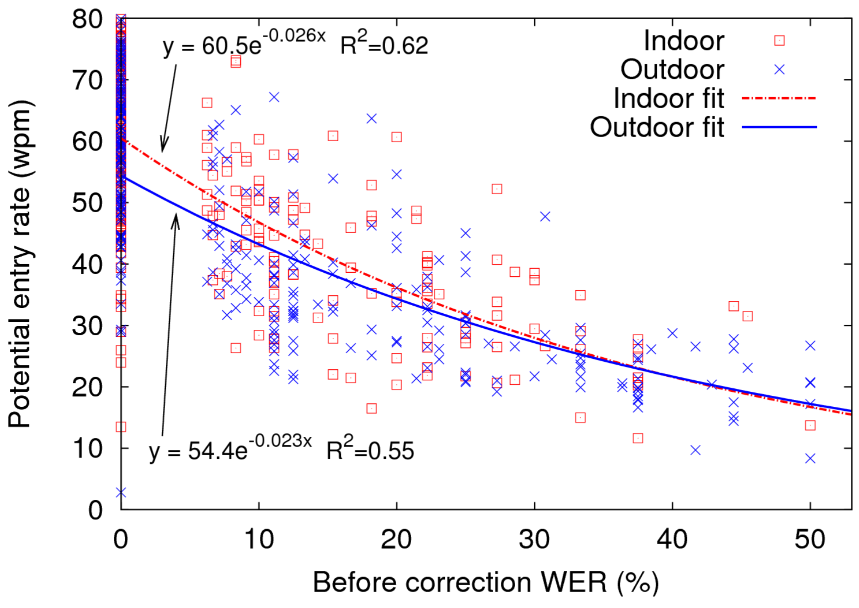
Figure 5.16 show how entry rate varied in the face of recognition errors. As can be seen by the number of points on the left side of figure 5.16a, many sentences were recognized with 0% WER. Unsurprisingly, I usually completed these sentences the fastest. However, as words errors started to occur, the decrease in entry rate was not as dramatic as one might expect. For example, in the 10% word error range, my entry rate dropped by only about 15% in both conditions. As shown in figure 5.16b, removing the recognition delay allowed much faster potential entry speeds.

5.6.6.3 Expert Correct Recognitions

In my expert pilot study, 48% of sentences were recognized completely correctly. Figure 5.17a shows the distribution over writing speed on these utterances. Removing the recognition delay (figure 5.17b), the entry rate became more sensitive to the small differences in time it took me to confirm the recognition result. This

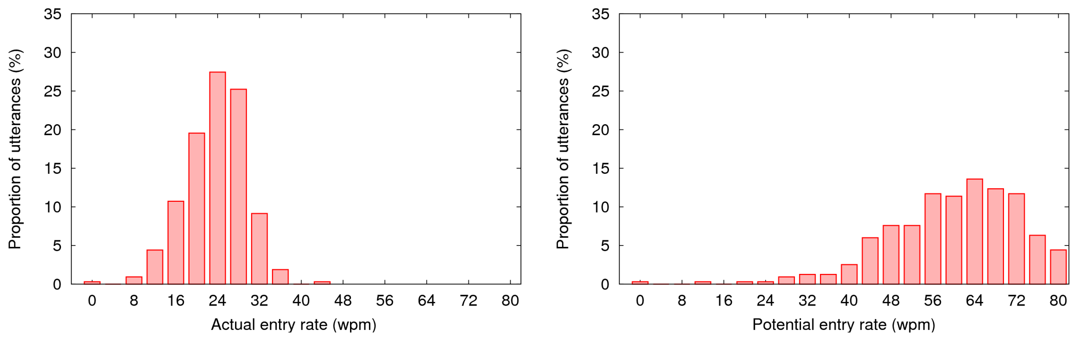


(a) Actual entry rate



(b) Potential entry rate

Figure 5.16: Expert's utterances by entry rate (wpm) and word error rate (WER). (a) shows actual performance of the expert, (b) shows potential writing performance assuming no recognition delay. See figure 5.17 for a detailed view of utterances at 0% WER.



(a) Actual recognition delay (b) No recognition delay

Figure 5.17: Distribution over the expert's entry rate on utterances that were recognized completely correct (0% WER). (a) shows actual performance, (b) shows potential performance assuming no recognition delay.

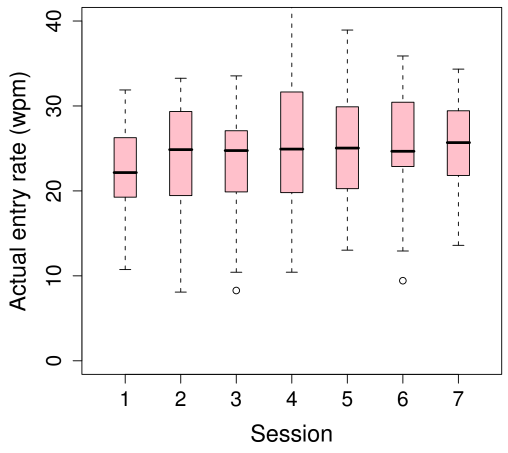
suggests an opportunity to improve entry rates by optimizing the interface to quickly confirm completely correct recognitions.

5.6.6.4 Expert Performance by Session

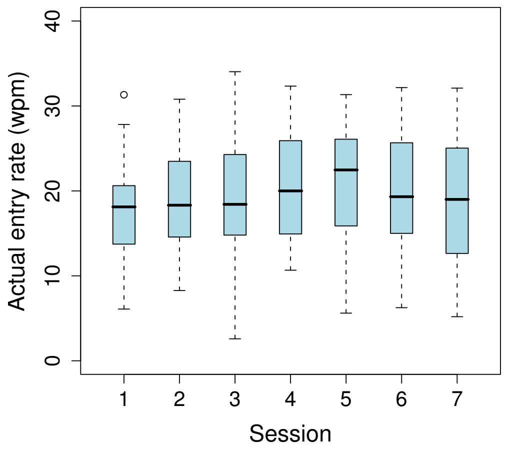
Figure 5.18 shows the text entry speed and recognition accuracy for each of my expert sessions. My performance did not improve much over the course of the seven sessions. This suggests that from the start of the pilot, I was driving the interface about as fast as possible.

5.6.7 Participant Variability

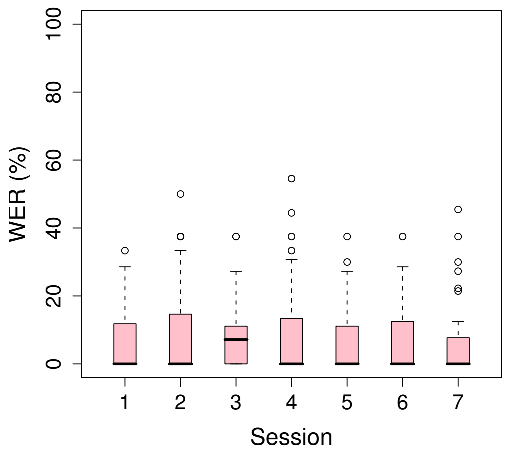
Figure 5.19 shows the text entry rate and recognition WER for each participant (both novices and expert). As expected, the expert is noticeably different from the novices. Outdoors, there was a high degree of variability among the participants in WER. This is likely due to the varying acoustic conditions at the time of each session.



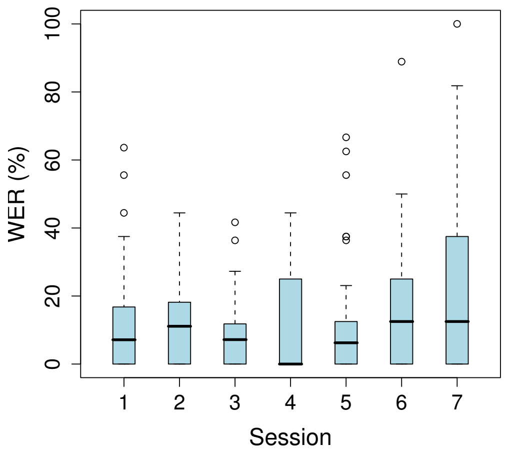
(a) Actual entry rate, indoors



(b) Actual entry rate, outdoors

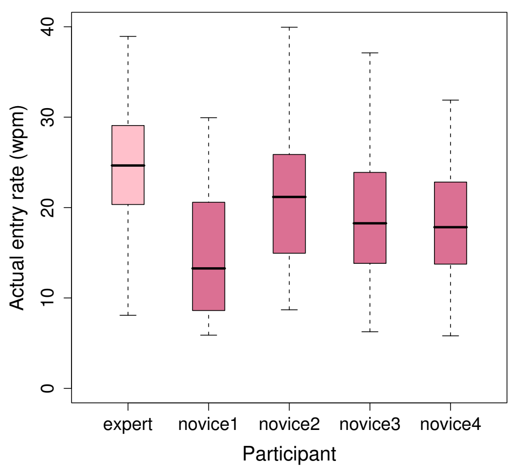


(c) Before correction error rate, indoors

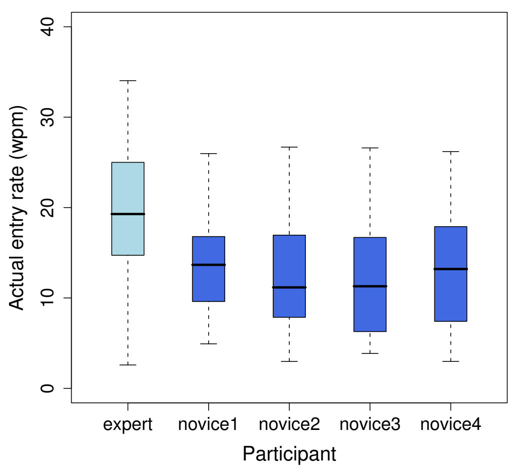


(d) Before correction error rate, outdoors

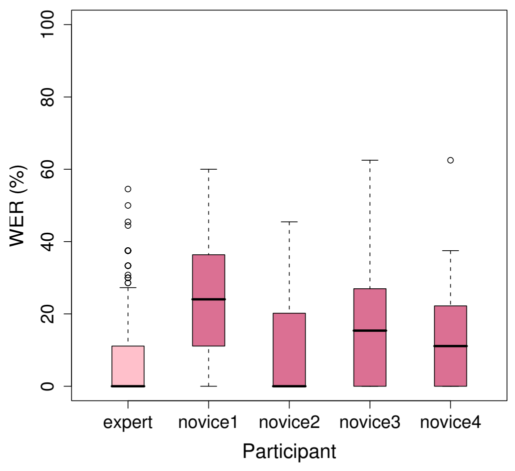
Figure 5.18: Expert's text entry rates (top) and recognition error rates (bottom) for each session. The box represents the interquartile range with the black line at the median.



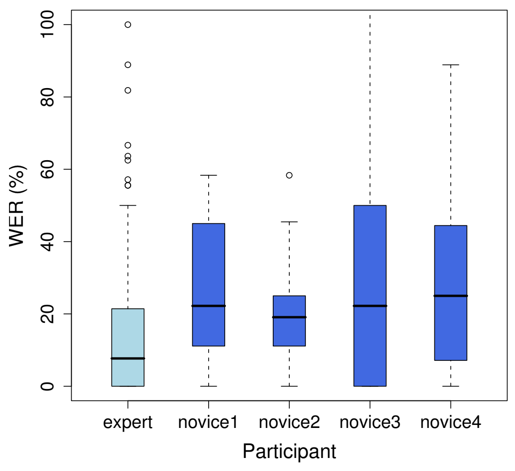
(a) Actual entry rate, indoors



(b) Actual entry rate, outdoors



(c) Before correction error rate, indoors



(d) Before correction error rate, outdoors

Figure 5.19: Text entry rates (top) and recognition error rates (bottom) for each participant. The box represents the interquartile range with the black line at the median.

5.6.8 Analysis of Test Sentences

I chose "easy" test sentences with a low perplexity of 18. In my experiment, novices had an indoor WER of 16% and 26% outdoors. So my choice of test sentences was successful in obtaining a reasonable recognition error rate. A possible problem with these sentences is their error distribution may differ from what you might get on a more difficult recognition task using more expensive recognition. For example, perhaps the low-perplexity sentences exhibit a much higher percentage of completely correct recognition than is realistic.

To investigate this, I compared recognition from my user trial performed on the N800 versus recognition of a collection of WSJ test sets performed on a desktop computer. For the WSJ data, I used WSJ0: si_et_05 si_dt_05 si_dt_jr, and WSJ1: si_dt_05 si_dt_st/sjm si_et_st/sjm si_et_h1/wsj64k. To give a similar average sentence length to the sentences in my user trial, I used WSJ utterances with a length of 16 words or less (1035 utterances). Using a 20K language model, these utterances had a perplexity of 155. I performed recognition using the 20K language model and a speaker-independent US-English continuous wide-band acoustic model. Decoding took about 2.4 $\times$ real-time (2.4 times as long as the input audio) on a 3 GHz desktop computer. During my user trials, the combined novice and expert utterances had a WER of 15.6% with an OOV rate of 1.8%. In the desktop recognition experiment, the WSJ utterances had a WER of 17.9% with an OOV rate of 1.9%.

As shown in figure 5.20, the distribution of the utterance error rates was similar between my user trial data and the WSJ test sets. This suggests that my "easy" test sentences provided a realistic approximation of how the interface might work in the future given a harder recognition task and more expensive acoustic and language models.

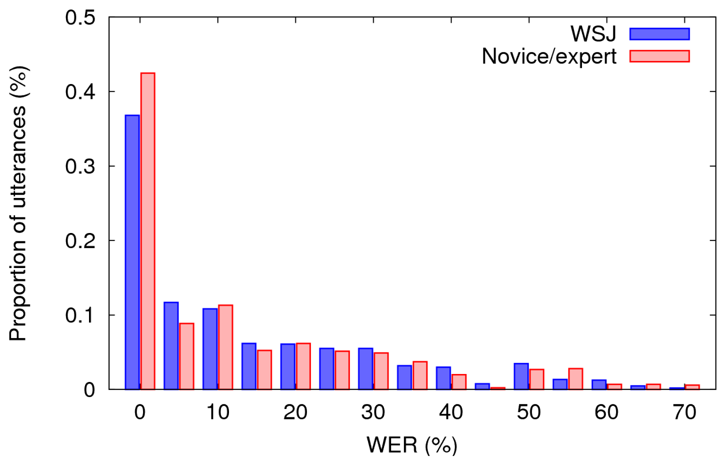


Figure 5.20: Proportion of utterances with a given WER (bucketed in 5% increments). The distribution of utterance WER on the harder WSJ test data using more expensive recognition was broadly similar to what users in my experiment experienced on easier test data using cheaper recognition.

5.7 Discussion

5.7.1 Limitations

5.7.1.1 Number of Participants

My novice user trial used only four people. My intention was not to collect enough data to make strong statistical conclusions on writing speed, error rates, etc. Rather, my goal was to test the interface and see how novices used the interface. In addition, I wanted to gain experience in performing an experiment in an actual mobile environment. In this respect, I think the small trial was beneficial and provided important practical and design insights (to be discussed in section 5.7.2).

My expert pilot study only used myself. Because I designed and built Parakeet, I know the interface better than anyone. Additionally, I am an experienced speech recognition user. Thus my results probably represent an upper-bound on performance of the current design. A further longitudinal study is needed to investigate how fast other experts could write using Parakeet.

5.7.1.2 User's Task Domain

Originally, I had hoped to have users transcribe email- or SMS-like messages. But testing showed that recognition was too hard in these domains. This probably stemmed from the lack of appropriate amounts of data to train language models in the email or SMS domains. It may be possible to address this issue either by finding appropriately large training corpora, or by using existing smaller corpora to adapt a well-estimated newswire language model.

While having users transcribe sentences is the typical task in text entry studies, having users compose novel messages would be more realistic. In particular, novice users may find it challenging to fluently dictate their own compositions. Disfluent speech could make recognition even more challenging.

5.7.1.3 Recognition Delays

Due to severely limited computational resources, there were recognition delays in the order of tens of seconds in my novice user trial (mean = 22 s, sd = 14 s, figure 5.21). Overall, including recording time and lattice post-processing time, the recognizer performed at 5 × real-time. Some (pathological) utterances took up to a minute for the system to recognize. These delays could have been reduced at the expense of an increase in recognition errors, but I felt having a realistic error rate was more important for the purposes of my study.

Despite the long recognition relays, my participants' mean (corrected) entry rate of 13 wpm walking outdoors is still about as fast as the text entry rate users obtain when sitting down and typing using T9 predictive text after several sessions [162].

Figure 5.22 shows the possible improvements in writing speed with varying reductions in recognition delay. While in practice, there will always be some

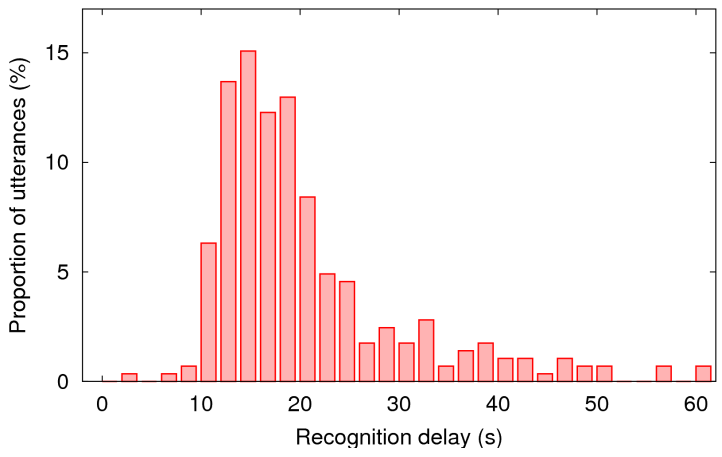


Figure 5.21: Recognition delays experienced by novice users (both indoors and outdoors).

recognition delay, faster devices and recognizers should allow a good part of these gains to be realized.

To show the potential on a much faster device, I tested the participants' audio on a 3 GHz desktop computer. The recognizer used the same acoustic models and configuration as in the user trials. I simply streamed the audio in real-time from a file rather than from a microphone. The desktop computer completed recognition after the last audio was received with a delay of only about half a second. Essentially the desktop computer was doing the speech decoding as fast as the audio arrived. The short delay at the end reflects post-processing operations that required the full recognition result.

A possible problem with the conjectured improvements based on faster recognition is that changes in the length and variability of recognition delays could result in different human performance characteristics. For example, perhaps users are using the delays to mentally rest or attend to their physical environment. A rapid interface with no delays might be too stressful and actually penalize performance. On the other hand, short delays might allow users to better remember what they are writing and thus benefit performance. Further research is needed to better understand the impact that delays have on human performance using a mobile interface.

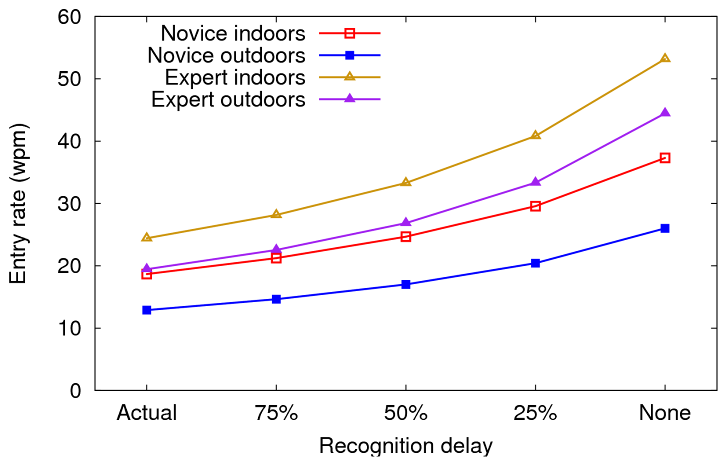


Figure 5.22: Potential text entry rates if recognition delays had been shorter.

5.7.1.4 Microphone Used

The Bluetooth microphone I used for my experiments was large and bulky. I chose this microphone in order to give the speech recognizer a good signal from a close-talking boom microphone. Further work is needed to see how more typical, compact headsets would perform.

5.7.1.5 Environmental Factors

My novice trials took place during a period of windy weather. Using data from a weather station on a nearby building, the average wind speed during my user's outdoor condition was 13 knots, gusting to 28 knots. At this location, the typical average wind speed is 4 knots, gusting to 12 knots. I believe the strong wind caused a higher than normal level of recognition errors in the outdoor condition. During the experiment, participants themselves noted how recognition errors seemed to increase when they spoke a sentence while on a windier section of the course.

Currently Parakeet's recognizer has no explicit noise robustness features. Such features would be helpful before testing Parakeet in more challenging acoustic environments (e.g. walking around a busy town center).

In addition, one trial took place on a sunny day and that participant had difficulty seeing the screen over parts of the course. A mobile device that is more readable in direct sunlight would make outdoor trials easier.

5.7.2 Design Implications

5.7.2.1 Review Screen

A large proportion of the sentences were recognized as completely correct. Despite this, participants were forced to enter the correction interface. Participants then had to scroll right in order to verify that the entire utterance was correct. It may be advantageous to first show a simple screen reviewing the entire recognition. The user could then either accept the sentence, or press on the first error to start correction at that location.

5.7.2.2 Easy Fallback

I noticed that for some utterances, the speech recognizer gave results with so many errors as to make correction an exercise in deleting everything and typing the entire sentence. Parakeet should better support this circumstance, allowing users to fallback to keyboard-only entry without having to explicitly erase each word first.

5.7.2.3 High Contrast

I noticed that participants sometimes found it hard to read the screen while outdoors because of glare. The user interface could benefit from a redesign that puts more emphasis on high contrast.

5.7.2.4 More Efficient Use of Screen Real Estate

I found that participants sometimes had trouble with target selection despite the fact that I designed buttons to be large. Buttons, particularly in the word confusion network display, could benefit from being larger. Given that participants rarely used the lower alternative word rows in the confusion network, it may be worthwhile removing a row or two to provide additional space for larger buttons.

5.7.2.5 Improved Speech Recognition

Speech recognition delays accounted for about 50% of users' entry times. As mobile devices get faster, this recognition delay will be reduced significantly. Delays might also be reduced by using network or distributed speech recognition [141]. In distributed or network speech recognition, the mobile device offloads most or all of the speech recognition work to a more powerful computer. I will use such an approach with Parakeet in the user study described in chapter 6, section 6.9. Recognition speed will have a very large impact on the practical entry rates achievable by continuous speech recognition on a mobile device. Improvements in recognition accuracy will also clearly be beneficial.

5.7.2.6 Device Improvements

The N800's battery is specified to last 3.5 hours while "browsing". I found that while using Parakeet, the battery lasted a substantially shorter amount of time (around 45 minutes). Power consumption and battery life would need to improve to make mobile speech recognition practical for sustained text entry.

The N800 can only record audio at a sampling rate of 8 kHz. I found this degraded recognition significantly compared to 16 kHz audio. Mobile devices which are intended to use speech recognition would benefit from wideband audio capture. In my informal tests, there was minimal additional computational cost recognizing wideband audio.

The N800 has 128MB of main memory (with additional swap space on a removable memory card). By using a compact acoustic and language model, I was able to keep Parakeet's memory demands to around 100MB. But a more real-world recognition task would require a larger language model. A device with more memory would clearly be advantageous. In addition, there are probably ways to reduce Parakeet's memory footprint. For example, Olsen et al. [113] implemented a Chinese Mandarin recognizer on a N800 using only 10MB of memory.

5.8 Related Work

Ogata and Goto [112] also used a word confusion network as a basis for a speech correction interface. Their system was tested on a desktop and benchmarked against conventional speech correction using a mouse and keyboard. They found that for entering Japanese text, the confusion network interface shortened entry time by 31%.

In relation to their work, my work incorporates several novel aspects:

- **Word candidate order** – They ordered all word candidates (including delete buttons) strictly by probability. I changed this so all delete buttons were in the bottom row. This was done for consistency and also to allow contiguous errors to be deleted in a single swipe.
- **Copying** – I added the ability to copy words between clusters.
- **Keyboard fallback** – I provided a fallback correction mechanism based on a predictive software keyboard.
- **Mobile design** – I designed and tested my interface on a mobile touch-screen device both indoors while seated and outdoors while walking.

Kurihara et al. [91] built upon the correction interface in [112]. In [91], an instructor's handwriting was combined with speech recognition results to provide a confusion network correction interface. Similar to my interface, their system also supports crossing-based selection in the confusion network. They evaluated their interface by having participants give or receive fake classroom lectures. Their system supported the fake instructors by automatically generating 22–70% of their pen strokes. Unlike Parakeet, their system required some use of handwriting recognition in order to select from the confusion network. Their goal was also very different. They wanted to allow quick output of a portion of the instructor's speech in order to aid student note taking. This is different from the verbatim output needed by people dictating to Parakeet.

Huggins-Daines and Rudnicky [75] demonstrated a touch-screen error correction technique for speech recognition. It allowed users to select a region in the 1-best result. Users could expand or contract the selected region using a touch interface. The user then selected an alternative hypothesis appearing in the time vicinity of the selected region. Unlike Parakeet, this interface initially only displayed the 1-best result and did not provide a fallback method of text entry. No user trial was reported.

Karpov et al. [83] tested a dictation interface on a Nokia mobile phone. Their system used isolated-word recognition (users had to pause after every word). They tested users' speed at entering short messages. The study was conducted indoors and messages had to be completely corrected. Corrections were done by selecting from a list of alternative words, repeating recognition, or switching to keypad input. Their users were able to write at 11 wpm using the isolated-word, speaker-dependent speech recognition.

Shuhm et al. [139] used handwriting to correct recognition errors. Their interface ran on a desktop and used the research recognizer JANUS. Novices had a (corrected) entry rate of 5 wpm using pen correction and 7 wpm using mouse and keyboard correction. The participants' initial utterances had a WER of 25%. For comparison, at a similar WER of 26%, my novices wrote at 13 wpm while walking around outdoors.

Several studies have simulated mobile speech recognition on a desktop computer. Fischer et al. [51] tested multitap and stylus soft keyboard correction of text entered via speech. Their interface ran on a desktop computer and used IBM ViaVoice for recognition. To simulate mobile use, they added a 10 second delay to all speech recognition events. Novices had a mean text entry rate of 22 wpm with multitap and 24 wpm using the soft keyboard. These entry rates are the best mean of any of the 4 different target texts used. Fischer et al. don't report their before correction WER. For comparison, my novices averaged 18 wpm indoors with a recognition delay of 18 seconds at a WER of 16%.

In another study simulating mobile recognition on a desktop, Cox and Walton [36] compared user performance writing short messages using multitap, predictive text, and speech. Participants used a desktop computer with an emulated mobile phone interface. Speech recognition used Nuance Dragon NaturallySpeaking 7. Participants were fastest using keys to navigate to the message entry interface and then using speech for text entry. Speech was the fastest (51 *uncorrected* wpm), followed by predictive text (19 *uncorrected* wpm), and finally multitap (13 *uncorrected* wpm).

Price et al. [120] investigated the effect that motion has on speech recognition accuracy. They had participants use a desktop commercial recognizer (IBM ViaVoice) both while seated and while walking on a treadmill. Participants were asked to compose short responses to questions. Participants did not perform correction and could not see their recognition result. Price et al. suggest adapting to a speaker's voice while walking may improve recognition both seated and walking. In my testing, I found no advantage to doing this (see section 5.5.6 for details).

5.9 Conclusions

In this chapter, I presented Parakeet: a touch-screen interface for continuous speech recognition on mobile devices. To my knowledge, this is the first exploration of text entry via continuous speech recognition in a truly mobile environment. My design of Parakeet was guided from two directions. First, I took advantage of empirical and qualitative findings in the HCI literature. Second, wherever possible, I adopted an engineering-driven design where I optimized the user interface based on the system's predicted behavior on empirical data. In Parakeet, I introduced several novel user interface enhancements. For example, I allowed words to be copied between confusion network clusters. I allowed easy replacement of a word with one of its morphological variants. I also added a fallback entry method via a predictive software keyboard.

The design of Parakeet was validated by a user study. I let participants use Parakeet both seated indoors and while walking outdoors. To my knowledge, no speech recognition text entry method has been tested with participants actually walking around. Among other things, the user study confirmed that word confusion networks were a useful representation for users. When the intended sentence was in the word confusion network, users were able to find and select it 96% of the time. I also showed that participants used the Parakeet crossing interface about 10% of the time, demonstrating that crossing was a useful complementary feature. Lastly, I made some practical design recommendations based on lessons learned in my user study.

My novice and expert user studies demonstrated that speech may be a surprisingly competitive mobile text entry method. Including the time it took to perform corrections, my novices were able to write at 18 wpm indoors and 13 wpm while walking outdoors. As an expert user, I was able to write at 24 wpm indoors and 20 wpm while walking outdoors. These rates are comparable to the 16 wpm users achieved with T9 while seated after 15 indoor sessions [162].

However, given the large recognition delays experienced in my study, users may have been faster simply typing on the software keyboard. For comparison, in [103], users wrote using a software keyboard at 28 wpm initially and 40 wpm after 400 minutes of practice. I believe that Parakeet could achieve comparable or faster entry rates if recognition delays were reduced. Limiting delays, novices in my study could have written at 37 wpm and my expert at 53 wpm. This demonstrates the potential a well designed interface like Parakeet has for making speech a very fast method for entering text while mobile.

Chapter 6

Open Vocabulary Recognition for Web Search

6.1 Overview

Searching the web by typing queries into a search engine like *Google* is a common activity. But typing queries using a keyboard can be difficult on a mobile device or for people with disabilities. In this chapter, I investigate how to enter web search queries using speech recognition.

Recognition of spoken search queries (henceforth search queries) is challenging for a number of reasons. First, search queries are not like normal English sentences. They tend to be short, have a flexible ordering of words, and draw words from a diverse and large vocabulary. This makes it challenging to construct good language models and pronunciation dictionaries for use in recognizing search queries. Second, constructing good models and dictionaries is hindered by the lack of web-search specific training materials. Third, web search is a moving target with new words and phrases appearing all the time. A recognizer relying on a fixed vocabulary would need frequent updating. Finally, given all the challenges, recognition errors are bound to occur. I argue that a good correction interface is required to help users complete their search task efficiently.

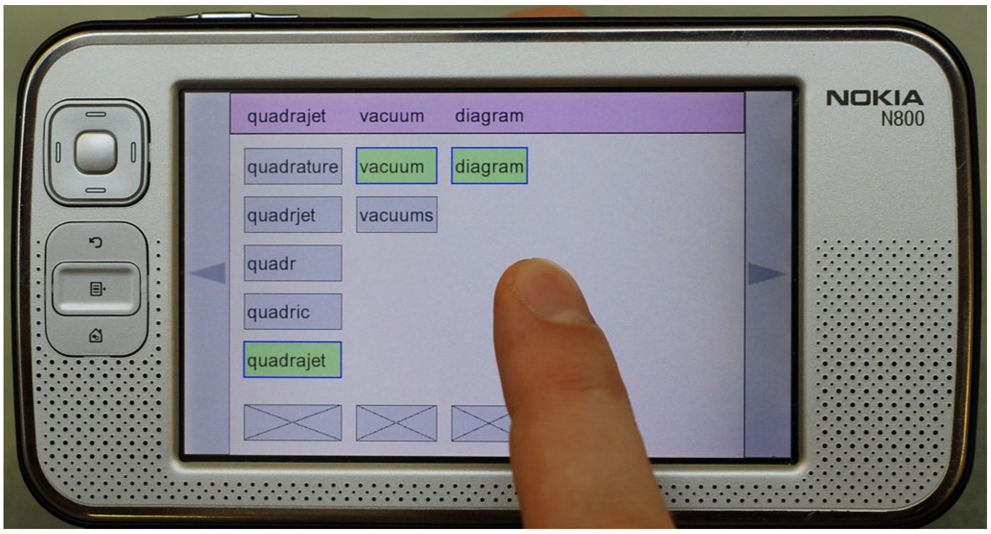


Figure 6.1: The voice web search correction interface. The user has spoken the query "quadrajet vacuum diagram". The query contains the out-of-vocabulary word "quadrajet". In the interface shown, errors are corrected by selecting alternative words from each column. In the first column, the recognizer has proposed the novel words "quadrature", "quadrjet", "quadr", "quadric" and "quadrajet".

I will describe the techniques and resources I used to address these challenges. I will demonstrate their utility in a system that allows users to speak and correct search queries on a mobile device. The system recognizes novel words and supports correction via a simple touch interface (figure 6.1).

I will use search queries obtained from typed web searches. I did this for practical reasons. At least currently, typed search query data is easier to obtain and much more plentiful. While there may be interesting differences between typed and spoken search queries, I do not investigate these differences here.

This chapter is structured as follows. First, I discuss why large, search-specific vocabularies are needed. I show that existing pronunciation dictionaries are insufficient to cover the vocabulary indicative of search queries. Second, I provide an overview of the joint multigram model. This model forms the basis for handling large vocabularies by allowing phone sequences to be inferred from letter sequences. I detail experiments that show how to make the joint multigram model perform well. I also investigate several new techniques for improving the model's performance. Third, I describe how the joint multigram model can be used to recognize novel words never seen in the training data. I extend this method to allow novel words to appear alongside their in-vocabulary competitors in a word confusion network [104]. Fourth, I describe a corpus of search queries I collected to study the problem of spoken search queries. Using the corpus, I build a system that can recognize spoken queries effectively. Finally, I present results of a formative user study in which participants spoke and corrected search queries on a mobile device while walking.

6.1.1 Publication Note

The work presented in this chapter was published in part in the papers "Combining Open Vocabulary Recognition and Word Confusion Networks" at the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP 2008) [149] and "Recognition and Correction of Voice Web Search Queries" at the International Conference on Spoken Language Processing (ICSLP 2009) [154]. Sections 6.7, 6.8, and 6.9 were joint work with Per Ola Kristensson.

6.2 The Large Vocabulary Problem

One way to handle the diverse and large vocabulary of search queries is to increase the size of the recognizer's fixed vocabulary. In the simplest case, words are added from sources that have pronunciations for each word. But such sources are limited in size and may not match the types of words used in search queries. For example, the CMU pronouncing dictionary [25] has 125K English words, but lacks many common search terms such as "ebay" and "firefox".

Of course words can be added from sources besides a pronunciation dictionary. For example, words can be added from a collection of past search queries or from words appearing in Wikipedia articles. So what effect does allowing words without pronunciations have on a vocabulary's ability to cover words seen in search queries? To find out, I collected a set of 3K queries (9.5K words) from a

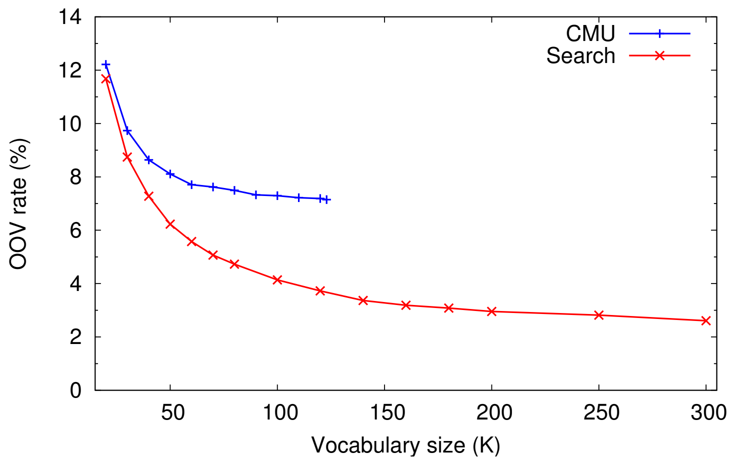


Figure 6.2: Percent of out-of-vocabulary (OOV) words in a test set of search queries using two different vocabulary sources. *CMU* used only words from the CMU dictionary. *Search* used words that appeared in a corpus of search queries.

web search engine [1]. While users had typed these queries, the queries provide an approximation of the sorts of things a user might say to a spoken web search interface. I proofread the queries to correct obvious typos. I also removed garbage and expanded abbreviations.

I built vocabularies of varying size using the most frequent words in a corpus of search queries (described in section 6.7). I restricted one set of vocabularies to use only words in the CMU dictionary (denoted CMU) while the other set could use any word (denoted Search). As shown in figure 6.2, using the Search vocabularies substantially reduced the OOV rate from 7.1% to 2.6%.

However, there are two problems with using the Search vocabularies. First, pronunciations will be needed for many of the words. For example, at a vocabulary size of 100K, 51% of words from the search query corpus had no pronunciation in the CMU dictionary. I address this first problem in sections 6.3 and 6.4 where I describe how I automatically generated pronunciations for words not in the dictionary.

The second problem is that even if a large vocabulary is used, there is still a remaining number of OOV words. These OOV words might be caused by a mismatch between the training data and the user's queries or by new vocabulary entering the language. I address this second problem in sections 6.5 where I describe how I allowed recognition of novel words that have never been seen before.

6.3 Letter-to-Phone Conversion

In this section, I describe the joint multigram model. This model serves as the basis for expanding the recognizer's vocabulary size by inferring a word's phone sequence from its letter sequence. It will also serve as the basis for recognizing novel words that are outside the recognizer's normal vocabulary.

The joint multigram model was developed by Deligne et al. [43; 44]. The idea is to learn a set of likely pairings between the written, grapheme unit sequences of a language and its spoken, phoneme unit sequences. For example, in English some common pairings include:

$$\begin{array}{l} \textit{Grapheme} \\ \texttt{Phonemes} \end{array} \left(\begin{array}{l} \textit{ing} \\ \texttt{ih ng} \end{array} \right) \left(\begin{array}{l} \textit{ation} \\ \texttt{ey sh ah n} \end{array} \right) \left(\begin{array}{l} \textit{sch} \\ \texttt{sh} \end{array} \right)$$

As in Bisani and Ney [16], I refer to these pairings as graphones. Graphones can be automatically learned from a pronunciation dictionary consisting of words and their corresponding phone sequences. In the remainder of this section, I describe the important details of the joint multigram model. A full derivation of the model appears in appendix C.

6.3.1 Graphones and Model Intuition

A graphone G_i is made up of a co-sequence of letters and phones:

$$G_i = \left(\begin{array}{c} \ell_1, \ell_2, ..., \ell_j \\ \rho_1, \rho_2, ..., \rho_k \end{array} \right).$$

Let $\ell(G_i)$ denote the letter sequence $\ell_1, \ell_2, ..., \ell_j$ of a particular graphone G_i and let $\rho(G_i)$ denote the phone sequence $\rho_1, \rho_2, ..., \rho_k$ of G_i.

Graphones are restricted to some minimum and maximum number of letters and phones. For English, I used the same range for both letters and phones. I denote a model by its range. For example, a 1–2 model consists of graphones having 1 or 2 letters and 1 or 2 phones. It is also possible to have models in which graphones have 0 letters or 0 phones (but not both).

An entry in a pronunciation dictionary consists of a letter sequence and its corresponding phone sequence. The model assumes dictionary entries do not have any markings denoting which of a word's letters are associated with which of its phones.

The joint multigram model is a generative model. To create a dictionary entry, graphones are assumed to be drawn independently from an inventory of all possible graphones. A dictionary entry is created by concatenating the letter and phones of the drawn graphones. The concatenation is continued until a graphone with the end-of-word symbol "•" is drawn. Before training, this end-of-word (EOW) symbol is implicitly added to the end of every training entry's letter sequence.

Given a graphone inventory, a particular dictionary word may have multiple possible segmentations. For example, under a 1–2 model, the entry for "cat" has the following 5 segmentations:

$$\begin{pmatrix} ca \\ \mathtt{k\ ae} \end{pmatrix}\begin{pmatrix} t\,\bullet \\ \mathtt{t} \end{pmatrix}, \begin{pmatrix} ca \\ \mathtt{k} \end{pmatrix}\begin{pmatrix} t\,\bullet \\ \mathtt{ae\ t} \end{pmatrix}, \begin{pmatrix} ca \\ \mathtt{k} \end{pmatrix}\begin{pmatrix} t \\ \mathtt{ae} \end{pmatrix}\begin{pmatrix} \bullet \\ \mathtt{t} \end{pmatrix},$$

$$\begin{pmatrix} c \\ \mathtt{k} \end{pmatrix}\begin{pmatrix} at \\ \mathtt{ae} \end{pmatrix}\begin{pmatrix} \bullet \\ \mathtt{t} \end{pmatrix}, \begin{pmatrix} c \\ \mathtt{k} \end{pmatrix}\begin{pmatrix} a \\ \mathtt{ae} \end{pmatrix}\begin{pmatrix} t\,\bullet \\ \mathtt{t} \end{pmatrix} \quad .$$

Informally, the job of the joint multigram model is to look through the possible segmentations of all words in the dictionary and decide how "useful" each graphone is for generating the dictionary. Probabilities are then assigned to each graphone according to its "usefulness". Given the inventory of graphones and their associated probabilities, it is possible to infer a phone sequence given a letter sequence (and vice-versa).

Table 6.1 gives some examples of the top graphones learned by a 1–5 model. As can be seen, the model has learned some common English suffixes and prefixes.

Ranks	Graphones				
1–5	$\begin{pmatrix} e\ r\ \bullet \\ \texttt{er} \end{pmatrix}$	$\begin{pmatrix} i\ n\ g\ \bullet \\ \texttt{ih ng} \end{pmatrix}$	$\begin{pmatrix} s \\ \texttt{s} \end{pmatrix}$	$\begin{pmatrix} s\ \bullet \\ \texttt{z} \end{pmatrix}$	$\begin{pmatrix} t \\ \texttt{t} \end{pmatrix}$
6–10	$\begin{pmatrix} m \\ \texttt{m} \end{pmatrix}$	$\begin{pmatrix} n \\ \texttt{n} \end{pmatrix}$	$\begin{pmatrix} b \\ \texttt{b} \end{pmatrix}$	$\begin{pmatrix} '\ s\ \bullet \\ \texttt{z} \end{pmatrix}$	$\begin{pmatrix} d \\ \texttt{d} \end{pmatrix}$
11–15	$\begin{pmatrix} y\ \bullet \\ \texttt{iy} \end{pmatrix}$	$\begin{pmatrix} l \\ \texttt{l} \end{pmatrix}$	$\begin{pmatrix} p \\ \texttt{p} \end{pmatrix}$	$\begin{pmatrix} t\ \bullet \\ \texttt{t} \end{pmatrix}$	$\begin{pmatrix} f \\ \texttt{f} \end{pmatrix}$
16–20	$\begin{pmatrix} e\ r \\ \texttt{er} \end{pmatrix}$	$\begin{pmatrix} m\ a\ n\ \bullet \\ \texttt{m ah n} \end{pmatrix}$	$\begin{pmatrix} e\ r\ s\ \bullet \\ \texttt{er z} \end{pmatrix}$	$\begin{pmatrix} a\ \bullet \\ \texttt{ah} \end{pmatrix}$	$\begin{pmatrix} r \\ \texttt{r} \end{pmatrix}$
21–25	$\begin{pmatrix} i\ n \\ \texttt{ih n} \end{pmatrix}$	$\begin{pmatrix} s\ \bullet \\ \texttt{s} \end{pmatrix}$	$\begin{pmatrix} g \\ \texttt{g} \end{pmatrix}$	$\begin{pmatrix} c \\ \texttt{k} \end{pmatrix}$	$\begin{pmatrix} h \\ \texttt{hh} \end{pmatrix}$
26–30	$\begin{pmatrix} r\ e \\ \texttt{r iy} \end{pmatrix}$	$\begin{pmatrix} e\ d\ \bullet \\ \texttt{d} \end{pmatrix}$	$\begin{pmatrix} t\ i\ o\ n\ \bullet \\ \texttt{sh ah n} \end{pmatrix}$	$\begin{pmatrix} e\ n\ \bullet \\ \texttt{ah n} \end{pmatrix}$	$\begin{pmatrix} o\ \bullet \\ \texttt{ow} \end{pmatrix}$

Table 6.1: The top-30 graphones learned by a 1–5 model. The symbol • denotes the end-of-word symbol.

6.3.2 Parameters, Training Data, and Segmentations

A graphone inventory consists of all possible graphones given the letters and phones of a language (subject to the model's length range). Each graphone G_i in the inventory has an associated unigram probability $P(G_i)$. If there are I graphones in the inventory, the parameters $\boldsymbol{\theta}$ of the joint multigram model are:

$$\boldsymbol{\theta} = \{G_1, ..., G_I, P(G_1), ..., P(G_I)\}\,. \tag{6.1}$$

For conceptual clarity, the learning process is viewed as the search for optimal probabilities using the set of all possible graphones. The size of this set is exponential in the length-range of the model. During initialization, a practical implementation represents only graphones that were actually seen in the training data (see section 6.4.3).

The model's training data $\boldsymbol{D}$ consists of a pronunciation dictionary with N entries:

$$\boldsymbol{D} = \{\boldsymbol{D}_1, \boldsymbol{D}_2, ..., \boldsymbol{D}_N\}\,. \tag{6.2}$$

The n^{th} entry in the dictionary has a letter sequence denoted $\boldsymbol{X}_n$ and a phone sequence denoted $\boldsymbol{Y}_n$:

$$\boldsymbol{D}_n = \begin{pmatrix} \boldsymbol{X}_n \\ \boldsymbol{Y}_n \end{pmatrix}\,. \tag{6.3}$$

Given a particular graphone inventory, dictionary entry $\boldsymbol{D}_n$ could have been generated by 0 or more possible graphone sequences. Let $\boldsymbol{S}_n$ denote the vector of the $|\boldsymbol{S}_n|$ possible segmentations for dictionary entry $\boldsymbol{D}_n$:

$$\boldsymbol{S}_n = \{\boldsymbol{S}_{n1}, \boldsymbol{S}_{n2} ..., \boldsymbol{S}_{n|\boldsymbol{S}_n|}\}\,. \tag{6.4}$$

$\boldsymbol{S}_{ns}$ is the s^{th} segmentation of the n^{th} dictionary entry. $\boldsymbol{S}_{ns}$ is a vector of the $|S_{ns}|$ graphone indices that make up the segmentation:

$$\boldsymbol{S}_{ns} = \{S_{ns1}, S_{ns2}, ..., S_{ns|\boldsymbol{S}_{ns}|}\} \tag{6.5}$$

In order for $\boldsymbol{S}_{ns}$ to be a valid segmentation, the graphone indexes in $\boldsymbol{S}_{ns}$ must exactly produce the dictionary entry's letter and phone sequences:

$$\boldsymbol{X}_n = \ell\left(G_{S_{ns1}}\right) \sqcup \ell\left(G_{S_{ns2}}\right) \sqcup \dots \sqcup \ell\left(G_{S_{ns|\boldsymbol{S}_{ns}|}}\right) \tag{6.6}$$

$$\boldsymbol{Y}_n = \rho\left(G_{s_{ns1}}\right) \sqcup \rho\left(G_{S_{ns2}}\right) \sqcup \dots \sqcup \rho\left(G_{S_{ns|\boldsymbol{S}_{ns}|}}\right) \tag{6.7}$$

where $\sqcup$ denotes concatenation of letter or phone sequences.

6.3.3 Model Assumptions

Graphones in a sequence are assumed to be independent:

$$P(\boldsymbol{S}_{ns}|\boldsymbol{\theta}) = \prod_{g=1}^{|\boldsymbol{S}_{ns}|} P(G_{S_{nsg}}|\boldsymbol{\theta})\,. \tag{6.8}$$

This independence assumption will be loosened later by the addition of an n-gram language model (see section 6.3.7). The probability of a dictionary entry is the sum over all possible graphone sequences that exactly generate an entry's letters and phones:

$$P(\boldsymbol{D}_n|\boldsymbol{\theta}) = \sum_{s=1}^{|\boldsymbol{S}_n|} P(\boldsymbol{S}_{ns}|\boldsymbol{\theta})\,. \tag{6.9}$$

Training examples from the dictionary are assumed to be independent. Thus the overall probability of the training data is:

$$P(\boldsymbol{D}|\boldsymbol{\theta}) = \prod_{n=1}^{N} \sum_{s=1}^{|\boldsymbol{S}_n|} \prod_{g=1}^{|\boldsymbol{S}_{ns}|} P(G_{S_{nsg}}|\boldsymbol{\theta})\,. \tag{6.10}$$

6.3.4 Reestimation Formula

A local optimum for the model parameters $\boldsymbol{\theta}$ can be found using expectation maximization (EM). Here I provide just the final result; for a complete derivation see appendix C. The reestimation uses the *responsibility* $\gamma(z_{ns})$ which is the degree to which the s^{th} segmentation explains the n^{th} training example:

$$\gamma(z_{ns}) = \frac{\prod_{g=1}^{|\boldsymbol{S}_{ns}|} P(G_{S_{nsg}}|\boldsymbol{\theta})}{\sum_{s=1}^{|\boldsymbol{S}_n|} \prod_{g=1}^{|\boldsymbol{S}_{ns}|} P(G_{S_{nsg}})|\boldsymbol{\theta})}\,. \tag{6.11}$$

The responsibility is how likely a particular segmentation is in comparison with all other segmentations for that dictionary entry. Using the responsibility, the reestimation formula at time step $(t+1)$ is:

$$P^{(t+1)}(G_m) = \frac{\sum_{n=1}^{N} \sum_{s=1}^{|S_n|} \gamma^{(t)}(z_{ns}) C(G_m|\boldsymbol{S}_{ns})}{\sum_{n=1}^{N} \sum_{s=1}^{|S_n|} \gamma^{(t)}(z_{ns}) C(\boldsymbol{S}_{ns})} \tag{6.12}$$

where $C(G_m|\boldsymbol{S}_{ns})$ is how many times graphone G_m appears in segmentation $\boldsymbol{S}_{ns}$ and $C(\boldsymbol{S}_{ns})$ is how many graphones are in that segmentation.

Thus at each training iteration, a particular graphone's probability is updated based on how often that graphone appears in all segmentations of the training data. These appearances are weighted according to how likely each particular segmentation was.

6.3.5 Viterbi Training

The reestimation formula (6.12) requires a summation over all possible graphone sequences for each training example. A cheap and cheerful form of training uses just the most likely graphone sequence for each dictionary entry. Let $\boldsymbol{S}_{n*}^{(t)}$ be the most likely graphone sequence for $\boldsymbol{D}_n$ at time step t:

$$\boldsymbol{S}_{n*}^{(t)} = \text{argmax}_{\boldsymbol{S}_{ns} \in \boldsymbol{S}_n} P(\boldsymbol{S}_{ns}|\boldsymbol{\theta}^{(t)})\,. \tag{6.13}$$

The responsibility $\gamma(z_{ns})$ is set to 1 when $\boldsymbol{S}_{ns} = \boldsymbol{S}_{n*}^{(t)}$ and 0 otherwise. The reestimation formula (6.12) for Viterbi training is defined as:

$$P^{(t+1)}(G_m) = \frac{\sum_{n=1}^{N} C(G_m|\boldsymbol{S}_{n*}^{(t)})}{\sum_{n=1}^{N} C(\boldsymbol{S}_{n*}^{(t)})}\,. \tag{6.14}$$

In Viterbi training, the probability of the data is improved by reestimating the probabilities using the relative frequency of how often each graphone appears in the most-likely segmentation of the training examples using the previous iteration's parameters.

6.3.6 EM Training

The reestimation formula (6.12) can efficiently be calculated using the forward–backward algorithm. By first introducing a forward variable α and a backwards variable β, the summation over all possible segmentations of each dictionary entry can be replaced by a sweep over all letter and phone positions. Here I provide the final result; for a full derivation see appendix C.2.1.

The forward variable $\alpha_n(x,y)$ accounts for the probability from the start up to and *including* the x^{th} letter and y^{th} phone of the n^{th} dictionary entry:

$$\alpha_n(x,y) = P\left((\boldsymbol{X}_n)_1^x, (\boldsymbol{Y}_n)_1^y \mid \boldsymbol{\theta}\right) . \tag{6.15}$$

Similarly, the backwards variable $\beta_n(x,y)$ accounts for the probability of everything *after* the x^{th} letter and y^{th} phone of the n^{th} dictionary entry:

$$\beta_n(x,y) = P\left((\boldsymbol{X}_n)_{x+1}^{|\boldsymbol{X}_n|}, (\boldsymbol{Y}_n)_{y+1}^{|\boldsymbol{Y}_n|} \mid \boldsymbol{\theta}\right) . \tag{6.16}$$

The reestimation formula used during forward-backward training is:

$$P^{(t+1)}(G_m) = \frac{\sum_{n=1}^{N} \frac{1}{\beta_n^{(t)}(0,0)} \sum_{x=1}^{|\boldsymbol{X}_n|} \sum_{y=1}^{|\boldsymbol{Y}_n|} \alpha_n^{(t)}(x-q, y-r) P^{(t)}(G_m) \beta_n^{(t)}(x,y) \delta_{x,y,m}}{\sum_{n=1}^{N} \frac{1}{\beta_n^{(t)}(0,0)} \sum_{x=1}^{|\boldsymbol{X}_n|} \sum_{y=1}^{|\boldsymbol{Y}_n|} \alpha_n^{(t)}(x,y) \beta_n^{(t)}(x,y)} \tag{6.17}$$

where graphone G_m has q letters and r phones and $\delta_{x,y,m}$ is an indicator variable which is one if the letters and phones immediately preceding position (x,y) match graphone G_m.

6.3.7 Modeling Graphone Dependency

The joint multigram model makes the assumption that graphones in a sequence are independent (6.8). To model dependency between graphones, the graphones' unigram probabilities are used to find the most likely segmentation of each entry in the training dictionary using (6.13). These segmentations are then used as training data for a standard n-gram language model in which graphones in a sequence are dependent on a prior window of $M - 1$ graphones:

$$P(\boldsymbol{S}_{ns}|\boldsymbol{\theta}) = \prod_{g=1}^{|\boldsymbol{S}_{ns}|} P(G_{S_{nsg}}|G_{S_{ns(g-1)}}, ..., G_{S_{ns(g-M+1)}}, \boldsymbol{\theta}) \,. \tag{6.18}$$

The training of a graphone n-gram language model can be done using a standard toolkit such as SRILM [138]. I found that training a well-performing graphone language model was not as simple as training a standard word language model. In section 6.4, I highlight the important details about how to train a well-performing graphone language model.

6.3.7.1 Search Using a Graphone Language Model

Using the graphone language model, a word's most likely phones can be found from its letters (and vice-versa). This is done by finding the most likely graphone sequence consistent with the known symbols. I implemented this search using the token passing paradigm [172]. In token passing, virtual tokens explore the search space of possible hypotheses. Each token keeps tracks of its probability, its location in the letter and phone sequence, and its history of previous graphones.

Using token passing, either a 1-best or n-best search is possible. For 1-best search, I found it was computationally feasible to do an exact search. In order to be tractable, care must be taken to prune tokens during an exact search. This is because a graphone inventory may include graphones that have no letters or no phones. These graphones can be added to a search hypothesis without consuming symbols in the search's known symbol sequence. Without pruning, the search could explore segmentations involving an infinite number of graphones.

I implemented two methods of pruning tokens while maintaining an exact search. The first method was to prune a token with a probability worse than the current best result. The second method was to prune a token at a given letter and phone position if its probability is worse than a previous token to visit that same position. In the second form of pruning, the previous token and the current token must have identical language model context in order for pruning to occur. If the language model backed off, a token's language model context was taken to be the shorter, backed-off context. This last point was important in order to keep the exact search tractable when using long-span language models.

It is also possible to find the n-best most likely phone sequences given a letter sequence. This is useful if multiple pronunciation variants are desired for a word. It might also allow more accurate phone sequences to be inferred by summing over alternate segmentations that share the same phone sequence. In the case of an n-best search, I did not attempt to guarantee finding exactly the top-n hypotheses. I pruned the n-best search by limiting the number of tokens at a particular letter and phone position. I also pruned tokens that were too improbable compared to the best previous token to visit a particular letter position.

6.4 Letter-to-Phone Experiments

In this section, I describe experiments showing how I achieved state-of-the-art letter-to-phone conversion using graphones. To my knowledge, this work provides the first detailed comparison of how different aspects of graphone language model training (such as smoothing, count cutoffs, and iterative training) affect letter-to-phone performance. I also experiment with a new search strategy based on summing over multiple possible graphone segmentations.

6.4.1 Experimental Setup

My experiments used the CMU dictionary, removing words with letters other than A–Z and apostrophe. I retained multiple pronunciations per word. I randomly split the dictionary using 70% for a training set, 10% for a development test set, and 20% for an evaluation test set. I used a random split to be comparable with past work that also used a random split (e.g. [19; 56]). When I performed

	graphone model size				
LM size	0–1	0–2	0–3	1–2	1–3
2-gram	18.76	10.19	8.70	10.84	9.83
3-gram	10.22	7.67	8.40	9.03	9.59
4-gram	7.61	7.55	8.40	8.97	9.58
5-gram	7.00	7.55	8.41	8.95	9.55
6-gram	6.86	7.54	8.39	8.96	9.55
7-gram	**6.83**	7.55	8.40	8.96	9.55

Table 6.2: Phone error rate (PER) of letter-to-phone conversion varying graphone size (columns) and language model size (rows). The lowest error rate is shown in bold.

the split, I kept a word and all its pronunciation variants as a unit. I have made my particular training, development and evaluation split available so other researchers can more easily compare techniques [150].

As an evaluation metric, I used the phone error rate (PER). PER is the minimum edit distance between the recognized phone sequence and its reference. For words with multiple pronunciations, I took the PER to be the minimum PER out of all pronunciations. I report error bars using a one standard error (σ_{se}) confidence interval obtained via per-word bootstrap resampling [17]. Except where specified, the letter-to-phone results I present in this section were on my development test set.

6.4.2 Graphone Size

I tested a variety of graphone sizes using different sized language models. As shown in table 6.2, models allowing graphones with no letters or no phones (0–X models) outperformed those with a 1 letter/phone minimum. As in [18] and [28], small 0–1 graphones with long-span language models provided the best letter-to-phone accuracy. Since gains were small beyond a 6-gram language model, I chose a 6-gram language model and 0–1 graphones for use in letter-to-phone conversion.

6.4.3 Initialization and Training

Training requires an initial inventory of graphones as well as probabilities of each graphone. I tried two initialization methods:

- **Flat** – I enumerated all possible graphones by combining every possible letter with every possible phone (including a "null" letter/phone). I then set the probability of each graphone to be equal. Flat initialization was only feasible for short models such as 0–1.
- **All sequences** – I searched for all possible segmentations of dictionary entries into graphone sequences, subject to the model's graphone size restriction. A complete search of all segmentations was computationally expensive for longer words. For long words, I searched only for sequences involving graphones with a size one larger than the minimum length of the model. The initial probability of a graphone was set based on its relative frequency in the segmentations found by the search.

I used two training methods:

- **Viterbi** – Training used just the most likely graphone sequence for each training example (see section 6.3.5). During Viterbi-style training, I pruned graphones if the number of times they were used in the alignment of the dictionary fell below a minimum count.
- **EM** – Training used a summation over all possible graphone sequence for each training example (see section 6.3.6). During EM-style training, I used the "evidence trimming" approach as in [16]. In evidence trimming, graphones are pruned if their maximum conditional probability given a dictionary entry became too low.

I tested models built with both initialization methods and both training methods. As shown in table 6.3, all models performed about the same. EM training of the unigram graphone probabilities made only a small improvement to the eventual PER of the model. This is probably because the graphone unigram

Graphone training	Initialization method	PER $\pm \sigma_{se}$
Viterbi	flat	6.83 $\pm$ 0.12
Viterbi	all sequences	6.83 $\pm$ 0.12
EM	flat	6.81 $\pm$ 0.12
EM	all sequences	6.79 $\pm$ 0.12

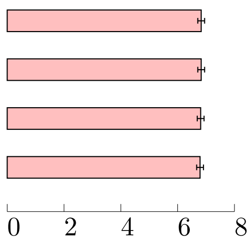


Table 6.3: Results using both initialization and both training methods to build a 6-gram 0–1 model.

probabilities learned via EM are merely a starting point. The unigrams are used to segment the dictionary in order to train a 2-gram language model, the 2-gram language model is used to train a 3-gram language model, and so on. These subsequent steps of training longer-span language models are making hard, Viterbi-like segmentation decisions. Any initial advantage offered by the EM unigram model would probably disappear in the subsequent training iterations of the longer-span language models.

6.4.4 Smoothing and Interpolation

As longer-span language models are used, sparsity in the training data can cause unreliable estimates of n-gram probabilities if they are calculated directly from the relative frequencies. For 2-gram and longer language models, smoothing of the n-gram probabilities can be done using a variety of techniques such as Good-Turing [61], Witten-Bell [14], absolute discounting [109], or Kneser-Ney [85]. Another choice is whether the model should always interpolate higher-order n-grams with lower-order n-grams. For an overview of language model smoothing techniques and interpolation, see [30].

As shown in table 6.4, the smoothing method chosen greatly influenced the accuracy of the model. I found interpolation was critical for obtaining good performance using long-span language models and small 0–1 graphones. I suspect

Smoothing method	Inter-polation	PER $\pm \sigma_{\text{se}}$
Good-Turing	no	8.19 ± 0.12
Witten–Bell	no	7.98 ± 0.13
absolute disc. of 0.9	no	11.76 ± 0.14
original Kneser–Ney	no	8.61 ± 0.13
Witten–Bell	yes	7.61 ± 0.13
absolute disc. of 0.5	yes	8.03 ± 0.13
absolute disc. of 0.8	yes	7.45 ± 0.12
absolute disc. of 0.9	yes	7.37 ± 0.12
absolute disc. of 1.0	yes	8.17 ± 0.13
original Kneser–Ney	yes	6.81 ± 0.12

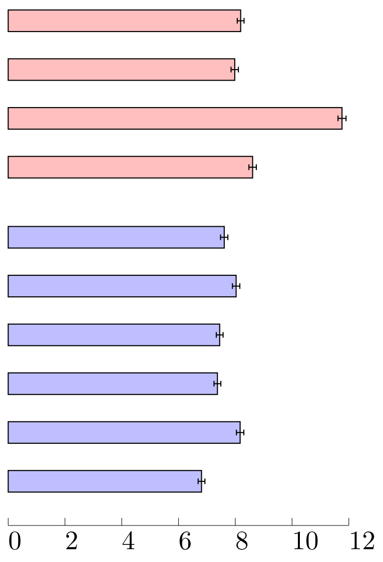


Table 6.4: PER of letter-to-phone conversion using different smoothing methods and interpolation during graphone language model training. Results are on a 6-gram 0–1 model.

this is due to the small amount of training data involved (around 100K words). The high-order n-grams probably needed to share information with the better estimated lower-order n-grams. Of all the methods, interpolated original Kneser-Ney performed the best.

6.4.5 Iterative Training

In the simplest case, a language model of the desired span (3-gram, 4-gram, etc.) is trained directly on the segmentation provided by the unigram model. I also investigated an iterative training scheme. This was similar to the scheme used by Chen to train a maximum entropy language model [28].

In iterative training, a 2-gram language model is first trained on the initial segmentation provided by the unigram model. This 2-gram language model is

Iter. LM	LM cutoffs	EOW	PER $\pm\ \sigma_{\text{se}}$
yes	0	yes	6.81 ± 0.12
no	0	yes	6.83 ± 0.12
yes	1/1/1/2/2/2	yes	7.29 ± 0.12
yes	0	no	6.61 ± 0.12

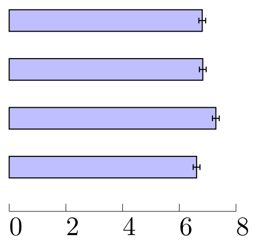


Table 6.5: Effect of changing one thing at a time from my default letter-to-phone model (first row). I changed: iterative language model training, language model count cutoffs, and use of the end-of-word (EOW) letter. Results are on a 6-gram 0–1 model.

then used to re-segment the training data, training another language model, and the process is repeated until convergence. After convergence, the span of the language model is increased by one. The new longer-span language model is then trained to convergence, and so on. To test for convergence, I used the Bayesian information criterion [127]. This criterion prefers models that explain the data well but which use fewer model parameters.

To test the benefit of the iterative scheme, I compared performance with a language model that was trained directly on the initial graphone segmentation provided by the EM-trained unigram model. The iterative scheme provide only a small (0.02% absolute) improvement in PER (row 1 versus row 2, table 6.5).

6.4.6 Count Cutoffs

Another issue in graphone language model training is how to handle n-grams that occur infrequently in the training data. Typically in a word language model, a cutoff value is used that treats low frequency events as if they had never occurred. But a word language model usually has orders of magnitude more data than is typically available when training a graphone language model.

By default, SRILM treats 1/2/3-grams occurring 1 time and any longer n-grams occurring 1–2 times as having never occurred. Given the small amount of training data involved in estimating my graphone language model, I trained my graphone language models with no count cutoffs. To show the benefit of training with no cutoffs, I compared with a 6-gram 0–1 model trained using SRILM's default cutoffs. The language model trained with the default cutoffs had a 0.48% higher PER (row 1 versus row 3, table 6.5).

6.4.7 End-of-Word Letter

To convert a sequence of graphones into likely OOV words, I added a special end-of-word (EOW) letter to the end of every training word. But for letter-to-phone conversion of a dictionary like CMU, the word boundaries are known. Thus it is not strictly necessary to include the EOW letter. I found using the EOW letter hurt performance somewhat, increasing PER by 0.20% absolute (row 1 versus row 4, table 6.5). Thus I will add the EOW letter only when the model must find word boundaries.

6.4.8 Summation Search

In letter-to-phone conversion, the best phone sequence is sought for a particular letter sequence. The exact graphone segmentation that generated a phone sequence may not be of interest. There may be in fact be more than one segmentation that leads to a particular phone sequence. This suggests an opportunity to improve letter-to-phone accuracy by summing over all segmentations.

Using an n-best search, I approximated the search for all possible graphone segmentations for a given letter sequence. I summed the probability of all segmentations that produced the same phone sequence. I then chose the phone sequence with the highest probability sum. I found the summation search with a 6-gram 0–1 model produced the same PER as a 1-best search. The n-best search also required substantially more computation. While the n-best search might be useful in circumstances when alternates to the 1-best are needed, it does not appear useful for improving 1-best accuracy. In work done in parallel with my own, Bisani and Ney [19] also found that summation search did not improve performance of their best letter-to-phone model.

6.4.9 Performance on Unseen Test Set

I tested for generalization by using my best model on unseen evaluation test data. My best model for letter-to-phone conversion was an EM-trained 0–1 graphone model. I trained the model without the EOW letter. For this final evaluation, I trained the model using both my training and development test sets (80% of the CMU dictionary). I initialized the graphone inventory with all possible graphones and set their probabilities to be equal. From the 0–1 graphones, I iteratively trained a 6-gram language model using no count cutoffs and using interpolated Kneser-Ney smoothing.

My best model had a PER of 6.51% ($\sigma_{\text{se}} = 0.08\%$) on the evaluation set. Other reported CMU letter-to-phone results are 5.88% by Bisani and Ney [19], 5.9% by Chen [28] and 7.0% by Galescu [56].

I tested the implementation released by Bisani and Ney [20] using my training, development and evaluation set. I found for a 0–1 9-gram model, their implementation was only 0.1% absolute better than my own (6.4% versus 6.5%). This was despite their use of a graphone-specific EM-training regimen which optimized the language model's smoothing parameters using Powell's method [119]. This shows that careful use of standard language modeling techniques can achieve state-of-the-art letter-to-phone performance.

6.5 Word+Graphone Language Models

As shown earlier, even a very large vocabulary using domain-specific vocabulary fails to cover a certain percentage of words. In this section, I show how to recognize novel words. This is done by training a language model on text where each OOV word has been replaced by its most likely graphone sequence. This word+graphone language model is then used for recognition, returning both words and

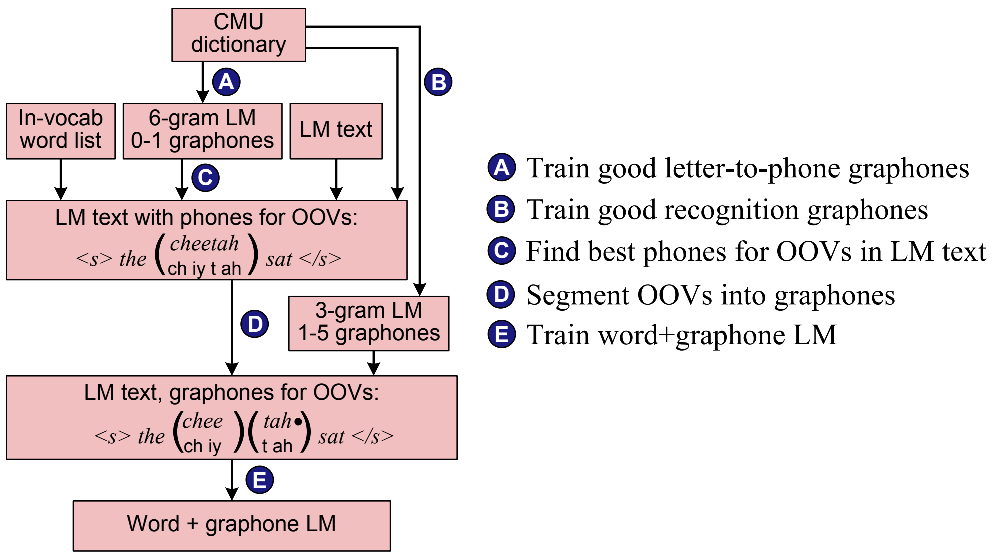

CMU dictionary
A
B
In-vocab word list
6-gram LM 0-1 graphones
LM text
C
LM text with phones for OOVs:
<s> the (cheetah / ch iy t ah) sat </s>
D
3-gram LM 1-5 graphones
LM text, graphones for OOVs:
<s> the (chee / ch iy)(tah• / t ah) sat </s>
E
Word + graphone LM
A Train good letter-to-phone graphones
B Train good recognition graphones
C Find best phones for OOVs in LM text
D Segment OOVs into graphones
E Train word+graphone LM

Figure 6.3: Overview of the process of training a word+graphone language model.

graphones. In this way, a novel OOV word can be recognized when the evidence suggests it is more appropriate than an in-vocabulary word.

6.5.1 Language Model Training

Previous research such as [18] has used a single-step process to train word+ graphone language models. In the single-step process, the graphone model used for recognition is also used to determine the phone sequence of OOV words in the language model training text. Typically, bigger graphones (such as 1–5) perform better for recognition [18]. In addition, 0–X graphones are typically not allowed as the recognizer usually requires non-null phone sequences for dictionary entries. But longer graphones are not as good at inferring the phone sequences for OOV words in the language model training text. Training on less accurate OOV phone sequences seems unlikely to be helpful in correctly recognizing OOVs.

To address this problem, I created a new two-step process utilizing two different graphone models (overview in figure 6.3). First, given an in-vocabulary list, I replaced each OOV word in the language model training text with its most likely graphone sequence. For example, consider the sentence "The cheetah sat".

If "cheetah" is not part of the in-vocabulary set of words, it will be replaced by its most likely graphone sequence:

$$\textit{The} \begin{pmatrix} c \\ \texttt{ch} \end{pmatrix} \begin{pmatrix} h \\ \end{pmatrix} \begin{pmatrix} e \\ \texttt{iy} \end{pmatrix} \begin{pmatrix} e \\ \end{pmatrix} \begin{pmatrix} t \\ \texttt{t} \end{pmatrix} \begin{pmatrix} a \\ \end{pmatrix} \begin{pmatrix} h \\ \texttt{ah} \end{pmatrix} \textit{sat.}$$

In this first step, OOV words were replaced using a graphone model tuned for letter-to-phone conversion (a 6-gram 0–1 model without the EOW letter). In cases where an OOV was in the CMU dictionary (but not in the in-vocabulary set used for recognition), I used the CMU phone sequence as a constraint when searching for the best phone sequence using the graphone model.

In the second step, I used the OOV phone sequences found with the best letter-to-phone model as a search constraint when segmenting OOVs with the graphone model used for recognition. For example, a 1–5 model with the EOW letter might re-segment the above example as:

$$\textit{The} \begin{pmatrix} \textit{chee} \\ \texttt{ch iy} \end{pmatrix} \begin{pmatrix} \textit{tah} \bullet \\ \texttt{t ah} \end{pmatrix} \textit{sat.}$$

Once the language model training text had been re-segmented using the bigger graphone model, the final word+graphone language model was trained.

6.5.2 OOV Penalties

During recognition, an empirically determined word insertion penalty is usually added to every word in the recognition hypothesis. This provides a parameter that can be adjusted to reduce insertion errors. Without any modification, a recognizer will penalize every lexical item in a word+graphone language model. This results in OOV words typically incurring multiple word insertion penalties as they usually consist of multiple graphones. No prior work has discussed what effect this has on recognition accuracy. I investigated different penalty schemes by modifying the recognizer to allow separate penalty values depending on the dictionary entry:

- **Word insertion penalty** – Added to all in-vocabulary words.
- **OOV unit penalty** – Added to all graphones.
- **OOV word penalty** – Added to all EOW graphones (in addition to incurring a OOV unit penalty).

6.5.3 Determining OOV Word Boundaries

A problem with using graphones for recognition is how to determine word boundaries. The recognizer may return several contiguous sequences of graphones, each of which represents a distinct word. For example, if the user says "The wily cheetah ran", the recognizer might return a sequence of words and graphones:

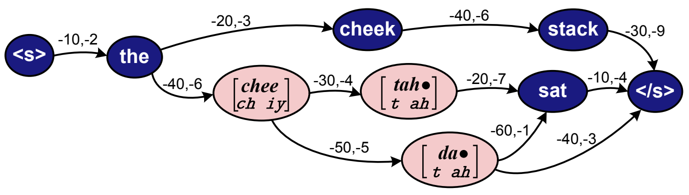
$$\mathit{The} \begin{pmatrix} \mathit{wi} \\ \mathtt{w\ ay} \end{pmatrix} \begin{pmatrix} \mathit{ly} \\ \mathtt{l\ iy} \end{pmatrix} \begin{pmatrix} \mathit{ch} \\ \mathtt{ch} \end{pmatrix} \begin{pmatrix} \mathit{ee} \\ \mathtt{iy} \end{pmatrix} \begin{pmatrix} \mathit{t} \\ \mathtt{t} \end{pmatrix} \begin{pmatrix} \mathit{ah} \\ \mathtt{ah} \end{pmatrix} \mathit{ran.}$$

In the prior work of Bisani and Ney [18], a simple heuristic of combining adjacent graphones into a single word was used. This heuristic obviously fails if a user speaks consecutive OOV words (e.g. an uncommon proper name or technical phrase).

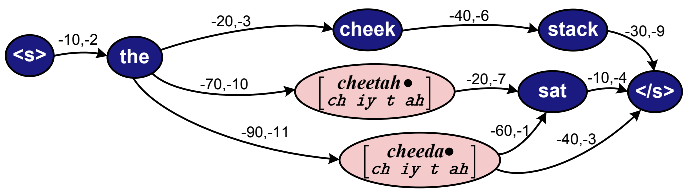
As previously described, I allowed graphone models to be trained with or without a special pseudo-letter that denotes the end of a word. This end-of-word (EOW) letter was added to the end of every OOV's letter sequence before training the word+graphone language model. At recognition time, the EOW letter can then be used to split a sequence of graphones into its constituent words.

6.5.4 Open Vocabulary Confusion Networks

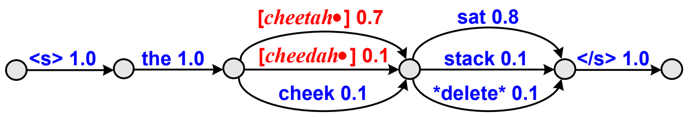
I extended the word+graphone recognition technique to allow creation of word confusion networks [104] containing both in- and out-of-vocabulary words. This provides a handy representation that I will use in the correction interface for spoken search queries.



(a) Original lattice



(b) Merged lattice



(c) Open vocabulary confusion network

Figure 6.4: The stages of creating an open vocabulary confusion network. The numbers on the edges of 6.4a and 6.4b denote acoustic and language model log likelihoods. The numbers on the edges of 6.4c denote word posterior probabilities.

Recognition using a word+graphone language model results in a recognition lattice containing both words and graphones (figure 6.4a). Graphone nodes are merged into OOV words while maintaining the lattice's original paths. After merging, the lattice contains complete OOV words (figure 6.4b). Thereafter, a word confusion network is created. The network's best guess is called the *consensus hypothesis* and is found by taking the highest probability hop in each set.

The lattice merge algorithm is an iterative process. First, a lattice node which is a graphone without the EOW letter is selected. For every graphone following this node, a new node is created, concatenating the two nodes' letters and phones. Edges are created to and from the new node, maintaining the original lattice paths and scores (see algorithm 1 for details). The next non-EOW node is selected, and the process continues. Upon completion, the lattice contains complete OOV words with known pronunciations (figure 6.4b). An open vocabulary confusion network (OVCN) is created by transforming the lattice into a confusion network (figure 6.4c). I built the confusion network using an algorithm based on the one implemented by SRILM [138].

Algorithm 1 can be computationally demanding as it requires creation of a new lattice node for every possible path between a starting graphone and all reachable ending graphones. To keep the algorithm tractable, I added pruning. I pruned a partial OOV word candidate if its average per letter combined acoustic and language model score became too unlikely compared to a successfully merged OOV. The pruning beam was applied based on the best, completely merged OOV word whose start time was within a window of the search candidate's start time. This helped ensure reasonable sets of OOV guesses were obtained for different time regions in the lattice.

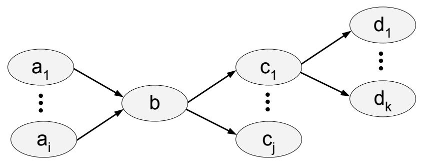
OOV unit and word penalties were incorporated by adding a single OOV unit penalty to each graphone's language model probability. EOW graphones were further adjusted to account for a single OOV word penalty. The merge algorithm then proceeded as normal.

6.6 Word+Graphone Experiments

In this section, I describe experimental results using word+graphone language models for recognition. I investigate the effectiveness of my new techniques of two-step training, variable OOV penalties, and open vocabulary confusion networks.

6.6.1 Experimental Setup

For recognition, I used HTK v3.4 [167; 168], HDecode, and the acoustic model training recipe from [147]. I trained cross-word triphones on the WSJ0 [57], WSJ1 [5] and TIMIT [58] corpora (223 hours). I parameterized audio into a 39-dimensional feature vector consisting of 12 Mel-frequency cepstral coefficients plus the 0^{th} cepstral, deltas and delta deltas, normalized using cepstral mean subtraction. The model had 10K tied-states with 32 continuous Gaussians per



```
N = lattice nodes, words/graphones ending at a time
E = lattice edges, with acoustic and LM scores
B = all nodes that are graphones without the EOW letter
while B not empty do
    b = some node from B
    for each child c of b do
        Create new node n where n:
          Ends at time of node c
          Concatenation of letters/phones in b and c
        for each child d of c do
            Create edge n→d with same score as c→d
        for each ancestor a of b do
            Create edge a→n with sum of scores of edges a→b and b→c
        if n does not have end letter then
            Add n to B
    Remove all edges to and from b, delete b
Remove all nodes no longer on any path through lattice
```

Algorithm 1: Merges a lattice's graphones into complete OOV words.

state (64 for silence). I used the CMU phone set without stress markings (39 phones plus silence). Each phone HMM had three output states and a left-to-right topology with self-loops. For a pronunciation dictionary, I used the CMU dictionary [25]. Decoding took about $6\times$ real-time (6 times as long as the input audio) on a 3 GHz machine.

In this section, I used newswire vocabularies and language models to allow comparison with past work on open vocabulary recognition (e.g. [18]). I trained baseline and word+graphone language models using SRILM and the CSR-III text corpus (222M words). I used the most frequently occurring words to generate 20K and 64K vocabularies. I trained language models using interpolated modified Kneser-Ney smoothing and a count cutoff of 1 for unigrams, 1 for bigrams, and

Model	Graphones	Bigrams	Trigrams
20K baseline	-	10.0M	8.5M
20K + 1-2 graphones	4K	11.6M	10.3M
20K + 1-3 graphones	9K	12.6M	10.4M
20K + 1-4 graphones	20K	13.5M	10.3M
20K + 1-5 graphones	33K	14.1M	10.2M
20K + 1-6 graphones	37K	14.3M	10.2M
20K + 1-7 graphones	40K	14.4M	10.2M
64K baseline	-	13.5M	8.9M
64K + 1-2 graphones	4K	14.6M	9.9M
64K + 1-3 graphones	9K	15.1M	9.9M
64K + 1-4 graphones	19K	15.4M	9.8M
64K + 1-5 graphones	31K	15.6M	9.7M
64K + 1-6 graphones	35K	15.7M	9.7M
64K + 1-7 graphones	37K	15.7M	9.7M

Table 6.6: The number of graphones, bigrams and trigrams in the baseline and word+graphone language models.

3 for trigrams. Table 6.6 shows the number of graphones and n-grams in the baseline and word+graphone language models.

I used a bigram language model for decoding and rescored lattices with a trigram model. I obtained the "best path" results by performing a Viterbi search of each utterance's recognition lattice while enforcing the appropriate word insertion, OOV unit and OOV word penalties. I merged OOV units into word-level results using the EOW-letter to segment any contiguous OOV words. I obtained the "conf net" results by using the consensus hypothesis from each utterance's confusion network. I constructed the confusion network by using algorithm 1 on the recognition lattice and then creating a confusion network using SRILM.

For a test set, I combined the WSJ0 si_dt_20 and WSJ1 si_dt_20 test sets (894 sentences, 2.3% OOV at 20K vocab, 0.3% at 64K). In this chapter, I refer to this combination of test sets as `si_dt_20`. Some example sentences from `si_dt_20`

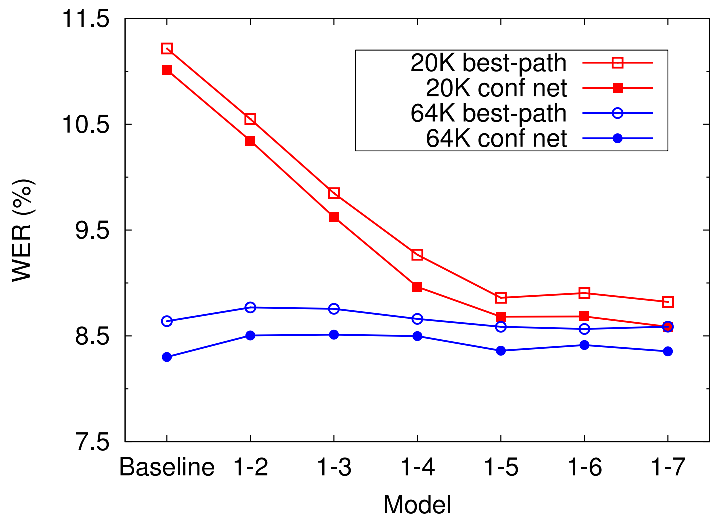


Figure 6.5: Best-path and confusion network results at 20K and 64K in-vocabulary sizes. The x-axis shows the language model used, from baseline (no graphones) up to 1–7 graphones. Results on the `si_dt_20` test set.

and their corresponding recognition results are shown in table 6.7.

6.6.2 Graphone Size and Decoding Technique

Figure 6.5 shows the recognition accuracy using 20K and 64K language models both without graphones (baseline) and with graphones of varying lengths. Using 20K in-vocabulary words, word+graphone language models reduced WER by up to 2.4% absolute, 21% relative. Longer graphones provided bigger WER reductions, though gains were small beyond 1–5 graphones. There were no gains using 64K in-vocabulary words. This is probably due to the low 0.3% OOV rate at a 64K vocabulary.

Figure 6.5 also shows the difference between my new open vocabulary confusion network technique and the existing best-path technique. I found that for all graphone sizes, open vocabulary confusion networks provided small, but consistent WER gain of about 0.2% absolute over a best-path technique.

Ref:	the serpent in this case is government
Base:	the $\underline{\text{servant}}$ in this case is government
OOV:	the $\begin{pmatrix} \textit{serp} \\ \texttt{s er p} \end{pmatrix} \begin{pmatrix} \textit{ent} \bullet \\ \texttt{ah n t} \end{pmatrix}$ in this case is government
Ref:	weaving dog fur into yarn isn't simple
Base:	$\underline{\text{wheeling}}$ dog $\underline{\text{for}}$ into yarn isn't simple
OOV:	$\begin{pmatrix} \textit{weav} \\ \texttt{w iy v} \end{pmatrix} \begin{pmatrix} \textit{ing} \bullet \\ \texttt{ih ng} \end{pmatrix}$ dog $\underline{\text{for}}$ into yarn isn't simple
Ref:	they lit bonfires smashed windows and threw stones at police
Base:	they $\underline{\text{live on fires}}$ smashed windows and threw stones $\underline{\text{and}}$ police
OOV:	they lit $\begin{pmatrix} \textit{bon} \\ \texttt{b aa n} \end{pmatrix} \begin{pmatrix} \textit{fire} \\ \texttt{f ay er} \end{pmatrix} \begin{pmatrix} \textit{s} \bullet \\ \texttt{z} \end{pmatrix}$ smashed windows and threw stones $\underline{\text{and}}$ police
Ref:	it's kind of peculiar
Base:	it's kind of $\underline{\text{the kill you}}$
OOV:	it's kind of $\underline{\text{the}}$ $\underline{\begin{pmatrix} \textit{kill} \\ \texttt{k ih l} \end{pmatrix} \begin{pmatrix} \textit{ian} \bullet \\ \texttt{y ah n} \end{pmatrix}}$
Ref:	it looks as though father sadlowski has a long haul ahead of him
Base:	it looks as though father $\underline{\text{said laos key have}}$ long haul ahead of him
OOV:	it looks as though father $\underline{\text{said} \begin{pmatrix} \textit{low} \\ \texttt{l aw} \end{pmatrix} \begin{pmatrix} \textit{sky} \bullet \\ \texttt{s k iy} \end{pmatrix} \text{have}}$ long haul ahead of him

Table 6.7: Some recognition examples from the `si_dt_20` test set. Recognition results used either a baseline in-vocabulary language model (denoted base) or a word+graphone language model (denoted OOV). Word errors are shown by $\underline{\text{red underlining}}$. In the first three examples, graphones have successfully recognized the OOV words "serpent", "weaving", and "bonfires". In the fourth example, graphones have tried and failed to recognize the in-vocabulary word "peculiar". In the last example, graphones have tried and failed to recognize the OOV proper name "sadlowski".

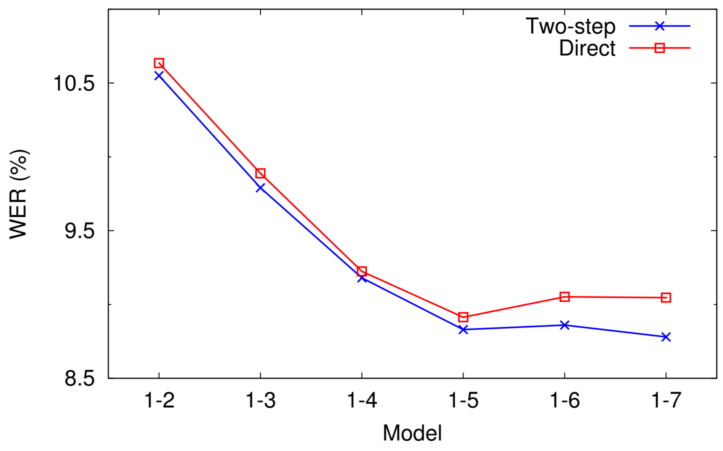


Figure 6.6: Comparison of the WER of a word+graphone models trained either using a direct alignment (top line) or using my new two-step process (bottom line).

6.6.3 Two-step Language Model Training

I compared my new two-step training process (described in section 6.5.1) to the existing method of directly aligning using the graphone model used by the recognizer. As shown in figure 6.6, the two-step process performed only slightly better than the single-step method. The longer 1–6 and 1–7 graphone models showed the largest benefit. This is probably because bigger graphones like 1–7 are not very good at letter-to-phone conversion. By using the two-step process, more accurate phone sequences were obtained for OOVs in the language model training text and this resulted in better OOV recognition.

6.6.4 Decoding Time

As shown in figure 6.7, decoding using graphones required substantially more computation time. On average, recognition with the baseline in-vocabulary language models performed at 2.5 × real-time (on a 2.4 GHz computer). On average, recognition using the word+graphone language models performed at 6.4 × real-time. The 20K and 64K language models took about the same time for decoding. The additional decoding time required by word+graphone language models was

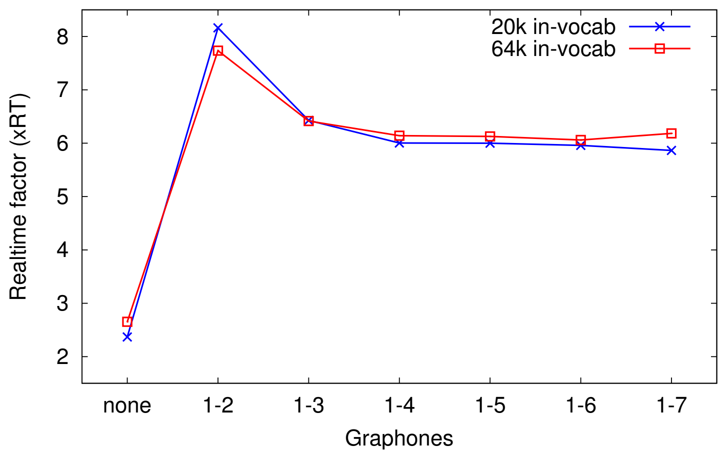


Figure 6.7: Real-time factor for decoding `si_dt_20` utterances using different word and word+graphone language models.

probably the result of additional search hypotheses caused by the inclusion of the short, sub-word graphones in the recognizer's vocabulary.

6.6.5 Insertion and OOV Penalties

I modified my recognizer to allow separate penalties for in-vocabulary words, graphones, and EOW-graphones (see section 6.5.2). Until this point, I used a uniform penalty for every dictionary item. This uniform penalty was set to -20 (log base e), a value found to be the optimum insertion penalty using a word-only language model on the `si_dt_20` test set.

I investigated if accuracy could be improved by varying the penalties associated with OOVs. Leaving the word insertion penalty fixed at -20, I varied both the OOV unit penalty and the OOV word penalty. As shown in figure 6.8, using a small OOV unit and OOV word penalty provided the best results. At least on `si_dt_20`, the added flexibility of separate OOV penalties did not outperform simply using a single insertion penalty for all dictionary items. So for further experiments, I will use a word insertion penalty of -20, an OOV unit penalty of -20, and an OOV word penalty of 0.

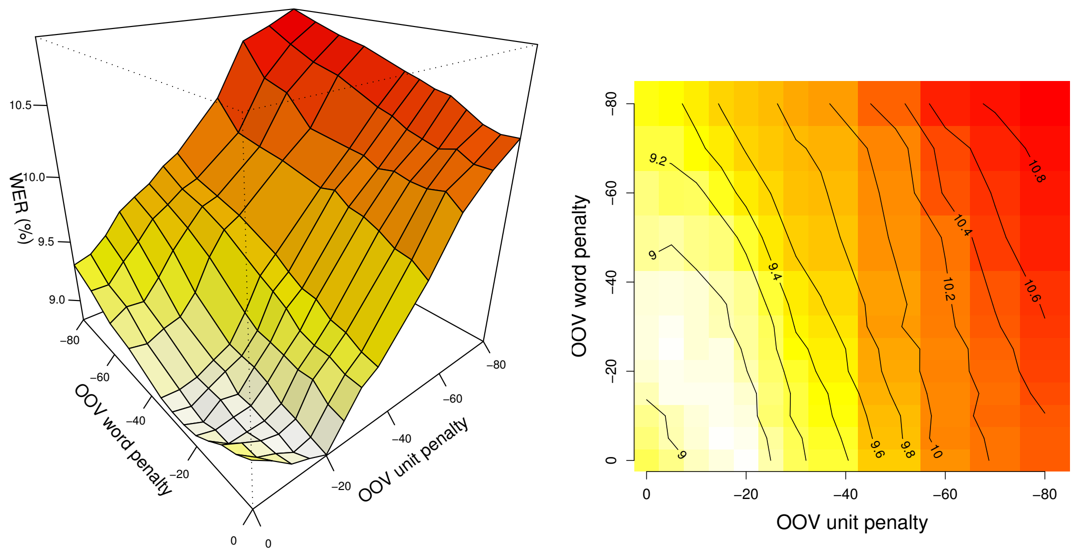


Figure 6.8: WER as both the OOV unit penalty and OOV word penalty were varied. Results on `si_dt_20` using a 20K + 1–5 graphone language model.

6.7 Web Search Query Corpus

I could find no publicly available text or speech corpora in the web search query domain, so I collected my own. As a basis, I used past user searches mined from various web search "spy" pages. These spy pages display recent or popular searches made by users of a particular search engine. Over a period of about four years, I collected around 4 million queries. To further expand my collection, I used the "related search" feature of the *Yahoo!* search engine via their developer API [4]. The *Yahoo!* related search feature returns a list of other possible queries based on the original query. Table 6.8 shows some example queries and their related search results.

I added the *Yahoo!* related search variants to my collection. I also used *Yahoo!* to filter out any of my original queries that had no related results. This helped eliminate many of the ill-formed queries in the original search spy results. After expansion and filtering, I had 7.1M search queries.

To provide more data, I fed individual "seed" words to the *Yahoo!* related search feature. For some seed words, *Yahoo!* would respond with a list of related searches that probably represent past searches on Yahoo!. As seed words, I chose

Original search	Related searches
bacon recipe	hot bacon dressing recipe
	chicken bacon recipe
	how to bake bacon recipe
	filet mignon wrapped in bacon recipe
	chicken wrapped in bacon recipe
	scallops bacon recipe
	bacon wrapped water chestnut recipe
german steins	beer steins
	german beer steins
	beer mugs
	beer glasses

Table 6.8: Example search queries and their related searches as returned by Yahoo!.

	Average	Median	Maximum	Standard deviation
Words	2.9 (2.6)	3 (2)	37 (39)	1.4 (1.7)
Chars	18.9 (16.9)	10 (15)	168 (224)	8.5 (9.2)

Table 6.9: Statistics about the queries in my corpus. Numbers in parentheses are from a study of search queries submitted to *Google* from mobile phones and PDAs [79].

words from the CMU dictionary, words in the titles of Wikipedia articles, and the top unigrams from the *Google* corpus [23]. From approximately 500K unique words, I obtained an additional 3.1M search queries.

I combined the training queries from both the spy pages and from the seeded queries. I reserved 10% of the queries as test data. This left 9.1M queries from which to train my language model. Table 6.9 summarizes my search query corpus. My corpus statistics were broadly similar to the published statistics on search queries submitted to *Google* from mobile phones and PDAs [79]. Obviously, my collection procedure may have resulted in a smaller and lower quality data collection than what might be available to commercial search engine companies. However, as I will show, even my modest corpus provided major gains over merely using readily-available newswire training text. I will also show that models built using my corpus were successful at recognizing spoken search queries.

6.8 Web Search Experiments

In this section, I describe computational experiments I conducted to test recognition techniques for spoken search queries. I compare accuracy using normal vocabularies, vocabularies with inferred pronunciations, and word+graphone language models.

6.8.1 Experimental Setup

I ran computational experiments using recorded audio containing spoken search queries. For recognition, I used HTK and HDecode as in prior experiments. I trained a speaker-independent acoustic model with 16 continuous Gaussians per state (32 for silence) and 8K tied-states. I report error bars using a one standard error (σ_{se}) confidence interval obtained via per-utterance bootstrap resampling [17].

To provide a development audio test set, I recorded 755 search queries. Audio was recorded at 16 kHz using a wired headset microphone. The queries were drawn from a search engine spy [1] that was not used for the original data collection. The test set had an OOV rate of 7.9% using a 86K vocabulary that contained all CMU dictionary words that occurred in the search query corpus. The average search query length was 3.0 words (19.5 characters). The test set had a per-word perplexity of 296 using a 86K vocabulary trigram language model trained on search queries. Of the 755 test set queries, 36% of the queries had also occurred in the language model's training set of queries.

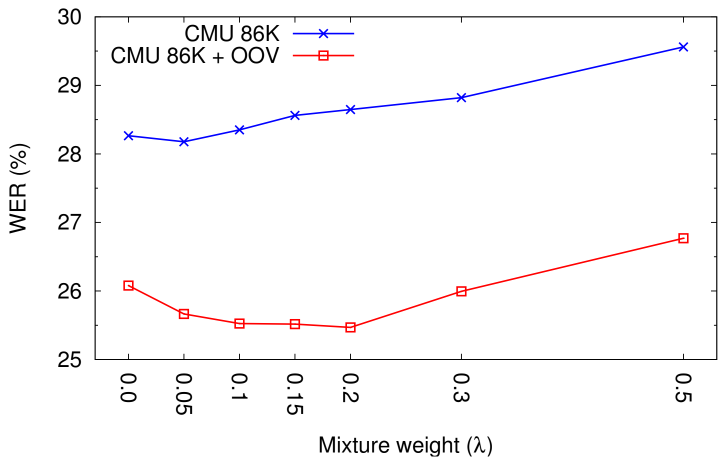


Figure 6.9: WER varying the mixture weight between the newswire and search language models. $\lambda = 0.0$ uses only the search language model, $\lambda = 0.5$ uses an equal mix of each. The in-vocabulary set was the 86K words in CMU that occurred in the search corpus.

6.8.2 Mixture Language Model

When building the language model there is a question of the relative importance of using search-specific training text versus easily-available newswire text. In addition, since my search corpus was somewhat small (27M words), I hypothesized it might help to use additional non-search training text.

To investigate this issue, I trained a range of language models that mixed a search language model and a newswire language model. For the newswire language model, I segmented the sentences in the CSR corpus (222M words) into short pseudo-sentences of 1 to 5 words. I trained the search and newswire language models separately and then created a mixture language model with SRILM [138]. The mixture model used a linear combination of the n-gram probabilities from each language model with the relative importance of the two models being controlled by a mixture weight λ. I trained both an in-vocabulary language model and a word+graphone language model that added 17K 1–4 graphones.

The in-vocabulary language model trained only on search data ($\lambda = 0.0$) had a WER of 28% and the word+graphone language model had a WER of 26% (figure 6.9). The in-vocabulary language model trained only on newswire data

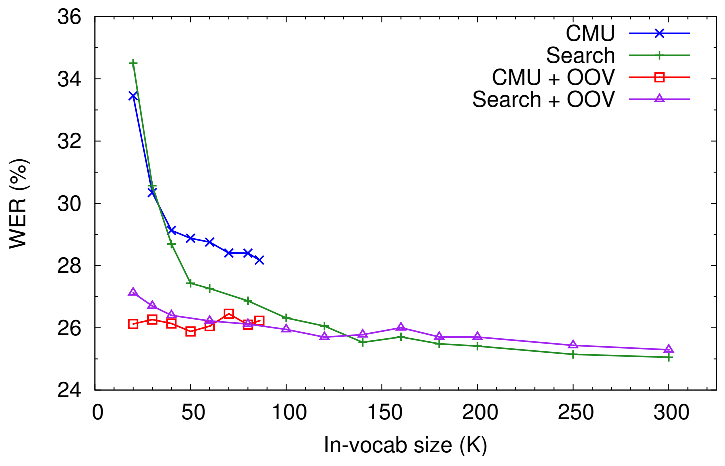


Figure 6.10: WER using different vocabularies types and sizes.

($\lambda = 1.0$) had a WER of 41% and the word+graphone language model had a WER of 37%. Thus it appears to be very beneficial to train on search-specific data.

Mixing in a small amount ($\lambda = 0.05$) of the newswire language model improved the in-vocabulary language model slightly, reducing WER by 0.1% absolute. The word+graphone language model saw more of an advantage, improving WER by 0.6% absolute for $\lambda = 0.2$. The newswire language model may be improving the OOV mixture by providing more data to train n-grams involving graphones. Except where specified, the remaining experiments used $\lambda = 0.05$.

6.8.3 Vocabulary Type

I tested two types of vocabularies, each consisting of the most frequent words seen in my search corpus. The first type (denoted CMU) was limited to words that occurred in the CMU pronunciation dictionary. The other type (denoted Search) was free to use any word. For words that were not in the CMU dictionary, the single best phone sequence was found with a letter-to-phone graphone model (0–1 graphones, 6-gram language model). In addition, I tested vocabularies that added 17K 1–4 graphones (denoted CMU+OOV and Search+OOV).

As shown in figure 6.10, the type and size of vocabulary had a large influence

LM size	$\times$RT	Memory	WER $\pm \sigma_{\text{se}}$
Unigram	1.4	152 MB	41.7 $\pm$ 1.4
Bigram	1.3	172 MB	27.1 $\pm$ 1.3
Trigram†	1.3	291 MB	26.3 $\pm$ 1.3

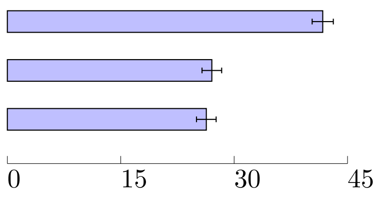


Table 6.10: Real-time factor, memory use, and WER for different language model sizes. Results using a 100K Search vocabulary. †Bigram used for recognition, rescored with a trigram.

on the WER. The CMU vocabulary performed the worst with a WER of 28% (using 86K words). For large vocabulary sizes, the other three vocabulary types performed nearly the same with a WER of about 25%. Given the extra recognition time and memory required by word+graphone language models, the Search vocabulary is to be preferred.

6.8.4 Number of Pronunciations

Instead of using just the 1-best pronunciation found for words in the Search vocabulary, I investigated including multiple pronunciations by doing an n-best search using the letter-to-phone model (see section 6.3.7.1). I tested assigning the pronunciation probabilities of the multiple alternatives either uniformly or by setting them proportional to their probability under the letter-to-phone model. I found there was no reliable improvement using multiple pronunciations. Using multiple pronunciations also slowed decoding significantly; using three pronunciation variants doubled decoding time.

6.8.5 Language Model Size

Prior work on web search query recognition has used a unigram language model [53; 129]. As shown in table 6.10, I found that using a bigram language model gave more accurate results and did not slow decoding. I found rescoring

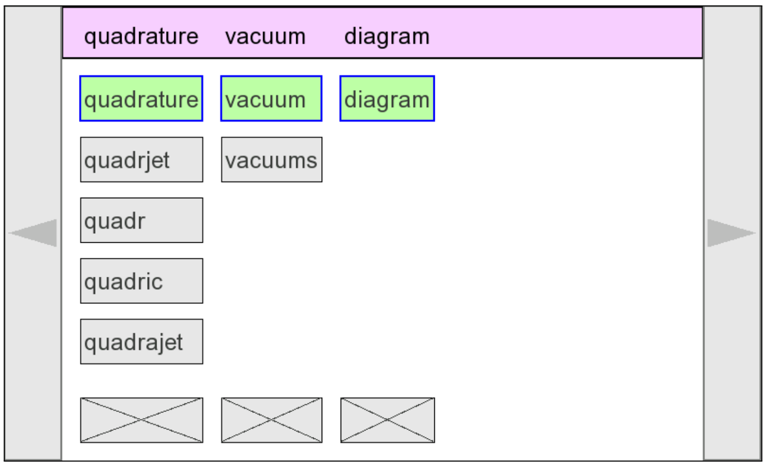


Figure 6.11: Example of correcting a query using Parakeet. The user has spoken "quadrajet vacuum diagram". The recognizer has proposed various novel words in the first column including the correct one. The user corrects recognition errors by selecting alternative words from each column. The "X" box at the bottom allows deletion of words.

the bigram lattices with a trigram language model improved accuracy slightly and required very little additional time. The memory use of HDecode with the bigram language model was only 13% more than with the unigram. The trigram language model however required 70% more memory than with the bigram. The 100K Search language model had 4.5M bigrams and 5.5M trigrams.

6.8.6 Performance using Correction Interface

I will describe computational experiments in the context of a correction interface based on the Parakeet system [152] (also described in chapter 5). The interface allows users to speak a query and then correct recognition errors by selecting words from a confusion network. The user can also delete a word using the "X" box. Figure 6.11 shows Parakeet's main correction interface. Words in the recognizer's best hypothesis are displayed at the top. Below each word in the best hypothesis are other probable alternative words.

Using these correction features, I simulated an "oracle" user. The oracle user

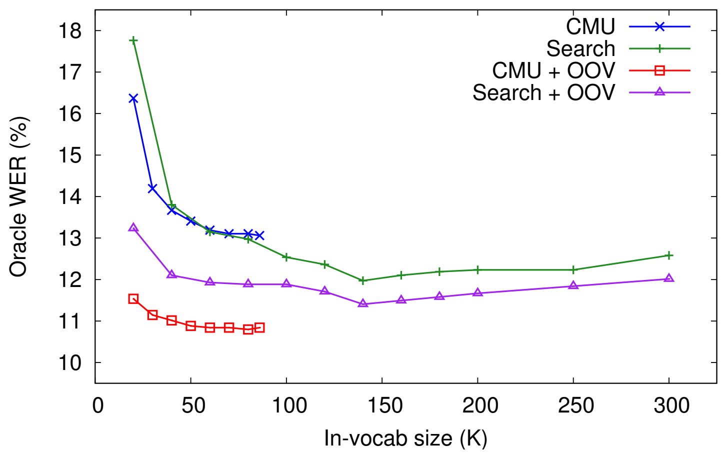


Figure 6.12: The oracle WER achievable using the Parakeet correction interface with four alternatives per recognized word and a delete box.

made optimal use of a given confusion network and set of interface actions to correct as many errors as possible in the recognition result. The number of word choices in Parakeet was chosen to provide the majority of oracle accuracy gains while still keeping buttons big enough for easy use by touch (see chapter 5).

As shown in figure 6.12, the correction interface offered substantial WER reductions across all vocabulary types and sizes. The CMU+OOV vocabulary was the best with an oracle WER of 11%. This means that from an initial 1-best WER of 25%, over half (56%) of the recognition errors could be corrected using just Parakeet's confusion network interface.

6.8.7 Testing on Realistic Audio

In the user study (to be described shortly), I collected 776 queries from 4 users. The users spoke the queries while walking around indoors using the Parakeet interface on a mobile device . I used the collected audio in offline experiments using the recognition techniques and setup described previously (section 6.8.1).

As shown in table 6.11, accuracy was improved by 34% relative using a mixture language model, switching to a Search vocabulary, and increasing the vocabulary size to 100K. I found that adding graphones to the language model helped the

Vocab type	In-vocab size	Mix λ	$\times$RT	WER $\pm\ \sigma_{\text{se}}$
CMU	86K	1.00	2.3	53.5 ± 1.6
CMU	86K	0.00	1.7	37.7 ± 1.5
CMU	86K	0.05	1.9	37.4 ± 1.5
CMU+OOV	86K	0.05	6.5	36.1 ± 1.4
Search	86K	0.05	2.3	35.7 ± 1.4
Search	100K	0.05	2.3	35.1 ± 1.4
Search	140K	0.05	2.7	35.3 ± 1.4
Search+OOV	140K	0.05	8.1	35.5 ± 1.4

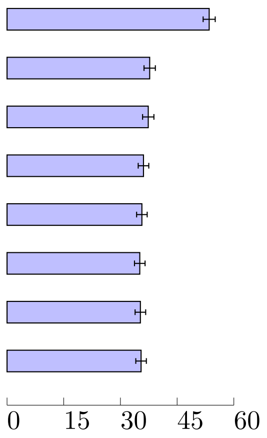


Table 6.11: Performance of different language models on the audio recorded during the user study. These results used the HTK recognition setup.

CMU vocabulary but not the Search vocabulary. The word+graphone language models were also found to be much more computationally expensive.

6.9 User Study

Having observed the theoretical benefits of introducing an error correction interface to voice search, I set out to investigate how real users would perform using such an interface. My user study had three goals. First, to collect externally valid speech data from users walking around and performing search queries on a mobile device. Second, to validate the design of my voice search system in a realistic setting. Third, to get an estimate of the envelope of voice search performance.

6.9.1 Participants and Apparatus

I recruited four participants from the university campus. All participants were female native speakers of American English and their ages ranged from 23 to 26.

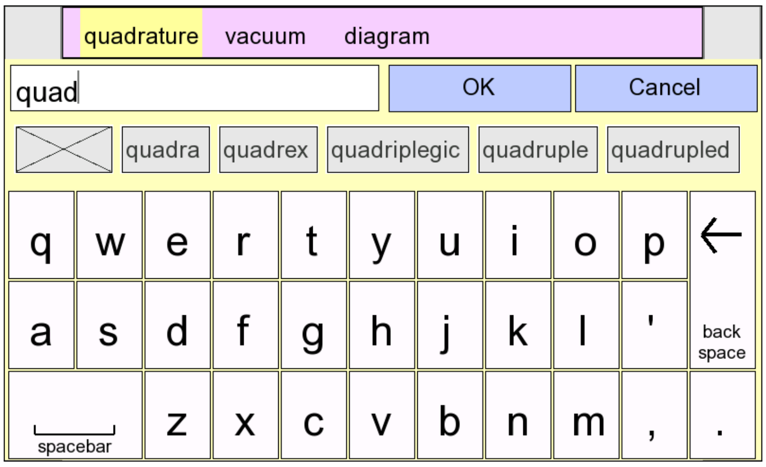


Figure 6.13: Parakeet's keyboard interface. As the user types, word predictions appear above the keyboard.

Participants were asked to speak a set of past queries obtained from a search engine spy [1]. Participants used the Parakeet system (chapter 5) running on a Nokia N800 mobile device (figure 6.1). As previously described, users could correct errors using Parakeet's confusion network interface (figure 6.11). They could also correct errors using Parakeet's on-screen keyboard (figure 6.13). Participants only had to speak and correct the query, they did not execute actual web searches or browse results.

I used a 100K Search vocabulary, automatically generating a single pronunciation for words not in the CMU dictionary. I used a language model without graphones because in pre-study testing I found that using a word+graphone language model was too slow to provide near real-time performance. I found I was able to obtain better accuracy using an in-vocabulary language model and using the extra processing time to reduce the amount of pruning needed during the recognition search.

Audio was recorded at a sampling rate of 16 kHz using a Jabra M5390 wireless microphone. I used PocketSphinx [74] for recognition. The recognition setup was closely based on the one described in chapter 5. For this study, I used a US-English acoustic model and increased the number of codebook Gaussians to 1024.

While Parakeet can perform recognition on the mobile device, this was not practical for this difficult recognition task. Instead, recognition was performed on a nearby laptop that was wirelessly connected to the N800 device. However, to the participant, it appeared just as if the device was doing the recognition.

6.9.2 Method and Setup

Participants first were given an overview of the task and shown the Parakeet interface. They were allowed to do 5 practice queries under the supervision of the experimenter.

Participants were prompted with a series of search queries on the screen. They were asked to enter the search queries as quickly and as accurately as possible. They were told to accept a search query if they deemed it sufficiently close to the original search query to be meaningful. For example, a minor misspelling of a search term may have been accepted by a user since a web search would still probably yield the intended result. After the participants were satisfied with their search query, they hit a special hardware button to move to the next query.

Participants spoke and corrected queries for 20 minutes. After a short break, they corrected queries for another 20 minutes. Participant 1 and 2 did a second session the following day. This session was identical to the first but without the training and practice portion.

6.9.3 Results

In total, participants completed 776 queries (2434 words). The OOV rate was 3.8% using a 100K Search vocabulary. In 43% of the queries, participants reviewed the 1-best result and went directly to the next query. In only 7% of the queries did participants respeak a query. Participants spent 47% of their correction time using the confusion network and 53% of their time using the on-screen keyboard.

Error rate	Before correction	After correction
Character (CER)	22.9	1.7
Word (WER)	48.0	8.3
Sentence (SER)	61.6	17.4

Table 6.12: Error rates before and after correction in the user study.

6.9.3.1 Error Rate

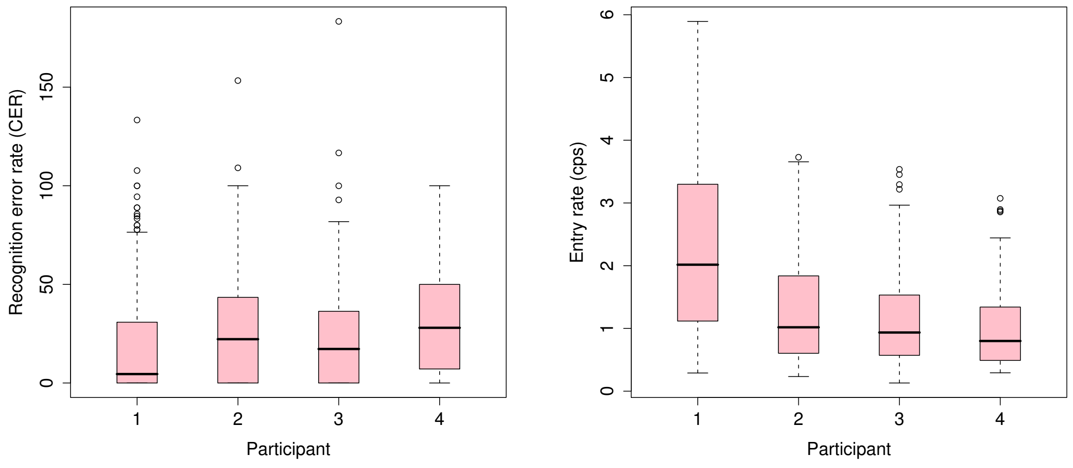
Users had to wait on average 2.7 s ± 0.8 s for the recognition result to appear. The average recognition WER was 48% (table 6.12). Participants corrected most recognition errors with an after correction WER of 8%.

The recognition WER seen in the user study was higher than the WER seen on the development test set used earlier. There were two reasons for this. First, listening to the audio we found it contained numerous audio artifacts introduced by the wireless microphone as well as significant breath noises from the participants. Second, the recognition setup used during the user study needed to operate in near real-time. It was thus less accurate than the one used in the offline experiments. Using the audio data recorded during the user study and using the 100K Search vocabulary, the HTK recognition setup had a lower WER of 35% (table 6.11).

I also measured recognition error rate using character error rate (CER) to provide a finer grain error measure than WER. Given the short nature of queries, WER can be high due to only minor differences such as spacing or pluralization. Participants average CER was 23%. Figure 6.14a shows the CER of each participant. Participant 1 had the lowest CER of 18% CER while the other three participants had a CER of between 25% and 32%.

6.9.3.2 Entry Rate

Including recording, recognition delay, and correction time, it took participants on average 18 s ± 15 s to enter a query. Participants' mean entry rate was



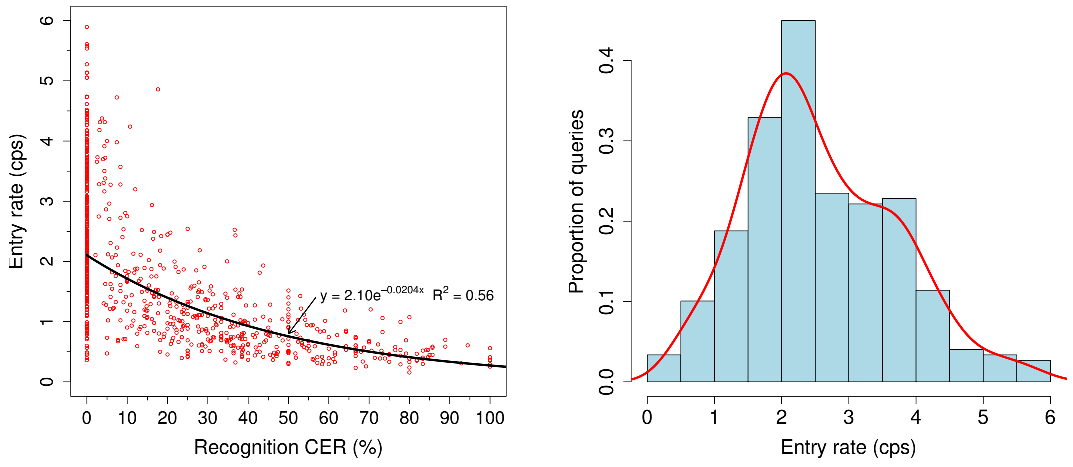
(a) Error rate (b) Entry rate

Figure 6.14: Error rate and entry rate of each participant in the user study.

1.7 cps ± 1.2 cps (characters per second). Figure 6.14b shows the entry rate of each participant. Participant 1's was particularly fast with an entry rate of 2.2 cps compared to the other three participants who were closer to 1.1 cps. This was probably due to her lower overall recognition error rate. As a reference point, Kamvar and Baluja [79] report that *Google*'s mobile search queries (typically typed using a telephone keypad or a QWERTY thumb keyboard) had a mean entry rate of 0.42 cps (calculated from table 2 in [79]).

6.9.3.3 Influence of Recognition Errors

As expected, participants' entry rate was heavily influenced by the recognition error rate. Figure 6.15a gives us an estimate of the envelope of voice search performance. It can be used to predict the performance of voice search at different recognition error levels. Notably, even at a 40% CER, the text entry rate is still at an acceptable 1 cps (12 wpm). At low error rates between 0–10% CER, I observed a range of entry speeds from 0.4 to 5.9 cps (4 to 71 wpm). In other words, spoken search queries can potentially be very fast.



(a) Results at all error rates

(b) Recognition correct

Figure 6.15: Entry rates of participants at different recognition error rates during the user study (left). When recognition was completely correct, I observed a range of entry rates (right).

6.10 Related Work

In this section, I review related work in letter-to-phone conversion, open vocabulary recognition, and the recognition of spoken search queries.

I used the joint multigram model for letter-to-phone conversion and also for open vocabulary recognition. This model was originally developed by Deligne et al. [43; 44]. They benchmarked the model on letter-to-phone conversion using a French pronunciation dictionary. They used Viterbi training and kept one pronunciation variant per training word (in contrast, I used all variants). Their best *French* result had a PER of 5.2% using 1–3 graphones and a 2-gram language model. They did not present results on a completely held-out evaluation test set (e.g. they optimized parameters such as model size on their test set).

Galescu and Allen [56] tested graphones on English letter-to-phone and phone-to-letter conversion. They used the NetTalk and CMU dictionaries and kept multiple pronunciations per word. While not explicitly stated, from the examples given, they appeared to be using 1–5 or longer graphones. They used a 4-gram language model with Witten-Bell smoothing. On the CMU letter-to-phone task, they obtained a PER of 7.0%. My best model performed slightly better with a PER of 6.51%.

Bisani and Ney [16] present experiments using graphones for letter-to-phone conversion in English and German. It is not clear whether they used multiple pronunciations per training word. They tested graphones of sizes 1–1 to 1–6 and language models sizes up to 3-gram. Similar to my findings, they found short graphones and a long-span language model worked best. Also similar to my findings, EM-training only improved their best model slightly (0.01% absolute). Their best English model used 1–2 graphones and a 3-gram language model. This model had a PER of 4.0% on the CELEX dictionary [26]. They did not give results on a completely held-out evaluation test set. A direct comparison with my result is difficult due to the different dictionary used.

Chen [28] used EM to train a 0–1 graphone model. He then iteratively trained a 7-gram maximum entropy language model. He presented results on an unseen evaluation test set. Similarly, I also used EM training, 0–1 graphones, iterative language model training, and an unseen evaluation test set. His 0.7% improvement over my model is perhaps due to his longer-span language model and different language model smoothing technique. It could also be due to our different training and test set splits.

The most extensive experiments using graphones were reported by Bisani and Ney in [19]. They showed their graphone models were superior to other models on a wide variety of pronunciation dictionaries. To my knowledge, their CMU result of 5.88% PER is the best error rate reported on the CMU data set. They trained their graphone language model using EM and optimized the discount parameters of their model using Powell's method [119]. As discussed in section 6.4.9, a model trained using their software and my training/test split had a PER of 6.41% (compared to my best result of 6.51%). In my experiments, their more complicated training regime offered only a small advantage in terms of letter-to-phone accuracy.

Other methods besides graphones have been used for letter-to-phone conversion. For example, decision trees [98], knowledge-based rules [46], pronunciation by analogy [39], neural networks [8], and HMMs [142]. These other techniques typically perform worse than the graphone approach. For a comparison of graphones versus some of these other techniques, see [19].

While some open vocabulary recognition techniques aim to just detect an OOV word without providing the spelling (e.g. [12], [166]), I will focus here on techniques that can produce an OOV's letter sequence.

To my knowledge, Galescu [55] was the first to incorporate sub-word units into a recognizer's vocabulary. He tested units that were constructed by using a mutual information criteria to merge automatically learned letter-phone pairings. Similar to my work, he also used a special pseudo-letter to present the end of a word. He performed recognition using a 3-gram with 20K in-vocabulary words and 11K sub-word units. He tested on two small test sets of 92 and 94 utterances with OOV rates of 2.4% and 3.9% respectively. His system obtained 0.7% and 1.9% relative WER reductions. Using a similar number of graphones (12K), I was able to obtain a much higher WER reduction of 14% relative at an OOV rate of 2.3%.

My best-path word+graphone recognition experiments were very similar to the experiments reported by Bisani and Ney [18]. Using 20K in-vocabulary words and 12K graphones, they obtained a 15% relative WER reduction at an OOV rate of 2.6%. I trained a model with a similar number of 12K 1–5 graphones and got a relative reduction of 14%. Their best performing result used 1–4 graphones and they reported degraded accuracy for longer graphones. My results on the other hand, showed additional accuracy gains were possible using longer graphones. Using 40K 1–7 graphones, I obtained a 21% relative WER reduction. My longer graphone models had substantially more graphones than the models they report. I speculate that their degraded performance may have been due to over-pruning of their graphone inventory.

A technique based on phone recognition is described in Decadt et al. [42]. A phone sequence is recognized using a 5-gram phone language model and then converted to letters using a memory-based machine learner [38]. Using a word recognizer, they had a WER of 14.7% on a Dutch test set (3.5% OOV at 40K vocab). Using a phone recognizer and converting to letters using their memory-based learner, they had a WER of 44.8%. This poor result suggests that phone-based recognition may not be powerful enough to provide good open vocabulary recognition accuracy.

In Siivolta et al. [132], a set of 64K likely Finnish sub-word "morph" units were learned using the minimum description length principle. The morphs consisted of just chunks of letters since the mapping from letters to phones is mostly trivial in Finnish. They compared recognition performance using 3-gram language models trained either on morphs, syllables, or conventional words. Due to the huge number of word forms in Finnish, the 64K word language model had a high 20% OOV rate. They found morph units performed the best, reducing WER by 22% relative. This approach is less likely to be successful in English due to the lower overall OOV rates of word-based language models and also English's non-trivial mapping of letters to phones.

In order to recognize spoken search queries, I automatically generated pronunciations for words not in the CMU dictionary. In Lööf et al. [100] and Gollan et al. [60], pronunciations were automatically generated using graphones to cover the most frequent OOV words in a corpus of parliamentary speeches. They found adding the OOV words to the lexicon reduced WER slightly (1.7% relative [100], 1.4% relative [60]).

There has been very little work on the recognition of spoken search queries. Franz and Milch [53] built a speech interface to the *Google* search engine using a commercial speech recognizer. Their interface took queries over the phone and results were viewed in a desktop web browser. In order to obtain real-time performance, they used a unigram language model. They learned common pairs of words in order to boost the recognition accuracy of the unigram model. They trained their model on 20M queries (compared to my smaller collection of 9M queries). On a test set of 18 speakers making 809 queries, their recognition was completely correct 43% of the time. On my test set of 4 speakers and 776 queries, recognition was completely correct 48% of the time.

Sherwani et al. [129] created a voice interface to Wikipedia. They used a commercial recognizer and a 100K unigram language model. In a user study, they tested the voice interface and a smart phone implementation that used T9 or multitap for query entry. No difference in task success was found between the two interfaces. In the voice interface, the initial recognition was completely correct 58% of the time. They found using speech was faster for entering queries, but slower for navigation tasks. Users reported preferring the non-speech smart phone interface.

6.11 Conclusions

In order to build a system to recognize search queries, I first had to address the handling of the large and diverse vocabulary typical of search queries. I did this by first investigating the use of graphones for letter-to-phone conversion. I would make the following recommendations to anyone interested in using graphones for letter-to-phone conversion:

1. **Graphone model size** – Use small graphones coupled with a long-span language model. I found 0–1 graphones and a long-span 6-gram language model performed well.
2. **Graphone initialization** – For small graphones, the graphone inventory can be initialized with a set of all possible graphones and the initial graphone probabilities set to be equal.
3. **Training method** – Train graphones using Viterbi instead of EM. Viterbi is almost as good as EM and much simpler to implement.
4. **Smoothing method** – Choose a language model smoothing method such as interpolated Kneser-Ney.
5. **Iterative training** – Only small gains were seen iteratively training the language model. It is simpler and almost as accurate to train the language model in a single-pass directly on the unigram graphone segmentation.

6. **Count cutoffs** – Avoid using count cutoffs during language model training.
7. **End-of-word letter** – Avoid using an end-of-word letter if word boundaries are already known.

After obtaining state-of-the-art letter-to-phone performance from graphones, I trained a hybrid language model containing both words and graphones. I tried a number of novel ideas to try and improve word+graphone recognition. First, I developed a new algorithm that allows the creation of open vocabulary confusion networks (OVCN) containing both in- and out-of-vocabulary words. I showed that using OVCNs provided small, but consistent improvements over existing best-path techniques. They also allow a convenient way to incorporate OOV alternatives into a correction interface like Parakeet.

Second, I implemented a two-step alignment process in which my best letter-to-phone model was used to obtain phone sequence for OOV words in the word+graphone language model training text. These phone sequences were then used as a constraint when segmenting OOVs using the model that would be used during recognition. The two-step alignment process provided only a small advantage.

Third, I examined having different penalties for in-vocabulary words, OOV units, and end-of-word OOV units. To my knowledge, all previous word+graphone research has used the same word insertion penalty for all lexical entries. I found that using a single penalty for all lexical types was in fact close to optimal.

I described the corpus I collected of over 10M search queries. Using this corpus, I showed how to train a good language model for search queries. I found that a vocabulary using inferred pronunciations was much better at recognizing queries than a vocabulary limited to a fixed pronunciation dictionary. I took advantage of this technique in a system I built to recognize spoken search queries on a mobile device. In a user study, I found that participants were able to speak and correct a search query in about 18 seconds with an entry rate of 1.7 cps (20 wpm). This was despite a high overall WER of 48%.

In conclusion, I believe that with an appropriate out-of-vocabulary model, a large search query corpus, and an efficient correction interface, speech recognition has the potential to be a fast and effective entry method for searching the web.

Chapter 7

Conclusions

In this thesis, I have examined how to improve the process of correcting speech recognition errors. I first looked at the problem of helping users detect errors. A recurring idea in the speech recognition field has been to use confidence scores to annotate recognition results. I conducted an experiment to test if users would be faster, or more accurate, at finding recognition errors if potential errors were visualized by underlining them in a shade of red. I found that confidence visualization neither hurt nor helped users detect errors. However, when errors were correctly visualized, users were more likely to detect them. But this advantage was offset by the user missing errors that were not correctly visualized.

Next, I looked at how users changed their speech during spoken corrections. I found users adopted a more hyperarticulate speaking style during corrections, changing their speaking rate, pausing, pitch, intensity, and formant frequencies. Despite these changes, I found no significant difference in recognition accuracy between normal and hyperarticulate utterances. I found the recognition of word or phrase corrections was particularly problematic. By adapting my acoustic model on correction-style speech, I obtained a 13% relative reduction in word error rate (WER) on spoken corrections.

I then developed two novel interfaces for the correction of speech recognition errors. The first was Speech Dasher. Speech Dasher uses a continuous zooming interface to allow navigation through the space of recognition hypotheses. For words not in the recognition hypothesis space, Speech Dasher supports easy fallback to a letter-by-letter predictive spelling of words. In a formative user study, participants used speech and gaze to write at 40 words per minute (wpm). This was despite a recognition WER of 22%. Using Dasher without speech, participants were substantially slower with an entry rate of 20 wpm.

The second interface I described was Parakeet: a touch-screen interface designed for efficient mobile text entry. Parakeet is based on selecting words from a word confusion network. In computational experiments, I observed that about half of all recognition errors could be corrected via Parakeet's word confusion network interface. With the inclusion of a predictive software keyboard, Parakeet enabled users to correct any error. In a user study, participants were able to write at 13 wpm while walking outdoors. This was despite a recognition WER of 26% and significant recognition delays. Neglecting the recognition delays, users could have written at up to 26 wpm outdoors. As a reference point, users of T9 wrote at 16 wpm while seated indoors after 15 sessions [162].

Finally, I investigated the problem of searching the web by voice. Using a simple model that describes letter and phone pairings, I showed that standard language modeling techniques can provide state-of-the-art letter-to-phone accuracy. I described how to extend existing techniques to recognize new words while maintaining a word lattice structure. This allowed the creation of open vocabulary word confusion networks containing both in- and out-of-vocabulary words. I took advantage of these techniques to test mobile entry of web search queries using the Parakeet interface. In a user study, I found users spoke and corrected search queries in about 18 seconds. This was despite a recognition WER of 48%. As a reference point, users entering text queries to *Google* using a phone or PDA took about 40 seconds [79].

7.1 Final Thought

In this thesis, I have shown that speech recognition does not need to be perfect to be useful. Even at high error rates, interfaces that leverage all the recognition information can yield efficient and easy to use speech interfaces. A good correction interface will degrade gracefully, gleaning whatever useful information is available from the user's speech. In this manner I believe we can improve the productivity and satisfaction of speech recognition users.

Appendix A

Speech Recognition Basics

In this appendix, I provide a basic overview of the speech recognition process. My goal is to give the reader enough background to understand the interfaces and experiments described in this thesis. For a more in-depth treatment of statistical speech recognition, see [34; 76; 78].

A.1 Statistical Formulation

Speech recognition is the search for the best word sequence $\hat{W}$ given an audio recording O:

$$\hat{W} = \operatorname*{argmax}_{W} P(W|O)\,. \tag{A.1}$$

Using Bayes' rule, this can be written as:

$$\hat{W} = \operatorname*{argmax}_{W} \frac{P(O|W)P(W)}{P(O)}\,. \tag{A.2}$$

The denominator in (A.2) can be dropped since it is invariant during the search for the best word sequence $\hat{W}$:

$$\hat{W} = \operatorname*{argmax}_{W} \underbrace{P(O|W)}_{\substack{\text{acoustic}\\ \text{model}}} \underbrace{P(W)}_{\substack{\text{language}\\ \text{model}}}\,. \tag{A.3}$$

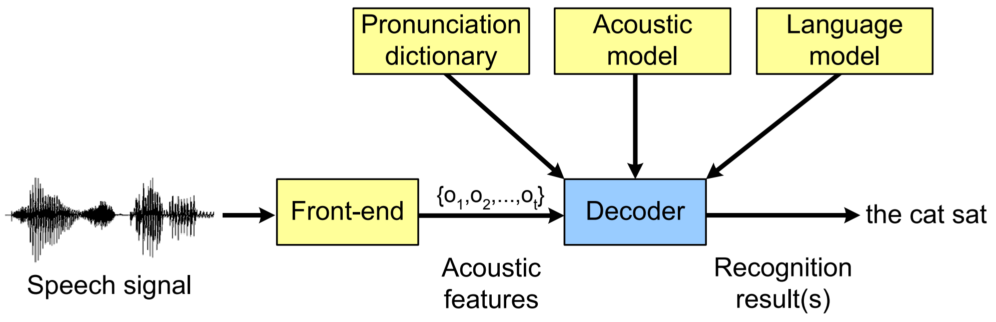


Figure A.1: Overview of the process and components involved in speech recognition.

The first part of (A.3) is the likelihood of the audio recording given a word sequence hypothesis and is called the *acoustic model*. The second part of (A.3) is the prior probability of the word sequence and is called the *language model*.

Figure A.1 provides a graphical depiction of the speech recognition process and its major components. In the rest of this appendix, I will describe the details of each component.

A.2 Acoustic Modeling

A.2.1 Front-end Processing

The acoustic model's job is to decide how probable the audio signal O is given a particular word sequence. The audio signal is discretized into a large number of samples per second (e.g. 16,000 samples per second for *wideband* audio or 8,000 samples per second for *narrowband* audio). Each sample can take on a large number of values (typically 2^{16}).

Recognition using this very high dimensional audio data would be difficult. So instead, a much lower dimensional representation is first calculated by the recognizer's *front-end*. A commonly used representation is based on Mel-frequency cepstral coefficients (MFCCs) [41]. The original audio is segmented into short segments (around 25 ms) and a set of MFCCs (often 13) is calculated for each segment. To help capture longer-term trends in the these features, the deltas (velocity) and delta deltas (acceleration) of the MFCCs can be concatenated to the feature vector. Thus the original acoustic signal O is assumed to be well represented by a sequence of acoustic observations:

$$O \approx \{o_1, o_2, ..., o_t\} \tag{A.4}$$

where each o_i is a low-dimensional (typically 39-dimensional) feature vector derived from the original signal.

Normally, to improve recognition accuracy, the mean of an utterance's acoustic features is subtracted from each frame of audio in that utterance. This technique, known as *cepstral mean normalization/subtraction* [99], typically uses the entire audio of an utterance to compute the cepstral mean. If recognition is to be performed on "live" streaming audio, a prior window of audio can be used to estimate the mean.

A.2.2 Hidden Markov Models

In order to search over different word sequence hypotheses, the *acoustic likelihood* of the observations given a particular word sequence W is needed:

$$P(o_1, o_2, ..., o_t | W)\,. \tag{A.5}$$

Currently, the predominant model for estimating this probability is the *hidden Markov model* (HMM). An HMM is a generative model that uses a finite state machine. States in the HMM generate acoustic observations from a multi-variate Gaussian distribution (often a mixture of Gaussians). When a mixture of Gaussians is used, each state has a set of mixture weights and those weights sum to one. Each Gaussian has a mean and a covariance matrix. In order to limit the number of model parameters, a diagonal covariance matrix is often used.

Figure A.2 shows a very simple HMM that can generate only three words "the", "cat", and "sat". Each word in this example HMM has three states. Each state generates an acoustic observation and then either transitions to itself (a self-loop), or transitions to the next state in the word. After finishing a word,

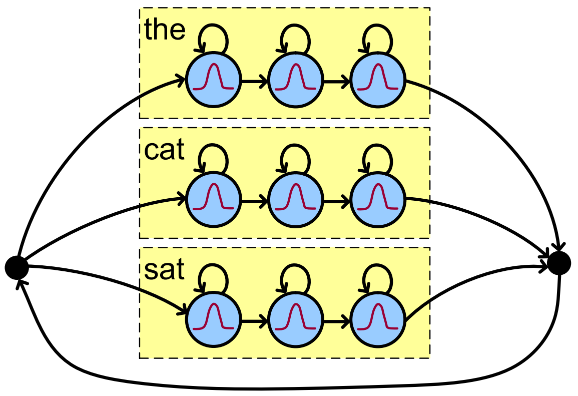


Figure A.2: An example HMM for recognizing the words "the", "cat" and "sat". Each word has three output states with a single Gaussian distribution in each output state. The solid black states are non-generating states and are used to combine the three word models into a single overall model.

the HMM loops back to the start, allowing another word to be generated, and so on. The transitions out of each state have probabilities and those probabilities sum to one.

Different paths through an HMM for a given word sequence may result in a different numbers of observations. For example, in figure A.2, the word sequence "cat" might generate only three observations (one at each state), or it might generate multiple observations at a particular state by using the self-loop transition. This mechanism helps HMMs handle variability in how fast or slow words are pronounced.

For further details about HMMs including how the model parameters are trained, see [121].

A.2.3 Sub-word Units

For large vocabulary recognition tasks (such as dictating an email), using a word-level HMM like figure A.2 is problematic. This is because the number of word models in the HMM would be very large and training reliable estimates for the parameters of each word model would be difficult. Instead, large vocabulary

Word	Probability	Phone sequence
the	0.7	*dh ah*
the	0.3	*dh iy*
cat	1.0	*k ae t*
sat	1.0	*s ae t*

Table A.1: An example pronunciation dictionary containing three words. These three words are pronounced using 7 different phones: *dh*, *ah*, *iy*, *k*, *ae*, *t*, and *s*.

speech recognition normally uses sub-word HMM units. A commonly chosen sub-word unit is the *phone*. A phone is a unit representing a particular speech sound in a language. To create an HMM using phones, first a mapping is needed between each word and its phone sequence (or possibly multiple phone sequences). An example *pronunciation dictionary* is shown in table A.1. Optionally, each entry for a word can have an associated *pronunciation probability*.

Using the pronunciation dictionary, an HMM is constructed concatenating together phone models based on a word's entry in the dictionary. Figure A.3 shows an example *monophone* acoustic model for a three word vocabulary. A monophone model has a single HMM for each of the phone units. Monophone models do not take into account changes that might occur to a particular phone due to its phonetic context (i.e. the phones appearing before or after it). While in figure A.3 some of the phone models appear multiple times, in practice these phone models would only be represented once with a single set of model parameters. This allows more robust estimation of the parameters since training data can be shared between words that have phones in common.

Since a phone's pronunciation can depend on the sounds before or after it, *triphones* are often used instead of monophones. A triphone represents how a particular base phone is realized given the phone before it and after it. In *word internal* triphones, the surrounding phonetic context does not include any phones from preceding or following words (i.e. the first phone of every word has no left phonetic context and the last phone of every word has no right phonetic context). In *cross-word* triphones, phonetic context crosses word boundaries.

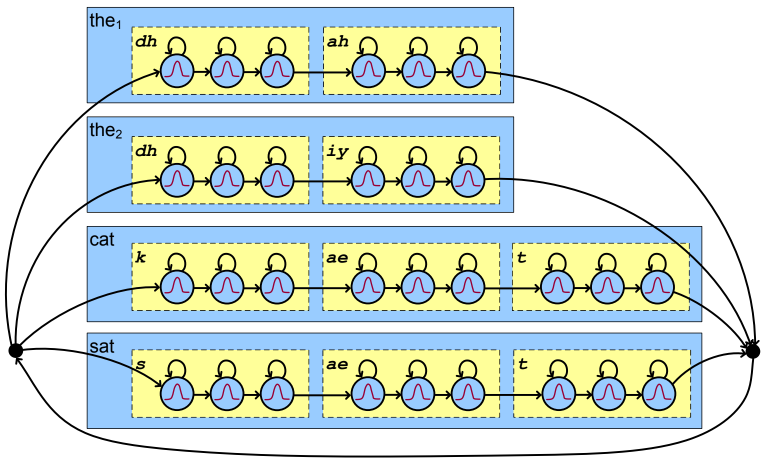


Figure A.3: An example HMM using phone models. This HMM can recognize the words "the", "cat" and "sat". Each word is made up of two or three phone models. The word "the" has two different pronunciation variants.

State-of-the-art recognizers usually use cross-word triphones (or even quinphones). In practice, there is not enough training data to reliably estimate triphones because there are so many of them. For example, in English there are around 40^3 different triphones (although not all of these will be observed in practice).

A popular approach to robustly estimate parameters despite training data sparsity is *parameter tying*. In parameter tying, particular sets of parameters can be shared between multiple triphone models. So for example, the parameters of the Gaussian mixture model might be tied among triphone states that are "similar". The similarity between states might take into account whether different triphones share the same base phone, appear in a similar phonetic context, etc. The tying of parameters can be found automatically by generating a decision tree that uses a set of linguistically motivated questions [171].

A.2.4 Semi-continuous Models

A large vocabulary recognizer may have thousands of tied-states. Each state in turn has its own mixture of Gaussians. The calculation of the acoustic likelihoods for all these Gaussian distributions can be computationally demanding.

An alternative approach is to use a *semi-continuous* acoustic model [73]. In a semi-continuous model, there is a single large pool of continuous Gaussian distributions. Each HMM output state has a set of mixture weights that control how important each Gaussian in the pool is for modeling speech in that state. This saves computation as the acoustic likelihood for an observation only needs to be calculated once for each Gaussian in the pool. Semi-continuous models are typically not as accurate as models with independent Gaussians at each state. I used semi-continuous models periodically in this thesis when real-time recognition was required.

A.2.5 Speaker Adaptation

The parameters of the acoustic model are normally trained on a large corpus of speech with known word transcriptions. This generates a *speaker-independent* acoustic model. The accuracy of the speaker-independent model can usually be substantially improved by performing *speaker adaptation*. In speaker adaptation, the parameters of the acoustic model are altered to better reflect an individual's voice. Adaptation can be *supervised* or *unsupervised*. In supervised adaptation, the text transcriptions of the adaptation utterances is known. In unsupervised adaptation, the text transcriptions are not known and a surrogate such as the best recognition result must be used instead.

In this thesis, I performed supervised adaptation using two popular techniques. The first is maximum likelihood linear regression (MLLR) [96]. In MLLR adaptation, a linear transform is computed that shifts the parameters of the acoustic model so as to maximize the likelihood of the adaptation data. A single MLLR transform can be computed for all Gaussians in an HMM. Alternatively, a number of different transforms can be computed. Each different transform applies only to a group of states that are "close" in acoustic space. The grouping of states can be found automatically using a regression class tree [54; 95]. MLLR adaptation tends to perform well even on small amounts of adaptation data.

The second form of adaptation is maximum a posteriori (MAP) adaptation [59]. In MAP adaptation, the model parameters are interpolated between a prior estimate (the speaker-independent parameters) and a maximum likelihood estimate based on the adaptation data. The degree to which the adaptation data influences the parameters is controlled by a settable parameter τ. Compared with MLLR, MAP adaptation typically requires more data in order to perform well. Given sufficient data, MAP adaptation tends to perform as well as an acoustic model trained just on the speaker's voice.

For a review of popular speaker adaptation techniques, see [163].

A.3 Language Modeling

The language model's job is to decide how probable a particular word sequence W is. This is normally done using a model that assumes the next word depends only on the prior history of words:

$$P(W) = \prod_{i=1}^{k} P(w_i|w_{i-1}, w_{i-2}, ..., w_1) \tag{A.6}$$

where w_i denotes each of the k words in the word sequence W.

The most common form of language model is the *n-gram language model.* In an n-gram language model, the next word depend only on the prior $(N-1)$ words:

$$P(W) \approx \prod_{i=1}^{k} P(w_i|w_{i-1}, w_{i-2}, ..., w_{i-N+1}) \,. \tag{A.7}$$

A language model has an associated *vocabulary.* The vocabulary defines all words that the language model can make predictions about. All the words in the vocabulary must have an entry in the recognizer's pronunciation dictionary. If a word is not in a language model's vocabulary, it is said to be *out-of-vocabulary (OOV).*

A.3.1 Training

An n-gram language model is estimated by counting the number of times a particular word appears in a given prior context of words in the training data. In a *unigram* language model, the prior context is ignored and a word's probability depends only on how often it occurred in the training text. In a *bigram* language model, a single word of prior context is used. In a *trigram* language model, two words of prior context are used.

For large amounts of training data, representing every n-gram seen in the training data may require too much storage. During training, a *count cutoff* can be used. A count cutoff causes n-grams occurring only a small number of times to be treated as if they had never occurred. An alternate is *entropy pruning* [137]. In entropy pruning, first a language model is trained without count cutoffs. The language model is then pruned to remove n-grams that do not contribute significantly to predicting the training text. Note that entropy pruning still requires that the initial unpruned model be small enough to fit within memory.

A.3.2 Smoothing and Interpolation

Using the raw counts to estimate the probabilities in (A.7) is problematic. The training data is unlikely to contain instances of every word in every possible prior context. For example, even if "the cheetah sat" was never seen in the training text, the probability of seeing "sat" after "the cheetah" should probably still be greater than zero. This problem is addressed by *smoothing*. In smoothing, some probability mass is reserved for events unseen in the training text.

While smoothing allows non-zero probability for the word "sat" after "the cheetah", it does not make use of information known about the distribution of words that follow the shortened context of "cheetah". The language modeling techniques of *backoff* and *interpolation* specify how this information is incorporated. In backoff, only if "the cheetah sat" was never seen in the training data is the shortened context used to estimate the probability of “sat”. In interpolation, the shortened context is always used to inform the estimate for the longer context.

For an overview and empirical evaluation of language model smoothing, back-off, count cutoffs, and interpolation, see [29].

A.4 Decoding

In speech recognition, *decoding* refers to the process of finding the most likely word hypothesis (or hypotheses) given the acoustic signal. Decoding normally uses the Viterbi algorithm to efficiently search the space of possible word sequences. Viterbi is a classic dynamic programming approach that makes the search efficient by utilizing the fact that observations are conditionally independent given the generating state in the HMM.

The hypotheses considered by the decoder need an aggregate likelihood that combines the acoustic and language model likelihoods. This combination is normally done using an empirically determined *language model scale factor*. The language model scale factor helps balance the difference in dynamic range between the likelihoods of the acoustic and language model. This is necessary due to incorrect assumptions made by the models. An additional empirically determined *word insertion penalty* is also commonly used. The word insertion penalty causes each word in a hypothesis to incur a fixed penalty. The word insertion penalty helps reduce the number of insertion errors.

Normally a decoder computes likelihoods in the log domain to prevent numeric underflow. So in the log domain, the search for the best hypothesis takes the form:

$$\hat{W} = \underset{W}{\operatorname{argmax}} \ln P(O|W) + \alpha \ln P(W) + \beta L \tag{A.8}$$

where α is the language model scale factor, β is the word insertion penalty, and L is the number of words in hypothesis W.

Note that if the pronunciation dictionary contains probabilities, (A.8) would also contain a term for the pronunciation probability and a *pronunciation scale factor*.

For large vocabulary recognition, it is not typically possible to search over all possible word sequences. In order to keep decoding computationally tractable, *beam search* is normally used. In beam search, word hypotheses that become too improbable compared to the current best hypothesis are discarded. This form of pruning is not admissible and can cause *search errors*. The *beam width* controls the trade-off between recognition speed and the number of search errors.

The decoder may want to return not only the best hypothesis, but also other competing hypotheses. A conceptual framework for doing this is *token passing* [172]. In token passing, virtual tokens explore the search space of possible hypotheses. Each token keeps tracks of its probability, its current location in the HMM model, and its past word history. The token passing search normally uses beam width pruning to remove improbable tokens. The search also typically limits the total number of tokens in any particular HMM state.

For an overview of different decoding strategies for speech recognition, see [11].

A.5 Result Representations

The results of recognition can be represented in a variety of ways:

- **1-best** – The 1-best result is simply the best sequence of words found during the decoder's search.
- **N-best list** – An n-best list is the the set of top hypotheses found during the search. These hypotheses are listed in order from most probable to least probable. The list can be limited to a fixed number of hypotheses.
- **Word lattice** – A word lattice is a directed acyclic graph where each path through the graph is a recognition hypothesis (figure A.4). The nodes in the lattice have a word label and a time. The edges between nodes contain the acoustic and language model likelihoods of that transition.

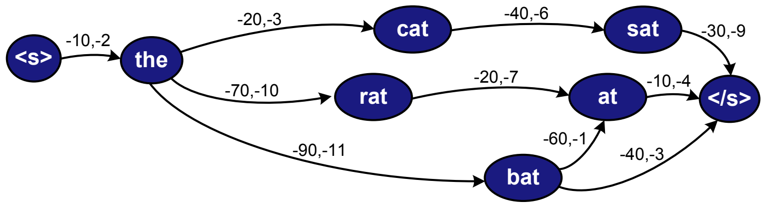


Figure A.4: An example recognition word lattice. The numbers on the edges represent acoustic and language model log likelihoods.

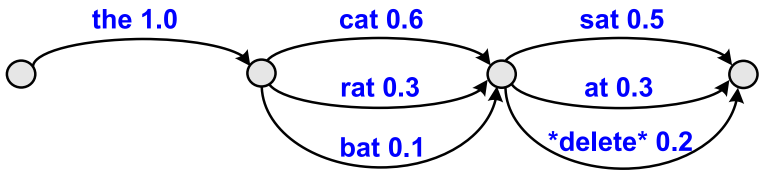


Figure A.5: An example word confusion network. The numbers on the edges represent posterior probabilities.

Lattices have several advantages to n-best lists. First, they can represent many more possible hypotheses. Second, they contain useful information about the recognizer's search such as times, acoustic likelihoods, and language model likelihoods. Third, lattices allow easy post-processing of the recognition hypothesis space. For example, a lattice can be expanded and rescored with a longer-span language model to improve recognition accuracy.

- **Word confusion network** – A word confusion network [104] is a time-ordered set of clusters where each cluster contains competing word hypotheses along with their posterior probabilities (figure A.5). The word confusion network is built from the lattice generated during the speech recognizer's search. The best recognition results, or *consensus hypothesis*, is found by taking the highest probability edge in each cluster. Word confusion networks can contain a "delete" pseudo-word. The delete word represents the hypothesis that nothing was said in that particular cluster.

  Word confusion networks have several advantages to lattices. First, they are much more compact. Second, the best path in a word confusion network tends to minimize the number of word errors [104]. Lattices, on the other hand, tend to minimize how often the entire sentence is correct.

A.6 Performance Metrics

A variety of different metrics are used in this thesis. Here is a description of the most important ones:

- **Word error rate** – The word error rate (WER) is the most common metric used to report the accuracy of a speech recognizer. The WER is the word edit distance required to change the recognition result into the correct (reference) text. It is normally reported as a percentage and calculated as:

  $$\text{WER }\% = 100 \cdot \frac{I + S + D}{T} \tag{A.9}$$

  where I is the number of insertion errors, S is the number of substitution errors, D is the number of deletion errors, and T is the number of words in the reference text. Note that WER is not a probability. It can be greater than 100% in cases were there are numerous insertion errors.

  The character error rate (CER) is analogous to WER but the edit distance is calculated based on individual characters instead of words. The CER is a finer grain measure and allows a word to "get credit" for being close to a reference word (e.g. differing in only say pluralization or ending).

- **Real-time factor ($\times$ RT)** – The real-time factor measures how much time decoding took as compared to the audio length of an utterance. For example, if a 10 second utterance took 20 seconds to recognize, the recognizer performed at $2 \times \text{RT}$.

- **Cross-entropy** – As used in this thesis, the cross-entropy measures how well a particular language model predicts events in some test text. The lower the cross-entropy, the better the language model predicts the text. The cross-entropy is the average number of bits of information provided by each word:

  $$H_P(W) = -\frac{1}{L} \log_2 P(W) \tag{A.10}$$

where W is the test text, L is the number of words in W, and P(W) is the probability of W under the language model.

- **Perplexity** – The perplexity measures the average number of "choices" a language model has when predicting the next word. For example, if a language consists entirely of the digits 0–9 and those digits are equally probable, the perplexity of the language is 10. The perplexity is calculated from the cross-entropy as follows:

$$PP_P(W) = 2^{H_P(W)}. \tag{A.11}$$

- **Out-of-vocabulary rate** – The out-of-vocabulary (OOV) rate is the percentage of words in a given text that are not in the recognizer's vocabulary. This is important as OOV words are bound to be recognized incorrectly. Furthermore, OOV words tend to cause "collateral damage" to nearby in-vocabulary words [164].

Appendix B

Corpus of Spoken Dictation and Corrections

B.1 Detail of Tasks

This section lists the sentences and corrections collected in the corpus of spoken dictation and corrections. The correct target sentence is listed first in each block. This sentences was read twice, once before any simulated errors (part one of experiment) and once before any errors on that particular sentence (part two of experiment). The highlighted portion of the first sentence (if any) shows which words were asked for during users' correction attempts. Words with a † superscript indicate a word that the user was asked to spell in the last correction attempt. Subsequent lines (if any) show the simulated errorful recognition results presented to the user. Words in red and italics were the simulated word errors and were presented to the user. The first two tasks listed below were the practice sentences. The remaining 40 tasks were given to users in random order.

1. when we moved in there was barely a tree around
   when we moved in there was *bear lee* a tree around
   when we moved in there was *pear* tree around

2. the activities are fun especially the trips curtis† said
   the activities are fun especially the trips *kurt* said
   the activities are fun especially the trips *court* said

3. i found four pieces of gold

4. we went on a gold mining tour
we went on a gold mining *tool or*
we went on a gold mining *or*
we went on a gold *mine in tool*

5. but the get together wasn't about making salad
but *they* get together *was in* about making salad
but the get together *was in* about making salad
but *to* get together *was it* about making salad
but the get together *was an* about making salad

6. grin has been signed to a two year contract
green *had* been signed to a *too* year contract
current has been signed to a two year *con tract*
drain has been signed to a two year contract

7. the bank forecast a slight profit for the full year

8. city income plummeted creating the worst budget crisis in a decade
city income plummeted *trading* the worst budget crisis in a decade
city income plummeted *grading* the worst budget crisis in a decade
city income plummeted *trading* the worst budget crisis in a decade

9. a recent trip to columbia state park stands out in his mind
the recent trip to columbia state park stands out in his mind
our recent trip to columbia state park stands out in his mind
but recent trip to columbia state park stands out in his mind

10. in fact even some english speaking parents felt left out of the loop

11. because he is still in office no one can be appointed to replace him
because *he's ill* *an* office no one can be appointed to replace him
because *he's bill* *and* office no one can be appointed to replace him
because *he's* still *an* office no one can be appointed to replace him

12. her father billy jones started driving the store's horse drawn delivery wagon at age twelve
her father billy jones started driving the *stores* horse drawn delivery *weighed in* at age twelve
her father billy jones started driving the *stored* horse drawn delivery *weigh and* at age twelve

13. at midday wells fargo stock was up three dollars to sixty dollars and twenty five cents

14. participants come from throughout the county
participants come from throughout *that* county
participants come from throughout *pick* county
participants come from throughout *rick* county
participants come from throughout *but* county

15. in tough times people look to mayors

16. no one has publicly called for him to step down
    no one has *published cleat* called for him to step down
    no one has *public cleat* called for him to step down
    no one has *public eat* called for him to step down

17. the dividend was reduced in order to strengthen our capital ratios

18. his white three bedroom house was the direct victim of the construction
    is white three bedroom *how's* was the direct victim of the construction
    in his white three bedroom house was the direct victim of the construction
    and white *e* bedroom house was the direct victim of the construction
    this white three bedroom house was the direct victim of the construction

19. although the line to get in seemed long we were seated quickly both times
    although the *light* to get in *seen* long we were seated quickly both times

20. a free house was a privately owned pub that purchased ale from a variety of breweries
    the free house was a privately owned *pubs* that purchased *gail* from a variety of breweries
    a free house was a privately owned *pubs* that purchased *gale* from a variety of breweries
    a free house was a privately owned *pubs* that purchased *gayle* from a variety of breweries

21. they have english† but it's geared for the anglo
    they have *been pushed* but it's geared for the anglo
    they have *been with* but it's geared for the anglo
    they have *been bush* but it's geared for the anglo

22. it is served with a bit of cognac and available daily
    it is served with a bit of *cold yak* *in* available daily
    it is served with a bit of *colon yak* *in* available daily

23. we sampled all but the dark brew
    we sampled all but the dark *per*
    we sampled all but the *dork* *per*
    we sampled all but the *darker* brew

24. the serving filled the platter and was only nine dollars

25. we wanted to document them on tape before it escapes
    we wanted to document them on tape *the* *forty skates*

26. he also recalls the eccentric old hermit† who lived nearby
    he also recalls *b is* *centric* *cold* *permit* who lived nearby
    he also recalls the *centric* *cold* *permit* who lived nearby
    he also recalls *it d* *centric* old *permit* who lived nearby
    he also recalls the eccentric old *earn it* who lived nearby
    he also recalls the eccentric old *permit* who lived nearby

27. during both visits the restaurant and outdoor patio were bustling

28. one speaker even claimed that occult† youth gangs threaten modern teenagers
one speaker even claimed that *a colt* youth gangs threaten modern teenagers
one speaker even claimed that *a cold* youth gangs threaten modern teenagers
one speaker even claimed that *a colt* *huge* gangs threaten modern teenagers
one speaker even claimed that *a colt* youth gangs threaten modern teenagers

29. but ingram† added it hasn't been an easy time for him
but *ingramham* *padded* it hasn't been an easy time for him
but *in from* *padded* it hasn't been an easy time for him
but *ingramham* added it hasn't been an easy time for him

30. to accompany your meal you can choose a pale amber or dark brew
to accompany your meal you can choose a pale amber *word* *park* *peru*
to accompany your meal you can choose a pale amber *word* *parked* *for*

31. when the district held a parent conference saturday the coffee and doughnuts were still plentiful
when the district held *the* parent conference saturday *at* the coffee and doughnuts were still plentiful
when the district held *the* parent conference saturday *of* the coffee and doughnuts were still plentiful
when the district held a parent conference saturday *at* the coffee and doughnuts were still plentiful
when the district held a parent conference saturday *an* the coffee and doughnuts were still plentiful

32. bulldozers were digging up dirt just a few yards from his corn crops last week
bull *over* digging up dirt just a few yards from his corn crops last week
he *dozing* *it* up dirt just a few yards from his corn crops last week
bulldozers *we're* digging up dirt just a few yards from his corn crops last week

33. the lobster bisque however is quite good
the *lop* *stirred* *this* *carver* is quite good
the *lop* *stirred* *this* *column* is quite good
the *lock* *stirred* *this* *carver* is quite good

34. arcadia officials declined to comment on jio's† situation
arcadia officials declined to comment on *geo's* situation
arcadia officials declined to comment on *geos* situation
arcadia officials declined to comment on *geode* situation

35. the linguine† was sauteed with mushrooms bay shrimp and prawns
the *lean guniea* was sauteed with mushrooms bay shrimp and prawns
the *lint green* was sauteed with mushrooms bay shrimp and prawns
the *lent guniea* was sauteed with mushrooms bay shrimp and prawns
the *new guinea* was sauteed with mushrooms bay shrimp and prawns
the *lynn guinea* was sauteed with mushrooms bay shrimp and prawns

36. he would come out of his house with an ax and no clothes on korts said
he would come out of his house with *a* *tax* and no clothes on *courts* said
he would come out of his house with *a* ax and no clothes on *ports* said
he would come out of his house with an ax and no clothes on *court* said

37. but he still carries a caseload as heavy as ware's†
but he still carries a caseload *is* heavy as *where's*
but he still carries a caseload as heavy as *where's*
but he still carries a caseload as heavy as *wares*

38. we had our first reunion last year said nona† christensen sixty six
we had our first reunion last *year's* said *no one* christensen sixty six
we had our first reunion last year said *no one* christensen sixty six
we had our first reunion last year said *none* christensen sixty six

39. a handful of specialty brews are listed on blackboards throughout the restaurant
a handful of specialty brews are listed on blackboards throughout the *rest*
a handful of specialty brews are listed on blackboards throughout the *rest aunt*
a handful of specialty brews are listed on blackboards throughout the *rest*
a handful of specialty brews are listed on blackboards throughout the *rest aught*

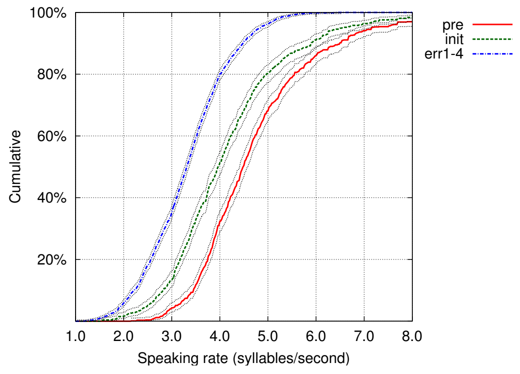
40. quality and economy are obviously the two watchwords at gerard's† in downtown san jose
quality and economy are obviously the two watchwords *and gerard* in downtown san jose
quality and economy are obviously the two watchwords at *gerard* in downtown san jose
quality and economy are obviously the two watchwords at *jerrad is* in downtown san jose

41. michael woodmansee† spent an evening in the kitchen with his friends throwing together three salads
michael *wood nancy* spent *to demean* the kitchen with his friends throwing together three salads
michael *would nancy the demean* the kitchen with his friends throwing together three salads
michael *would man see* spent an evening in the kitchen with his friends throwing together three salads

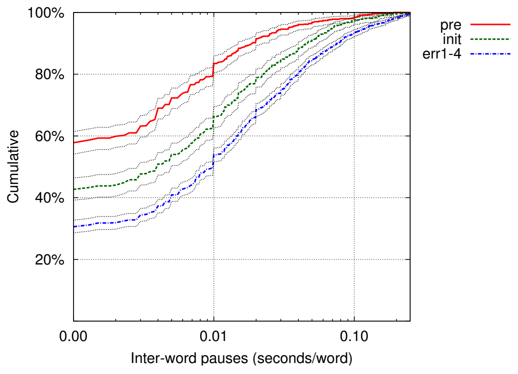
42. korts said their parents bought a summer home between love and fritch creeks in nineteen eighteen
quartz said their parents bought a summer home between *loved* and *fritsch* creeks in nineteen eighteen
courts said their parents bought a summer home between love and *fritsch* creeks in nineteen eighteen
quartz said their parents bought a summer home between *loved* and *fritsch* creeks in nineteen eighteen

B.2 All Cumulative Distributions

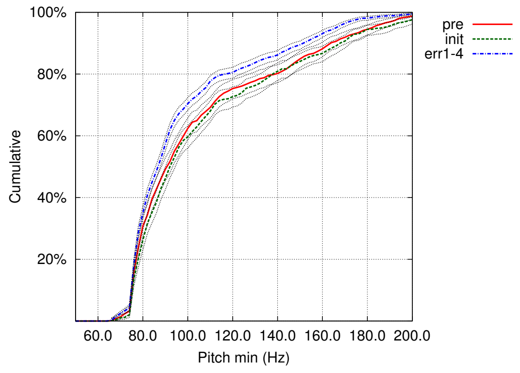
In this section, I give all the cumulative distributions of acoustic properties which I analyzed in section 3.3. The dotted lines in graphs indicate two-sigma error bars (calculated using the method in appendix D.1).



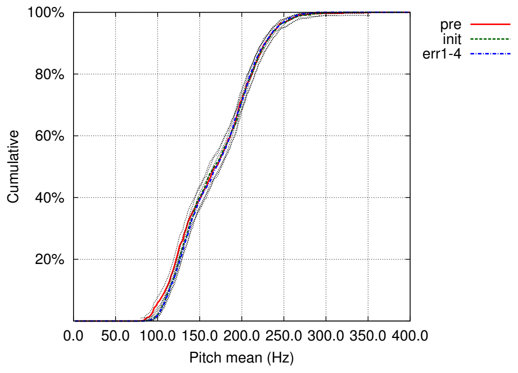
(a) Speaking rate, claim: err1-4 < init < pre



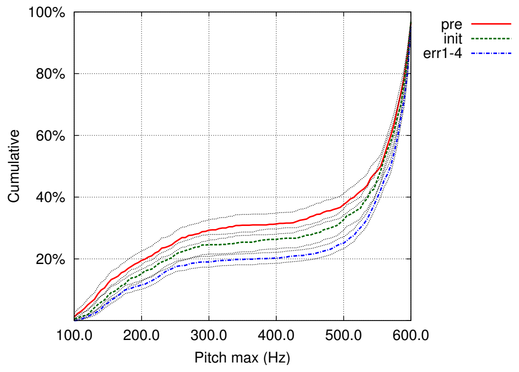
(b) Inter-word pausing, claim: pre < init < err1-4



(c) Pitch min, claim: err1-4 < pre, init

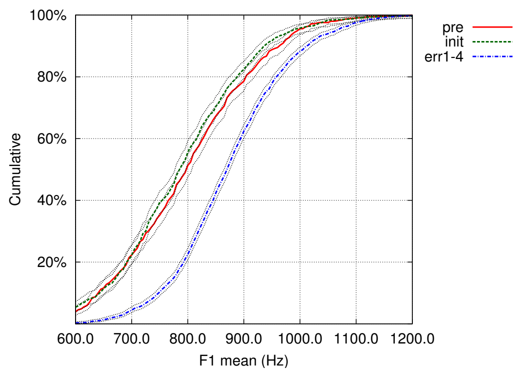


(d) Pitch mean, claim: no difference

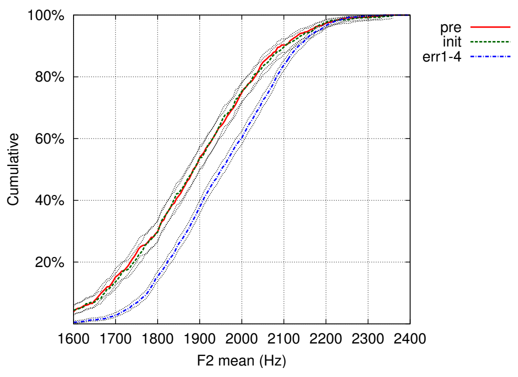


(e) Pitch max, claim: pre < init < err1-4

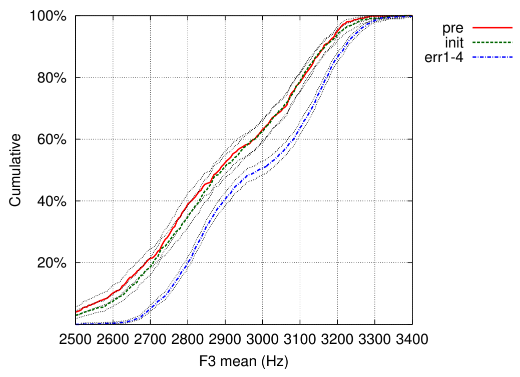
Figure B.1: Cumulative distributions of speaking rate and pitch properties.



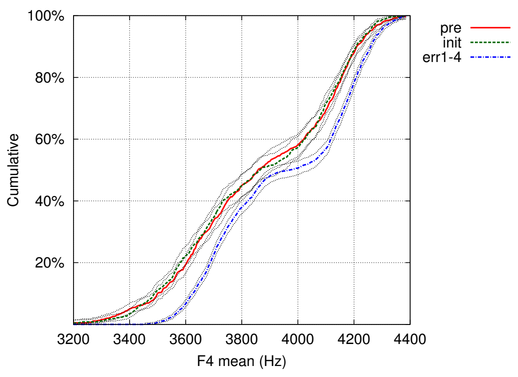
(a) F1 mean, claim: pre, init < err1-4



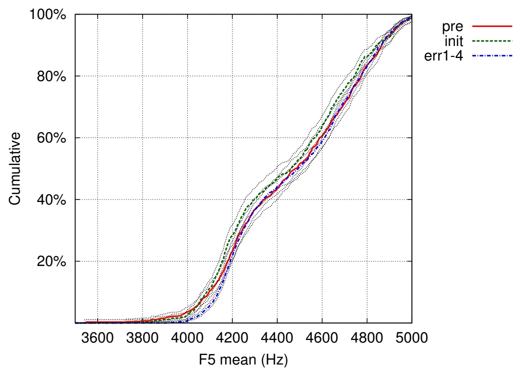
(b) F2 mean, claim: pre, init < err1-4



(c) F3 mean, claim: pre, init < err1-4

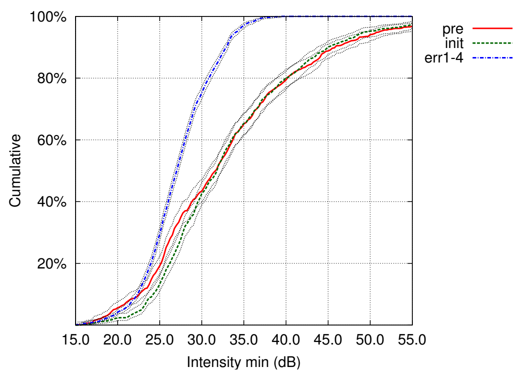


(d) F4 mean, claim: no difference

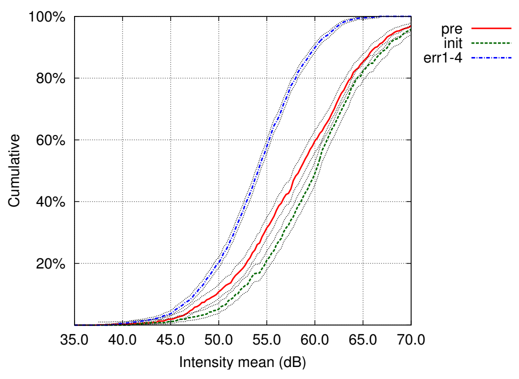


(e) F5 mean, claim: no difference

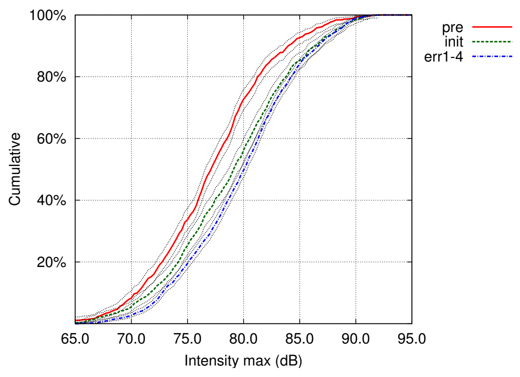
Figure B.2: Cumulative distributions of formants.



(a) Intensity min, claim: err1-4 < pre, init



(b) Intensity mean, claim: err1-4 < pre, init



(c) Intensity max, claim: pre < init, err1-4

Figure B.3: Cumulative distributions of intensity related properties.

Appendix C

Joint Multigram Derivation

As I found no detailed derivation of the joint multigram model in the literature, I provide one here. I will use a derivation in which the role of the latent variables is made explicit (as in Bishop [21], chapter 9). While the joint multigram model itself is more general, I use notation and narrative specific to joint multigrams applied to modeling letter and phone correspondences. I also assume a model in which each graphone deterministically generates a fixed set of letters and phones.

In the original reestimation formula presented in [43; 44], the forward-backward reestimation formula did not address training on a dictionary of multiple, independent training examples. I provide a formula showing optimization with respect to the entire dictionary

C.1 Latent Variable Formulation

The goal of training is to maximize the probability of the data given the model parameters. Maximizing (6.10) is equivalent to maximizing the log:

$$\log P(\boldsymbol{D}|\boldsymbol{\theta}) = \sum_{n=1}^{N} \log \sum_{s=1}^{|\boldsymbol{S}_n|} \prod_{g=1}^{|\boldsymbol{S}_{ns}|} P(G_{S_{nsg}}|\boldsymbol{\theta})\,. \tag{C.1}$$

The summation inside the log of (C.1) makes direct optimization of this quantity difficult. To assist optimization efforts, a latent variable vector $\boldsymbol{z}_n$ is introduced for each dictionary entry $\boldsymbol{D}_n$. The vector $\boldsymbol{z}_n$ specifies which of the $|\boldsymbol{S}_n|$ possible segmentations generated $\boldsymbol{D}_n$. $\boldsymbol{z}_n$ is a $|\boldsymbol{S}_n|$-dimensional binary random variable having a 1-of-$|\boldsymbol{S}_n|$ representation in which exactly one $z_{ns} = 1$ and all other elements are 0. If $z_{ns} = 1$ then the s^{th} segmentation generated entry $\boldsymbol{D}_n$.

Given the latent variable which specifies which segmentation a dictionary entry used, the probability of a training example is:

$$P(\boldsymbol{D}_n|\boldsymbol{z}_n, \boldsymbol{\theta}) = \prod_{s=1}^{|\boldsymbol{S}_n|} \left[\prod_{g=1}^{|\boldsymbol{S}_{ns}|} P(G_{S_{nsg}}|\boldsymbol{\theta}) \right]^{z_{ns}} . \tag{C.2}$$

Letting $\boldsymbol{Z}$ denote the set of latent variables for all dictionary entries, the joint distribution of the training data is:

$$P(\boldsymbol{D}|\boldsymbol{Z}, \boldsymbol{\theta}) = \prod_{n=1}^{N} \prod_{s=1}^{|\boldsymbol{S}_n|} \left[\prod_{g=1}^{|\boldsymbol{S}_{ns}|} P(G_{S_{nsg}}|\boldsymbol{\theta}) \right]^{z_{ns}} . \tag{C.3}$$

Taking the logarithm:

$$\log P(\boldsymbol{D}|\boldsymbol{Z}, \boldsymbol{\theta}) = \sum_{n=1}^{N} \sum_{s=1}^{|\boldsymbol{S}_n|} z_{ns} \left[\sum_{g=1}^{|\boldsymbol{S}_{ns}|} \log P(G_{S_{nsg}}|\boldsymbol{\theta}) \right] . \tag{C.4}$$

The posterior probability of a particular z_{ns} will be called the responsibility and denoted by $\gamma(z_{ns})$:

$$\gamma(z_{ns}) = P(z_{ns} = 1|\boldsymbol{D}_n, \boldsymbol{\theta}) . \tag{C.5}$$

Rewriting the responsibility using the definition of conditional probability:

$$\gamma(z_{ns}) = \frac{P(z_{ns} = 1, \boldsymbol{D}_n|\boldsymbol{\theta})}{\sum_{s=1}^{|\boldsymbol{S}_{ns}|} P(z_{ns} = 1, \boldsymbol{D}_n|\boldsymbol{\theta})} . \tag{C.6}$$

Since z_{ns} specifies the precise segmentation to use from dictionary entry D_n, this can be written:

$$\gamma(z_{ns}) = \frac{P(\boldsymbol{S}_{ns}|\boldsymbol{\theta})}{\sum_{s=1}^{|\boldsymbol{S}_n|} P(\boldsymbol{S}_{ns}|\boldsymbol{\theta})} . \tag{C.7}$$

Expanding the probability of the graphone sequences using (6.8):

$$\gamma(z_{ns}) = \frac{\prod_{g=1}^{|\boldsymbol{S}_{ns}|} P(G_{S_{nsg}}|\boldsymbol{\theta})}{\sum_{s=1}^{|\boldsymbol{S}_n|} \prod_{g=1}^{|\boldsymbol{S}_{ns}|} P(G_{S_{nsg}})|\boldsymbol{\theta})} . \tag{C.8}$$

Thus the responsibility is simply how likely a particular segmentation is in comparison with all other segmentations for that dictionary entry.

C.2 Expectation Maximization

With the joint distribution of the data and latent variables, an expectation maximization (EM) algorithm can be used to find a local optimum for the model parameters. I take a variational view of this procedure as in Neal and Hinton [108].

Let $Q(\boldsymbol{Z})$ be some probability distribution over the latent variables. Maximization is on a function F which is defined as:

$$F(Q(\boldsymbol{Z}), \boldsymbol{\theta}) = \mathbb{E}_{Q(\boldsymbol{Z})}[\log P(\boldsymbol{D}, \boldsymbol{Z}|\boldsymbol{\theta})] + \mathbb{H}[Q(\boldsymbol{Z})] \tag{C.9}$$

where $\mathbb{E}_{Q(\boldsymbol{Z})}[\cdot]$ denotes the expectation with respect to the distribution Q as it ranges over $\boldsymbol{Z}$ and $\mathbb{H}[\cdot]$ denotes the entropy of a distribution.

Expanding the expectation and entropy terms:

$$F(Q(\boldsymbol{Z}), \boldsymbol{\theta}) = \sum_{\boldsymbol{Z}} Q(\boldsymbol{Z}) \log P(\boldsymbol{D}, \boldsymbol{Z}|\boldsymbol{\theta}) - \sum_{\boldsymbol{Z}} Q(\boldsymbol{Z}) \log Q(\boldsymbol{Z}) . \tag{C.10}$$

Using $P(\boldsymbol{D}, \boldsymbol{Z}|\boldsymbol{\theta}) = P(\boldsymbol{Z}|\boldsymbol{\theta}, \boldsymbol{D})P(\boldsymbol{D}|\boldsymbol{\theta})$ this rewrites as:

$$F(Q(\boldsymbol{Z}), \boldsymbol{\theta}) = \sum_{\boldsymbol{Z}} Q(\boldsymbol{Z}) \log \frac{P(\boldsymbol{Z}|\boldsymbol{\theta}, \boldsymbol{D})}{Q(\boldsymbol{Z})} + \sum_{\boldsymbol{Z}} Q(\boldsymbol{Z}) \log P(\boldsymbol{D}|\boldsymbol{\theta}) . \tag{C.11}$$

The first term on the right hand side of (C.11) is the negative of the relative entropy or KL divergence [90]. The second term on the right hand side of (C.11) can be simplified noting that $Q(\boldsymbol{Z})$ is a probability distribution and must sum to one. Using these two facts, (C.11) can be rewritten as:

$$F(Q(\boldsymbol{Z}), \boldsymbol{\theta}) = -\mathrm{KL}(Q(\boldsymbol{Z})||P(\boldsymbol{Z}|\boldsymbol{D}, \boldsymbol{\theta})) + \log P(\boldsymbol{D}|\boldsymbol{\theta}) . \tag{C.12}$$

As shown in [108], F can be maximized in a two-step process which is guaranteed to converge to at least a local maximum. The local maximum of F is also a local maximum of $\log P(\boldsymbol{D}|\boldsymbol{\theta})$. The two steps are:

E step: Holding parameters fixed, set $Q^{(t+1)}(\boldsymbol{Z})$ to maximize $F(Q(\boldsymbol{Z}), \boldsymbol{\theta}^{(t)})$.
M step: Holding Q fixed, set $\boldsymbol{\theta}^{(t+1)}$ to maximize $F(Q^{(t+1)}, \boldsymbol{\theta})$.

where $^{(t)}$ denotes the previous time step and $^{(t+1)}$ denote the current time step.

In the E step, since $P(\boldsymbol{Z}|\boldsymbol{D}, \boldsymbol{\theta})$ is easy to compute, no approximation to the posterior is necessary. $Q^{(t+1)}(\boldsymbol{Z})$ is simply taken to be $P(\boldsymbol{Z}|\boldsymbol{D}, \boldsymbol{\theta}^{(t)})$. This makes the KL divergence zero, insuring the E step does not cause F to decrease.

In the M step, F is maximized with respect to $\boldsymbol{\theta}$ using the form given in (C.9). Since the entropy does not depend on $\boldsymbol{\theta}$, the maximization is on just the expectation:

$$\mathbb{E}_{Q(\boldsymbol{Z})} = \sum_{\boldsymbol{Z}} Q(\boldsymbol{Z}) \log P(\boldsymbol{D}, \boldsymbol{Z}|\boldsymbol{\theta}) \,. \tag{C.13}$$

Expanding $\log P(\boldsymbol{D}, \boldsymbol{Z}|\boldsymbol{\theta})$ using (C.4):

$$\mathbb{E}_{Q(\boldsymbol{Z})} = \sum_{\boldsymbol{Z}} Q(\boldsymbol{Z}) \sum_{n=1}^{N} \sum_{s=1}^{|\boldsymbol{S}_n|} z_{ns} \left[\sum_{g=1}^{|\boldsymbol{S}_{ns}|} \log P(G_{S_{nsg}}|\boldsymbol{\theta}) \right] . \tag{C.14}$$

Only the terms where $z_{ns} = 1$ contribute to the summation. These terms are multiplied by $Q(\boldsymbol{Z})$ which is being held fixed at $P(\boldsymbol{Z}|\boldsymbol{D}, \boldsymbol{\theta}^{(t)})$:

$$\mathbb{E}_{Q(\boldsymbol{Z})} = \sum_{n=1}^{N} \sum_{s=1}^{|\boldsymbol{S}_n|} P(z_{ns}|\boldsymbol{D}_n, \boldsymbol{\theta}^{(t)}) \left[\sum_{g=1}^{|\boldsymbol{S}_{ns}|} \log P(G_{S_{nsg}}|\boldsymbol{\theta}) \right] . \tag{C.15}$$

The posterior probability of the latent variable is the responsibility (C.8) using the previous parameters $\boldsymbol{\theta}^{(t)}$:

$$P(z_{ns}|\boldsymbol{D}_n, \boldsymbol{\theta}^{(t)}) = \gamma^{(t)}(z_{ns}) = \frac{\prod_{g=1}^{|\boldsymbol{S}_{ns}|} P(G_{S_{nsg}}|\boldsymbol{\theta}^{(t)})}{\sum_{s=1}^{|\boldsymbol{S}_n|} \prod_{g=1}^{|\boldsymbol{S}_{ns}|} P(G_{S_{nsg}})|\boldsymbol{\theta}^{(t)})} . \tag{C.16}$$

Using the responsibility, the expectation is:

$$\mathbb{E}_{Q(\boldsymbol{Z})} = \sum_{n=1}^{N} \sum_{s=1}^{|\boldsymbol{S}_n|} \gamma^{(t)}(z_{ns}) \left[\sum_{g=1}^{|\boldsymbol{S}_{ns}|} \log P(G_{S_{nsg}}|\boldsymbol{\theta}) \right] . \tag{C.17}$$

Instead of summing over each graphone in a particular segmentation, the sum is taken over the I graphones in the inventory. If $C(G_i|\boldsymbol{S}_{ns})$ is the count of times the graphone G_i appeared in $\boldsymbol{S}_{ns}$:

$$\mathbb{E}_{Q(\boldsymbol{Z})} = \sum_{i=1}^{I} \sum_{n=1}^{N} \sum_{s=1}^{|\boldsymbol{S}_n|} \gamma^{(t)}(z_{ns}) C(G_i|\boldsymbol{S}_{ns}) \log P(G_i|\boldsymbol{\theta}) \,. \tag{C.18}$$

Maximization of this expectation with respect to the graphone probabilities uses the following lemma:

Lemma C.2.1. *To maximize* $\sum_i c_i \log p_i$ *with respect to the set of probabilities* $\{p_i\}$, *set* $p_i^* = \frac{c_i}{\sum_i c_i}$.

See appendix D.2 for a proof. The expectation (C.18) can be written as:

$$\mathbb{E}_{Q(\boldsymbol{Z})} = \sum_{i=1}^{I} c_i \log p_i \tag{C.19}$$

where

$$c_i = \sum_{n=1}^{N} \sum_{s=1}^{|\boldsymbol{S}_n|} \gamma^{(t)}(z_{ns}) C(G_i|\boldsymbol{S}_{ns}) \tag{C.20}$$

$$p_i = P(G_i|\boldsymbol{\theta}) \,. \tag{C.21}$$

Using lemma C.2.1, the expectation is maximized by setting the probability of a particular G_m to:

$$P^{(t+1)}(G_m) = \frac{c_m}{\sum_{i=1}^{I} c_i} \tag{C.22}$$

$$= \frac{\sum_{n=1}^{N} \sum_{s=1}^{|\boldsymbol{S}_n|} \gamma^{(t)}(z_{ns}) C(G_m|\boldsymbol{S}_{ns})}{\sum_{i=1}^{I} \sum_{n=1}^{N} \sum_{s=1}^{|\boldsymbol{S}_n|} \gamma^{(t)}(z_{ns}) C(G_i|\boldsymbol{S}_{ns})} \,. \tag{C.23}$$

Simplifying the denominator using the count of all graphones in a sequence $C(\boldsymbol{S}_{ns})$, the final reestimation formula is:

$$P^{(t+1)}(G_m) = \frac{\sum_{n=1}^{N} \sum_{s=1}^{|S_n|} \gamma^{(t)}(z_{ns}) C(G_m|\boldsymbol{S}_{ns})}{\sum_{n=1}^{N} \sum_{s=1}^{|S_n|} \gamma^{(t)}(z_{ns}) C(\boldsymbol{S}_{ns})} \,. \tag{C.24}$$

So at each step, a particular graphone's probability is updated based on how often that graphone appears in all segmentations of the training data. These appearances are weighted according to how likely each particular segmentation was.

C.2.1 Forward-Backward Training

Recall that dictionary entry $\boldsymbol{D}_n$ consisted of a letter sequence $\boldsymbol{X}_n$ and a phone sequence $\boldsymbol{Y}_n$. Let $|\boldsymbol{X}_n|$ and $|\boldsymbol{Y}_n|$ denote the number of letters and phones in $\boldsymbol{D}_n$ respectively. Let $(\boldsymbol{X}_n)_j^k$ and $(\boldsymbol{Y}_n)_j^k$ specify the letter/phone sequence starting at the j^{th} position and ending at the k^{th} position.

The probability of a dictionary entry can be written:

$$P(\boldsymbol{D}_n \mid \boldsymbol{\theta}) = P\left((\boldsymbol{X}_n)_1^{|\boldsymbol{X}_n|}, (\boldsymbol{Y}_n)_1^{|\boldsymbol{Y}_n|} \mid \boldsymbol{\theta}\right) . \tag{C.25}$$

The forward variable $\alpha_n(x, y)$ accounts for the probability from the start up to and *including* the x^{th} letter and y^{th} phone of the n^{th} dictionary entry:

$$\alpha_n(x, y) = P\left((\boldsymbol{X}_n)_1^x, (\boldsymbol{Y}_n)_1^y \mid \boldsymbol{\theta}\right) . \tag{C.26}$$

Similarly, the backwards variable $\beta_n(x, y)$ accounts for the probability of everything *after* the x^{th} letter and y^{th} phone of the n^{th} dictionary entry:

$$\beta_n(x, y) = P\left((\boldsymbol{X}_n)_{x+1}^{|\boldsymbol{X}_n|}, (\boldsymbol{Y}_n)_{y+1}^{|\boldsymbol{Y}_n|} \mid \boldsymbol{\theta}\right) . \tag{C.27}$$

Given the α and β variable formulation, all the following are equivalent:

$$P(\boldsymbol{D}_n \mid \boldsymbol{\theta}) = \alpha_n(|\boldsymbol{X}_n|, |\boldsymbol{Y}_n|) = \beta_n(0, 0) = \sum_{s=1}^{|\boldsymbol{S}_n|} P(S_{ns}|\boldsymbol{\theta}) . \tag{C.28}$$

Let $q_{\min}$ and $q_{\max}$ denote the minimum and maximum number of letters in the model being trained. Let $r_{\min}$ and $r_{\max}$ denote the minimum and maximum number of phones. The forward variable $\alpha_n(x, y)$ is recursively calculated using preceding α's that are a single graphone "hop" from location (x, y). These preceding α's are multiplied by the probability of the graphone used to hop to position (x, y).

So to be precise, the initial conditions are $\alpha_n(0,0) = 1$ and $\alpha_n(x,y) = 0$ if $x < 0$ or $y < 0$. For $0 \leq x \leq |\boldsymbol{X}_n|$ and $0 \leq y \leq |\boldsymbol{Y}_n|$ the forward variable is:

$$\alpha_n(x,y) = \sum_{q=q_{\min}}^{q_{\max}} \sum_{r=r_{\min}}^{r_{\max}} \alpha_n(x-q, y-r) P\left(G_{(x,y,q,r)} | \boldsymbol{\theta}\right) \tag{C.29}$$

where the graphone $G_{(x,y,q,r)}$ matches the preceding letters and phones:

$$\ell\left(G_{(x,y,q,r)}\right) = (\boldsymbol{X}_n)_{x-q+1}^{x} \tag{C.30}$$

$$\rho\left(G_{(x,y,q,r)}\right) = (\boldsymbol{Y}_n)_{y-r+1}^{y} . \tag{C.31}$$

The beta variable's initial conditions are $\beta_n(|\boldsymbol{X}_n|, |\boldsymbol{Y}_n|) = 1$ and $\beta_n(x,y) = 0$ if $x > |\boldsymbol{X}_n|$ or $y > |\boldsymbol{Y}_n|$. For $0 \leq x \leq |\boldsymbol{X}_n|$ and $0 \leq y \leq |\boldsymbol{Y}_n|$ the backward variable is:

$$\beta_n(x,y) = \sum_{q=q_{\min}}^{q_{\max}} \sum_{r=r_{\min}}^{r_{\max}} \beta_n(x+q, y+r) P\left(g_{(x,y,q,r)} | \boldsymbol{\theta}\right) \tag{C.32}$$

where the graphone $G_{(x,y,q,r)}$ matches the following letters and phones:

$$\ell\left(G_{(x,y,q,r)}\right) = (\boldsymbol{X}_n)_{x+1}^{x+q} \tag{C.33}$$

$$\rho\left(G_{(x,y,q,r)}\right) = (\boldsymbol{Y}_n)_{y+1}^{y+r} . \tag{C.34}$$

Note that for any letter and phone position (x, y) in a segmentation, the probability of the segmentation can be written as:

$$\prod_{g=1}^{|\boldsymbol{S}_{ns}|} P(G_{S_{nsg}} | \boldsymbol{\theta}^{(t)}) = \alpha_{ns}^{(t)}(x,y) \beta_{ns}^{(t)}(x,y) . \tag{C.35}$$

Equipped with the α and β variables, an efficient way to compute the reestimation formula can be found which avoids searching over all possible word segmentations. From (C.22), the quantity c_i contains the problematic search over segmentations:

$$c_i = \sum_{n=1}^{N} \sum_{s=1}^{|\boldsymbol{S}_n|} \gamma^{(t)}(z_{ns}) C(G_i|\boldsymbol{S}_{ns}) \,. \tag{C.36}$$

Focusing on just the n^{th} training example, the quantity needed is:

$$c_{in} = \sum_{s=1}^{|\boldsymbol{S}_n|} \gamma^{(t)}(z_{ns}) C(G_i|\boldsymbol{S}_{ns}) \,. \tag{C.37}$$

Rewriting the responsibility using (C.7) and pulling out the common denominator:

$$c_{in} = \frac{\sum_{s=1}^{|\boldsymbol{S}_n|} P(\boldsymbol{S}_{ns}|\boldsymbol{\theta}) C(G_i|\boldsymbol{S}_{ns})}{\sum_{s=1}^{|\boldsymbol{S}_n|} P(\boldsymbol{S}_{ns}|\boldsymbol{\theta})} \,. \tag{C.38}$$

Replacing the denominator with a form from (C.28):

$$c_{in} = \frac{\sum_{s=1}^{|\boldsymbol{S}_n|} P(\boldsymbol{S}_{ns}|\boldsymbol{\theta}) C(G_i|\boldsymbol{S}_{ns})}{\beta_n(0,0)} \,. \tag{C.39}$$

Using this form of c_{in} and expanding $P(\boldsymbol{S}_{ns}|\boldsymbol{\theta})$ with (6.8), the reestimation formula (C.24) is:

$$P^{(t+1)}(G_m) = \frac{\sum_{n=1}^{N} \frac{1}{\beta_n(0,0)} \sum_{s=1}^{|S_n|} C(G_m|\boldsymbol{S}_{ns}) \prod_{g=1}^{|\boldsymbol{S}_{ns}|} P(G_{S_{nsg}}|\boldsymbol{\theta})}{\sum_{n=1}^{N} \frac{1}{\beta_n(0,0)} \sum_{s=1}^{|S_n|} C(\boldsymbol{S}_{ns}) \prod_{g=1}^{|\boldsymbol{S}_{ns}|} P(G_{S_{nsg}}|\boldsymbol{\theta})} \,. \tag{C.40}$$

The summation of segmentation in (C.40) can now be simplified using the α and β variables. To see this, consider the five possible paths in the example in figure C.1. The only points in the lattice with non-zero α and β variables are points that are on one of the five paths. The probability of any path can be obtained by using (C.35) and a (x, y) pair on the path of interest.

If the sum of (C.35) is taken over all possible (x, y) pairs, each path's probability appears in the sum exactly as many times as the number of lattice points on that path:

$$\sum_{x=1}^{|\boldsymbol{X}_n|} \sum_{y=1}^{|\boldsymbol{Y}_n|} \alpha_n(x,y)\beta_n(x,y) = \sum_{s=1}^{|S_n|} C(\boldsymbol{S}_{ns}) \prod_{g=1}^{|\boldsymbol{S}_{ns}|} P(G_{S_{nsg}}|\boldsymbol{\theta}^{(t)}) \,. \tag{C.41}$$

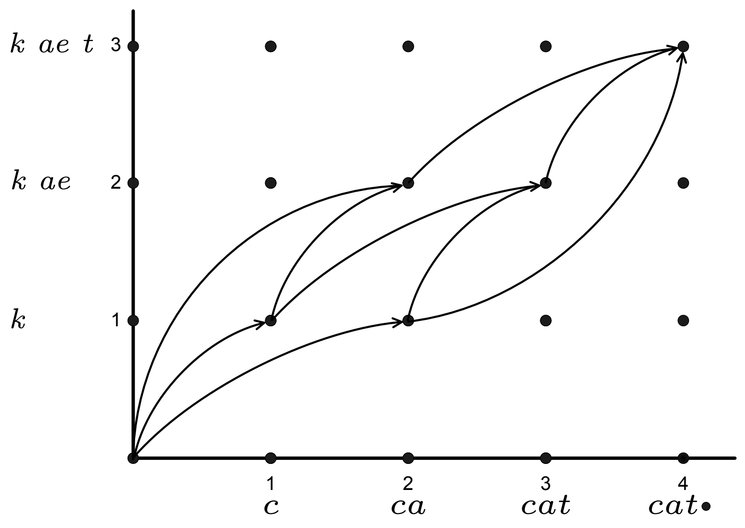


Figure C.1: The five possible paths through the lattice for the entry "cat" using a 1-2 model. A point in the lattice is reached by any sequence of graphones which generate exactly the letters and phones specified by that lattice position.

The above equation (C.41) allows replacement of the summation over segmentation in the denominator of the reestimation formula. In the case of the numerator, only instances of the graphone G_m are counted. To limit the summation to only this graphone, an indicator variable $\delta_{x,y,m}$ is introduced which is only non-zero if the letters and phones immediately preceding position (x, y) match graphone G_m:

$$\delta_{x,y,m} = \begin{cases} 1 & \text{if } \ \ell\left(G_m\right) = \left(\boldsymbol{X}_n\right)_{x-q+1}^{x} \text{ and } \rho\left(G_m\right) = \left(\boldsymbol{Y}_n\right)_{y-r+1}^{y} \\ 0 & \text{otherwise} \end{cases} \tag{C.42}$$

where graphone G_m has q letters and r phones.

Using the indicator variable and stopping the α term early to allow for an explicit $P^{(t)}(G_m)$ term, the replacement needed for the numerator is:

$$\sum_{s=1}^{|S_n|} C(G_m|\boldsymbol{S}_{ns}) \prod_{g=1}^{|\boldsymbol{S}_{ns}|} P(G_{S_{nsg}}|\boldsymbol{\theta}^{(t)}) = \sum_{x=1}^{|\boldsymbol{X}_n|} \sum_{y=1}^{|\boldsymbol{Y}_n|} \alpha_n(x-q, y-r) P^{(t)}(G_m) \beta_n(x, y) \delta_{x,y,m} \,. \tag{C.43}$$

Thus, the final reestimation formula is:

$$P^{(t+1)}(G_m) = \frac{\sum_{n=1}^{N} \frac{1}{\beta_n^{(t)}(0,0)} \sum_{x=1}^{|\boldsymbol{X}_n|} \sum_{y=1}^{|\boldsymbol{Y}_n|} \alpha_n^{(t)}(x-q, y-r) P^{(t)}(G_m) \beta_n^{(t)}(x,y) \delta_{x,y,m}}{\sum_{n=1}^{N} \frac{1}{\beta_n^{(t)}(0,0)} \sum_{x=1}^{|\boldsymbol{X}_n|} \sum_{y=1}^{|\boldsymbol{Y}_n|} \alpha_n^{(t)}(x,y) \beta_n^{(t)}(x,y)}. \tag{C.44}$$

Appendix D

Mathematical Details

D.1 Cumulative Distribution Error Bars

To compare cumulative distributions, it is useful to have error bars to indicate the uncertainty about each distribution. In this section, I describe the likelihood-based method I used to determine error bars for cumulative distributions.

For a cumulative distribution F, let F_x be the true probability at coordinate x with a random draw x_n satisfying $x_n \leq F_x$. The true distribution is not known, but we have observed a set of samples taken from this distribution. At x, A samples were above x and B were below x. The probability of this occurring is

$$Q(F_x) = P(A, B|F_x) \propto (F_x)^B (1 - F_x)^A \,. \tag{D.1}$$

To find error bars at x, we approximate $\ln Q$, the log of the posterior distribution over F_x, using Laplace's method. The second order Taylor-expansion of $\ln Q$ around a point $\hat{F}_x$ is

$$\ln Q(F_x) \simeq \ln Q(\hat{F}_x) + \tag{D.2}$$

$$\frac{\partial}{\partial F_x} \ln Q(F_x)|_{F_x=\hat{F}_x} \cdot (F_x - \hat{F}_x) + \tag{D.3}$$

$$\frac{\partial^2}{\partial F_x^2} \ln Q(F_x)|_{F_x=\hat{F}_x} \cdot \frac{(F_x - \hat{F}_x)^2}{2!} \,. \tag{D.4}$$

We choose $\hat{F}_x$ to be at $\ln Q$'s peak:

$$\frac{\partial}{\partial F_x} \ln Q(F_x)|_{F_x=\hat{F}_x} = 0\,. \tag{D.5}$$

Substituting D.1 in:

$$\frac{\partial}{\partial F_x} \left(\ln F_x^B + \ln(1-F_x)^A\right)\Big|_{F_x=\hat{F}_x} = 0\,. \tag{D.6}$$

Solving D.6 yields a point estimate for the cumulative distribution at x given observations A and B:

$$\hat{F}_x = \frac{B}{A+B}\,. \tag{D.7}$$

Given D.5, the Laplace approximation D.2 becomes:

$$\ln Q(F_x) \simeq \ln Q(\hat{F}_x) + \frac{\partial^2}{\partial F_x^2} \ln Q(F_x)|_{F_x=\hat{F}_x} \frac{(F_x-\hat{F}_x)^2}{2}\,. \tag{D.8}$$

Q is better approximated in the logit basis [101] where:

$$a_x = \ln \frac{F_x}{1-F_x} \tag{D.9}$$

$$F_x = \frac{1}{1+e^{-a_x}}\,. \tag{D.10}$$

In this basis, the log of our posterior distribution is:

$$\begin{aligned}
\ln Q(F_x) \;&\propto\; \frac{\partial^2}{\partial a_x^2} \ln Q\left(\frac{1}{1+e^{-a_x}}\right)\Bigg|_{a_x=\hat{a}_x} \cdot \frac{(a_x-\hat{a}_x)^2}{2} && \text{(D.11)}\\
&\propto\; \frac{\partial^2}{\partial a_x^2} \ln\left(\left(\frac{1}{1+e^{-a_x}}\right)^B \left(\frac{1}{1+e^{a_x}}\right)^A\right)\Bigg|_{a_x=\hat{a}_x} \cdot \frac{(a_x-\hat{a}_x)^2}{2} && \text{(D.12)}\\
&\propto\; \frac{-(A+B)\,e^{a_x}}{(1+e^{a_x})^2}\Bigg|_{a_x=\ln\left(\frac{\hat{F}_x}{1-\hat{F}_x}\right)} \cdot \frac{(a_x-\hat{a}_x)^2}{2} && \text{(D.13)}\\
&\propto\; \frac{-(A+B)\,e^{a_x}}{(1+e^{a_x})^2}\Bigg|_{a_x=\ln\left(\frac{B}{A}\right)} \cdot \frac{(a_x-\hat{a}_x)^2}{2} && \text{(D.14)}\\
&\propto\; \frac{-AB}{A+B} \cdot \frac{(a_x-\hat{a}_x)^2}{2}\,. && \text{(D.15)}
\end{aligned}$$

Q is thus an unnormalized Gaussian:

$$Q(F_x) \propto e^{\frac{-(a_x-\hat{a}_x)^2}{2\cdot\sigma_{\hat{a}}^2}} \tag{D.16}$$

where

$$\sigma_a^2 = \frac{A+B}{AB}\,. \tag{D.17}$$

Upper and lower z-sigma errors bars are at logit:

$$\ln\frac{B}{A} \pm z\sigma_a\,. \tag{D.18}$$

Or converting back from the logit basis:

$$\frac{1}{1+e^{\left(\ln\frac{A}{B}\pm z\sigma_a\right)}}\,. \tag{D.19}$$

D.2 Maximization Proof

Lemma D.2.1. *To maximize* $\sum_i c_i \log p_i$ *with respect to the set of probabilities* $\{p_i\}$*, set* $p_i^* = \frac{c_i}{\sum_i c_i}$.

Proof. The set $\{p_i\}$ is a probability distribution, so $p_i \geq 0$ and $\sum_i p_i = 1$. Using the sum to one constraint, maximization of $Q = \sum_i c_i \log p_i$ is done using a Lagrange multiplier:

$$L(Q,\lambda) = Q + \lambda\left(\left(\sum_i p_i\right) - 1\right) \tag{D.20}$$

$$\bigtriangledown_{p_i} L(Q,\lambda) = 0\,. \tag{D.21}$$

For the mth event:

$$\frac{\partial}{\partial p_m} L(Q,\lambda) = \frac{c_m}{p_m} + \lambda = 0 \tag{D.22}$$

$$p_m = \frac{c_m}{-\lambda}\,. \tag{D.23}$$

Using (D.23) and the sum to one constraint:

$$\sum_i \frac{c_i}{-\lambda} = 1\,. \tag{D.24}$$

Solving for λ in (D.23) and plugging into (D.24) yields:

$$p_m^* = \frac{c_m}{\sum_i c_i}\,. \tag{D.25}$$

□

Appendix E

Corpora Details

I used a variety of text and acoustic data sets during this thesis. Here I provide a brief description of the corpora and test sets I used.

E.1 English Gigaword Text Corpus

The English Gigaword corpus [63] consists of articles from the New York Times, Xinhua News Agency, Associated Press, and Agence France Press newswire services. The articles are from 1994 to 2002. After processing the original files in the corpus, I obtained a training set of 778M words. This corpus was used to train the language model in chapter 3.

E.2 CSR-III Text Corpus

The CSR-III text corpus [64] consists of articles from the Associated Press, San Jose Mercury, and the Wall Street Journal. The articles are from 1987 to 1991. After processing the original files in the corpus, I obtained a training set of 222M words. This corpus was used to train the language models used in chapter 2, 4, 5, and 6.

The corpus has a "setaside" directory containing 15M words of text from the same newswire sources. This text was not used during language model training. I drew target sentences from the set-aside sentences in the user studies reported in chapter 2, 4, and 5.

E.3 TIMIT Corpus

The TIMIT corpus [58] is a collection of US-English audio recording. The corpus has a total of 6300 utterances from 630 speakers (approximately 12 hours of audio). Unlike most other acoustic training corpora, the TIMIT corpus has time-aligned phonetic transcriptions. I used these transcriptions to initialize the monophone models in my HTK training recipe [147]. This recipe was used during the training of the HTK acoustic models in chapter 2 and 6.

E.4 WSJ0 and WSJ1 Corpora

The WSJ0 and WSJ1 corpora [5; 57; 118] contain US-English recording of sentences primarily from the Wall Street Journal (WSJ). This is the largest publicly available data for training acoustic models on dictation-style audio. The corpus contains recording from a Sennheiser microphone and from a secondary microphone. In this thesis, I only used the Sennheiser recordings.

I used two training subsets of the WSJ corpora in this thesis:

- **SI-284** – This training set combines the WSJ0 SI-84 training set (84 speakers with about 100 utterances per speaker) and the WSJ1 SI-200 training set (200 speakers with about 100 utterances per speaker). This training set has about 66 hours of audio from the “short-term” speakers. I used this set to train the acoustic model in chapter 3.
- **wsj_all** – In experiments with my training recipe [147], I found that using training data from all speakers (short-term, long-term, and journalists) provided the best accuracy. This training set contained 211 hours of audio data. I used this set to train the acoustic models in chapter 4, 5, and 6.

E.4 WSJ0 and WSJ1 Corpora

I used a variety of test sets from the WSJ corpora. I list test sets by the corpus and subdirectory in which they are found. Where applicable, I list alternative names used to refer to a test set (in this thesis and elsewhere).

- **WSJ1 si_dt_s2 / Dev CSR Spoke 2** – Read sentences from the San Jose Mercury newspaper and utterances in the ATIS (Air Travel Information Service) domain. In this thesis, I used only the San Jose Mercury portion of the test set. This resulted in a test set of 207 sentences from 10 speakers.
- **WSJ1 si_dt_20 / Dev CSR Hub 1** – Read WSJ sentences that have a vocabulary size of 64K. I removed 22 sentences that had verbalized punctuation. This yielded 491 sentences from 10 speakers.
- **WSJ1 si_dt_05 / Dev CSR Hub 2** – Read WSJ sentences that have a vocabulary size of 5K. This set contained 513 sentences from 10 speakers.
- **WSJ1 si_et_s2 / Nov'93 CSR Spoke 2** – Read sentences from the San Jose Mercury newspaper. This set contained 214 sentences from 10 speakers.
- **WSJ1 si_et_h1 / Nov'93 CSR Hub 1** – Read WSJ sentences that have a vocabulary size of 64K. This set contained 213 sentences from 10 speakers.
- **WSJ1 si_et_h2 / Nov'93 CSR Hub 2** – Read WSJ sentences that have a vocabulary size of 5K. This set contained 215 sentences from 10 speakers.
- **WSJ0 si_dt_20** – Read WSJ sentences that have a vocabulary size of 64K. This set contained 403 sentences from 10 speakers.
- **WSJ0 si_dt_05** – Read WSJ sentences that have a vocabulary size of 5K. This set contained 410 sentences from 10 speakers.
- **WSJ0 si_et_05 / Nov'92** – Read WSJ sentences that have a vocabulary size of 5K. This set contained 330 sentences from 8 speakers.
- **WSJ0 si_dt_jr** – Spontaneously dictated sentences recorded by journalists. This set contained 320 sentences from 4 speakers.

E.5 WSJCAM0 Corpus

The WSJCAM0 corpus [124; 125] contains UK-English recording of primarily Wall Street Journal sentences. The corpus contains about 16 hours of acoustic training data. I used this corpus to train the UK-English acoustic models used in chapter 2, 4 and 5. In chapter 2, I used the si_dt_5b test set from this corpus. This test set consists of read WSJ sentences using a 5K vocabulary (374 sentences from 20 speakers).

References

[1] Dogpile SearchSpy. http://www.dogpile.com/dogpile/ws/searchspy/rfcid=4101/rfcp=InternalNavigation/_iceUrlFlag=11?_IceUrl=true. Accessed December 8, 2008. 162, 192, 199

[2] GStreamer: Open source multimedia framework. http://gstreamer.freedesktop.org/. Accessed December 8, 2008. 125

[3] Xentec NV, home of Vox Studio. http://www.xentec.be/index.htm. Accessed June 18th, 2009. 48

[4] Yahoo! developer network home. http://developer.yahoo.com. Accessed December 8, 2008. 190

[5] CSR-II (WSJ1) complete. http://www.ldc.upenn.edu/Catalog/CatalogEntry.jsp?catalogId=LDC94S13A, 1994. Linguistic Data Consortium, Philadelphia, PA, USA. Accessed June 5th, 2009. 13, 38, 46, 81, 124, 183, 252

[6] H. Abdi. Bonferroni and Šidák corrections for multiple comparisons. In *Encyclopedia of Measurement and Statistics*. Sage, 2007. 21

[7] J. Accot and S. Zhai. More than dotting the i's — foundations for crossing-based interfaces. In *CHI '02: Proceedings of the SIGCHI Conference on Human factors in Computing Systems*, pages 73–80. ACM, 2002. 117

[8] M. J. Adamson and I. Damper. A recurrent network that learns to pronounce English text. In *Proceedings of the International Conference on Spoken Language Processing*, pages 1704–1707, October 1996. 205

[9] C. K. Akman. A computer aided transcription tool. Master's thesis, Boğazicçi University, Istanbul, Turkey, January 2007. 106

[10] F. Alleva, X. Huang, M.-Y. Hwang, and L. Jiang. Can continuous speech recognizers handle isolated speech? In *Proceedings of European Conference on Speech Communication and Technology*, pages 911–914, 1997. 55

[11] X. L. Aubert. A brief overview of decoding techniques for large vocabulary continuous speech recognition. In *ASR-2000*, pages 91–97, September 2000. 225

[12] I. Bazzi and J. Glass. Learning units for domain-independent out-of-vocabulary word modeling. In *Proceedings of European Conference on Speech Communication and Technology*, pages 61–64, September 2001. 205

[13] L. Bell and J. Gustafson. Repetition and its phonetic realizations: Investigating a swedish database of spontaneous computer directed speech. In *Proceedings of ICPhS*, pages 1221–1224, 1999. 35, 60

[14] T. C. Bell, J. G. Cleary, and I. H. Witten. *Text Compression*. Prentice Hall, NJ, 1990. 79, 174

[15] R. Bencina and P. Burk. PortAudio - an API for portable real-time audio. In *Audio Anecdotes*, pages 361–368, 2004. 12

[16] M. Bisani and H. Ney. Investigations on joint-multigram models for grapheme-to-phoneme conversion. *Proceedings of the International Conference on Spoken Language Processing*, pages 105–108, September 2002. 163, 173, 204

[17] M. Bisani and H. Ney. Bootstrap estimates for confidence intervals in ASR performance evaluation. *Proceedings of the IEEE Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 409–411, May 2004. 46, 127, 131, 172, 192

[18] M. Bisani and H. Ney. Open vocabulary speech recognition with flat hybrid models. *Proceedings of European Conference on Speech Communication and Technology*, pages 725–728, September 2005. 172, 179, 181, 184, 205

[19] M. Bisani and H. Ney. Joint-sequence models for grapheme-to-phoneme conversion. *Speech Communications*, 50(5):434–451, 2008. 171, 178, 204, 205

[20] M. Bisani and H. Ney. Sequitur G2P – a trainable grapheme-to-phoneme converter. `http://www-i6.informatik.rwth-aachen.de/web/Software/g2p.html`, December 2008. Accessed June 5th, 2009. 178

[21] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer-Verlag New York, Inc., 2006. 237

[22] P. Boersma and D. Weenink. Praat: Doing phonetics by computer. `http://www.praat.org/`. Accessed December 8, 2008. 41

[23] T. Brants and A. Franz. Web 1T 5-gram version 1. `http://www.ldc.upenn.edu/Catalog/CatalogEntry.jsp?catalogId=LDC2006T13`, 2006. Linguistic Data Consortium, Philadelphia, PA, USA. Accessed June 5th, 2009. 191

[24] M. Burke, B. Amento, and P. Isenhour. Error correction of voice mail transcripts in SCANMail. In *CHI '06: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 339–348, 2006. 28, 32

[25] Carnegie Mellon Speech Group. The CMU pronouncing dictionary. `http://www.speech.cs.cmu.edu/cgi-bin/cmudict`. Accessed January 6th, 2009. 12, 46, 161, 184

[26] Center for Lexical Information, Max Planck Institute for Psycholinguistics. CELEX lexical database of English (version 2.5). `http://www.ldc.upenn.edu/Catalog/CatalogEntry.jsp?catalogId=LDC96L14`. Accessed April 21st, 2009. 204

[27] L. L. Chase. *Error-responsive feedback mechanisms for speech recognizers*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, USA, 1997. 10

[28] S. F. Chen. Conditional and joint models for grapheme-to-phoneme conversion. *Proceedings of European Conference on Speech Communication and Technology*, pages 2033–2036, September 2003. 172, 175, 178, 204

[29] S. F. Chen and J. Goodman. An empirical study of smoothing techniques for language modeling. In *Proceedings of the 34th annual meeting on Association for Computational Linguistics*, pages 310–318, Morristown, NJ, USA, 1996. Association for Computational Linguistics. 224

[30] S. F. Chen and J. T. Goodman. An empirical study of smoothing techniques for language modeling. Technical report, Computer Science Group, Harvard University, 1998. 174

[31] S. Choularton. Investigating the acoustic sources of speech recognition errors. `http://www.ics.mq.edu/au/~stephenc/inter2005.pdf`. Accessed 2005. 59

[32] J. Cleary and I. Witten. Data compression using adaptive coding and partial string matching. *IEEE Transactions on Communications*, 32(4):396–402, April 1984. 78

[33] J. Cohen. Embedded speech recognition applications in mobile phones: status, trends and challenges. *Proceedings of the IEEE Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 5352–5355, 2008. 110

[34] R. A. Cole. Survey of the state of the art in human language technology. `http://cslu.cse.ogi.edu/HLTsurvey/`, 1996. Accessed April 26th, 2009. 215

[35] C. Collins, S. Carpendale, and G. Penn. Visualization of uncertainty in lattices to support decision-making. In *Proceedings of Eurographics/IEEE VGTC Symposium on Visualization*, pages 51–58, Norrköping, Sweden, 2007. 33

[36] A. Cox and A. Walton. Evaluating the viability of speech recognition for mobile text entry. In *Proceedings of HCI 2004: Design for Life*, pages 25–28, 2004. 156

[37] A. Crossan, R. Murray-Smith, S. Brewster, J. Kelly, and B. Musizza. Gait phase effects in mobile interaction. In *CHI '05: Extended abstracts on Human Factors in Computing Systems*, pages 1312–1315. ACM, 2005. 122

[38] W. Daelemans, J. Zavrel, K. van der Sloot, and A. van den Bosch. TiMBL: Tilburg memory-based learner. `http://ilk.uvt.nl/timbl/`. Accessed December 8, 2008. 206

[39] R. I. Damper and J. F. G. Eastmond. Pronouncing text by analogy. In *Proceedings of the 16th Conference on Computational Linguistics*, pages 268–273, Morristown, NJ, USA, 1996. Association for Computational Linguistics. 205

[40] J. J. Darragh, I. H. Witten, and M. L. James. The reactive keyboard: A predictive typing aid. *Computer*, 23(11):41–49, 1990. 123

[41] S. B. Davis and P. Mermelstein. Comparison of parametric representations for monosyllable word recognition in continuously spoken sentences. *IEEE Transactions on Speech and Audio Processing*, 28(4):357–366, 1980. 216

[42] B. Decadt, J. Duchateau, W. Daelemans, and P. Wambacq. Transcription of out-of-vocabulary words in large vocabulary speech recognition based on phoneme-to-grapheme conversion. *Proceedings of the IEEE Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 861–864, May 2002. 205

[43] S. Deligne and F. Bimbot. Inference of variable-length linguistic and acoustic units by multigrams. *Speech Communication*, 23(3):223–241, 1997. 163, 203, 237

[44] S. Deligne, F. Yvon, and F. Bimbot. Variable-length sequence matching for phonetic transcription using joint multigrams. *Proceedings of European Conference on Speech Communication and Technology*, pages 2243–2246, 1995. 163, 203, 237

[45] E. Devine, S. Gaehde, and A. Curtis. Comparative evaluation of three continuous speech recognition software packages in the generation of medical reports. *Journal of the American Medical Informatics Association*, 7:462–468, 2000. 8, 13

[46] H. Elovitz, R. Johnson, A. McHugh, and J. Shore. Letter-to-sound rules for automatic translation of English text to phonetics. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 24(6):446–459, 1976. 205

[47] T. Endo, N. Ward, and M. Terada. Can confidence scores help users post-editing speech recognizer output? In *International Conference on Spoken Language Processing*, pages 1469–1472, Denver, CO, 2002. 8, 33

[48] T. Fabian, R. Lieb, G. Ruske, and M. Thomae. Impact of word graph density on the quality of posterior probability based confidence measures. In *Proceedings of European Conference on Speech Communication and Technology*, pages 917–920, September 2003. 30

[49] J. Feng and A. Sears. Using confidence scores to improve hands-free speech based navigation in continuous dictation systems. *ACM Transactions on Computer Human Interaction*, 11(4):329–356, 2004. 33

[50] M. Finke and T. Zeppenfeld. LVCSR switchboard April 1996 evaluation report. In *Proceedings of the LVCSR Hub 5 Workshop*, April 1996. 11

[51] A. Fischer, K. Price, and A. Sears. Speech-based text entry for mobile handheld devices: An analysis of efficacy and error correction techniques for server-based solutions. *International Journal of Human-Computer Interaction*, 19(3):279–304, 2006. 155

[52] P. Fitts. The information capacity of the human motor system in controlling the amplitude of movement. *Journal of Experimental Psychology*, 47(6):381–391, 1954. 11, 117

[53] A. Franz and B. Milch. Searching the web by voice. In *Proceedings of the Conference on Computational Linguistics*, pages 1213–1217, 2002. 195, 206

[54] M. J. F. Gales. The generation and use of regression class trees for MLLR adaptation. Technical Report CUED/F-INFENG/TR263, Cambridge University, 1996. 221

[55] L. Galescu. Recognition of out-of-vocabulary words with sub-lexical language models. In *Proceedings of European Conference on Speech Communication and Technology*, pages 249–252, September 2003. 205

[56] L. Galescu and J. Allen. Bi-directional conversion between graphemes and phonemes using a joint n-gram model. *Proceedings of the 4th ISCA Tutorial and Research Workshop on Speech Synthesis*, 2001. 171, 178, 203

[57] J. Garofalo, D. Graff, D. Paul, and D. Pallett. CSR-I (WSJ0) complete. `http://www.ldc.upenn.edu/Catalog/CatalogEntry.jsp?catalogId=LDC93S6A`, 1994. Linguistic Data Consortium, Philadelphia, PA, USA. Accessed June 5th, 2009. 46, 81, 124, 183, 252

[58] J. S. Garofolo, L. F. Lamel, W. M. Fisher, J. G. Fiscus, D. S. Pallett, N. L. Dahlgren, and V. Zue. TIMIT acoustic-phonetic continuous speech corpus. `http://www.ldc.upenn.edu/Catalog/CatalogEntry.jsp?catalogId=LDC93S1`, 1993. Linguistic Data Consortium, Philadelphia, PA, USA. Accessed June 5th, 2009. 46, 183, 252

[59] J. Gauvain and C. Lee. Maximum a posteriori estimation for multivariate gaussian mixture observations of Markov chains. *IEEE Transactions on Speech and Audio Processing*, 2(2):291–298, April 1994. 13, 81, 124, 222

[60] C. Gollan, M. Bisani, S. Kanthak, R. Schlüter, and H. Ney. Cross domain automatic transcription on the TC-STAR EPPS corpus. *Proceedings of the IEEE Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 825–828, March 2005. 206

[61] I. J. Good. The population frequencies of species and the estimation of population parameters. *Biometrika*, 40(3 and 4):237–264, 1953. 174

[62] J. Goodman, G. Venolia, K. Steury, and C. Parker. Language modeling for soft keyboards. In *IUI '02: Proceedings of the 7th International Conference on Intelligent User Interfaces*, pages 194–195. ACM, 2002. 123

[63] D. Graff. English gigaword corpus. `http://www.ldc.upenn.edu/Catalog/CatalogEntry.jsp?catalogId=LDC2003T05`, 2003. Linguistic Data Consortium, Philadelphia, PA, USA. Accessed June 5th, 2009. 251

[64] D. Graff, R. Rosenfeld, and D. Pau. CSR-III text. `http://www.ldc.upenn.edu/Catalog/CatalogEntry.jsp?catalogId=LDC95T6`, 1995. Lin-

guistic Data Consortium, Philadelphia, PA, USA. Accessed June 5th, 2009. 12, 47, 118, 125, 251

[65] D. Hakkani-Tür, F. Béchet, G. Riccardi, and G. Tur. Beyond ASR 1-best: Using word confusion networks in spoken language understanding. *Journal of Computer Speech and Language*, 20(4):495–514, 2006. 30, 114

[66] C. A. Halverson, D. B. Horn, C.-M. Karat, and J. Karat. The beauty of errors: Patterns of error correction in desktop speech systems. In *Proceedings of INTERACT*, pages 133–140, 1999. 35, 61, 68

[67] I. L. Hetherington. PocketSUMMIT: small footprint continuous speech recognition. In *Proceedings of the International Conference on Spoken Language Processing*, pages 1465–1468, August 2007. 110

[68] J. Hirschberg, D. Litman, and M. Swerts. Prosodic cues to recognition errors. In *Proceedings of the Automatic Speech Recognition and Understanding Workshop (ASRU'99)*, pages 349–352, 1999. 59

[69] J. Hirschberg, D. Litman, and M. Swerts. Generalizing prosodic prediction of speech recognition errors. In *In Proceedings of the 6th International Conference of Spoken Language Processing*, pages 254–257, 2000. 59

[70] J. Hirschberg, D. Litman, and M. Swerts. Prosodic and other cues to speech recognition failures. *Speech Communication*, 43(1-2):155–175, June 2004. 59

[71] J. Hirschberg, D. Litman, and M. Swerts. Characterizing and predicting corrections in spoken dialogue systems. *Computational Linguistics*, 32(3):417–438, 2006. 59

[72] P. G. Howard. *The design and analysis of efficient lossless data compression systems*. PhD thesis, Brown University, 1993. 79

[73] X. D. Huang and M. A. Jack. Semi-continuous hidden markov models for speech signals. *Computer Speech and Language*, 8(3):239–252, 1989. 221

[74] D. Huggins-Daines, M. Kumar, A. Chan, A. W. Black, M. Ravishankar, and A. I. Rudnicky. PocketSphinx: A free, real-time continuous speech recognition system for hand-held devices. In *Proceedings of the IEEE Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 185–188, May 2006. 81, 110, 124, 126, 199

[75] D. Huggins-Daines and A. I. Rudnicky. Interactive ASR error correction for touchscreen devices. In *Proceedings of ACL-8: HLT Demo Session*, pages 17–19. ACL, 2008. 106, 155

[76] F. Jelinek. *Statistical Methods for Speech Recognition*. The MIT Press, January 1998. 215

[77] H. Jiang. Confidence measures for speech recognition: A survey. *Speech Communication*, 45(4):455–470, 2005. 30

[78] D. Jurafsky and J. H. Martin. *Speech and Language Processing*. Prentice Hall, second edition, May 2008. 215

[79] M. Kamvar and S. Baluja. Deciphering trends in mobile search. *IEEE Computer*, 40(8):58–62, 2007. 5, 191, 202, 212

[80] C.-M. Karat, C. Halverson, D. Horn, and J. Karat. Patterns of entry and correction in large vocabulary continuous speech recognition systems. In *CHI '99: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 568–575. ACM, 1999. 8, 35, 61, 62, 65, 68, 107, 109, 112

[81] J. Karat, D. Horn, C. Halverson, and C. Karat. Overcoming unusability: Developing efficient strategies in speech recognition systems. In *CHI '00: Extended abstracts on Human Factors in Computing Systems*, pages 141–142. ACM, 2000. 13

[82] A. K. Karlson, B. B. Bederson, and J. L. Contreras-Vidal. Understanding one-handed use of mobile devices. In J. Lumsden, editor, *Handbook of Research on User Interface Design and Evaluation for Mobile Technology*, pages 86–100. Idea Group, 2008. 113

[83] E. Karpov, I. Kiss, J. Leppnen, J. Olsen, D. Oria, S. Sivadas, and J. Tian. Short message dictation on symbian series 60 mobile phones. In *Workshop on Speech in Mobile and Pervasive Environments*, pages 126–127, 2006. 155

[84] T. Kemp and T. Schaaf. Estimating confidence using word lattices. In *Proceedings of European Conference on Speech Communication and Technology*, pages 827–830, 1997. 11

[85] R. Kneser and H. Ney. Improved backing-off for m-gram language modeling. In *Proceedings of the IEEE Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 181–184, 1995. 174

[86] H. H. Koester. Usage, performance, and satisfaction outcomes for experienced users of automatic speech recognition. *Journal of Rehabilitation Research and Development*, 41(5):739–755, September 2004. 8, 13, 35, 62, 65, 107

[87] T. W. Kohler, C. Fugen, S. Stker, and A. Waibel. Rapid porting of ASR-systems to mobile devices. In *Proceedings of the International Conference on Spoken Language Processing*, pages 233–236, September 2005. 110

[88] P. O. Kristensson and S. Zhai. Relaxing stylus typing precision by geometric pattern matching. In *IUI '05: Proceedings of the 10th International Conference on Intelligent User Interfaces*, pages 151–158. ACM, 2005. 123

[89] P. O. Kristensson and S. Zhai. Improving word-recognizers using an interactive lexicon with active and passive words. In *IUI '08: Proceedings of the 13th International Conference on Intelligent User Interfaces*, pages 353–356, New York, NY, USA, 2008. ACM. 112

[90] S. Kullback and R. A. Leibler. On information and sufficiency. *Annals of Mathematical Statistics*, 22(1):79–86, 1951. 239

[91] K. Kurihara, M. Goto, J. Ogata, and T. Igarashi. Speech pen: Predictive handwriting based on ambient multimodal recognition. In *CHI '06: Proceedings of the SIGCHI Conference on Human factors in Computing Systems*, pages 851–860. ACM, 2006. 106, 117, 154

[92] J. Lai and J. Vergo. MedSpeak: Report creation with continuous speech recognition. In *CHI '97: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 431–438. ACM, 1997. 8

[93] K. Larson and D. Mowatt. Speech error correction: The story of the alternates list. *International Journal of Speech Technology*, pages 183–194, 2003. 65, 108, 109

[94] LDC. CSR-III text corpus. `http://www.ldc.upenn.edu/Catalog/CatalogEntry.jsp?catalogId=LDC95T6`. Accessed December 8, 2008. 129

[95] C. J. Leggetter and P. C. Woodland. Flexible speaker adaptation using maximum likelihood linear regression. In *Proceedings of the ARPA Spoken Language Technology Workshop*, pages 110–115, 1995. 221

[96] C. J. Leggetter and P. C. Woodland. Maximum likelihood linear regression for speaker adaptation of continuous density hidden Markov models. *Computer Speech and Language*, pages 171–185, 1995. 12, 57, 81, 124, 221

[97] G.-A. Levow. Characterizing and recognizing spoken corrections in human-computer dialogue. In *COLING-ACL*, pages 736–742, 1998. 35, 58

[98] X. H. Li Jiang, Hsiao-Wuen Hon. Improvements on a trainable letter-to-sound converter. In *Proceedings of European Conference on Speech Communication and Technology*, pages 605–608, September 1997. 205

[99] F.-H. Liu, R. M. Stern, X. Huang, and A. Acero. Efficient cepstral normalization for robust speech recognition. In *Proceedings of ARPA Human Language Technology Workshop*, pages 69–74, March 1993. 82, 125, 217

[100] J. Lööf, C. Gollan, S. Hahn, G. Heigold, B. Hoffmeister, C. Plahl, D. Rybach, R. Schlüter, and H. Ney. The RWTH 2007 TC-STAR evaluation system for european english and spanish. In *Proceedings of the International Conference on Spoken Language Processing*, pages 2145–2148, August 2007. 206

[101] D. J. C. MacKay. Choice of basis for Laplace approximation. *Machine Learning*, 33(1):77–86, 1998. 248

[102] I. S. MacKenzie and R. W. Soukoreff. Text entry for mobile computing: Models and methods, theory and practice. *Human-Computer Interaction*, 17:147–198, 2002. 109

[103] I. S. MacKenzie and S. X. Zhang. The design and evaluation of a high-performance soft keyboard. In *CHI '99: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 25–31, New York, NY, USA, 1999. ACM. 157

[104] L. Mangu, E. Brill, and A. Stolcke. Finding consensus in speech recognition: Word error minimization and other applications of confusion networks. *Computer Speech and Language*, 14(4):373–400, 2000. 9, 105, 161, 181, 226

[105] A. Martin, G. Doddington, T. Kamm, M. Ordowski, and M. Przybocki. The DET curve in assessment of detection task performance. In *Proceedings of European Conference on Speech Communication and Technology*, pages 1895–1898, Rhodes, Greece, 1997. 30

[106] Microsoft. Speech software development kit 5.1. `http://www.microsoft.com/downloads/details.aspx?FamilyID=5e86ec97-40a7-453f-b0ee-6583171b4530`. Accessed January 6th, 2009. 46

[107] A. Moffat. Implementing the PPM data compression scheme. *IEEE Transactions on Communications*, 38(11):1917–1921, November 1990. 79

[108] R. M. Neal and G. E. Hinton. A new view of the EM algorithm that justifies incremental, sparse and other variants. In M. I. Jordan, editor, *Learning in Graphical Models*, pages 355–368. Kluwer Academic Publishers, 1998. 239, 240

[109] H. Ney, U. Essen, and R. Kneser. On structuring probabilistic dependencies in stochastic language modelling. *Computer Speech and Language*, 8:1–38, 1994. 174

[110] M. Novak. Towards large vocabulary ASR on embedded platforms. In *Proceedings of the International Conference on Spoken Language Processing*, pages 2309–2312, October 2004. 110

[111] Nuance. Dragon NaturallySpeaking speech recognition software. http://www.nuance.com/naturallyspeaking/. Accessed April 27th, 2009. 46

[112] J. Ogata and M. Goto. Speech repair: Quick error correction just by using selection operation for speech input interfaces. In *Proceedings of the International Conference on Spoken Language Processing*, pages 133–136, September 2005. 154

[113] J. Olsen, Y. Cao, G. Ding, and X. Yang. A decoder for large vocabulary continuous short message dictation on embedded devices. *Proceedings of the IEEE Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 4337–4340, March 2008. 125, 154

[114] A. Oulasvirta, S. Tamminen, V. Roto, and J. Kuorelahti. Interaction in 4-second bursts: the fragmented nature of attentional resources in mobile HCI. In *CHI '05: Proceedings of the SIGCHI Conference on Human factors in Computing Systems*, pages 919–928, New York, NY, USA, 2005. ACM Press. 113, 140

[115] S. Oviatt. Modeling hyperarticulate speech during human-computer error resolution. In *Proceedings of the International Conference on Spoken Language Processing*, pages 797–800, 1996. 35, 58

[116] S. Oviatt, P. Cohen, L. Wu, J. Vergo, L. Duncan, B. Suhm, J. Bers, T. Holzman, T. Winograd, J. Landay, J. Larson, and D. Ferro. Designing the user interface for multimodal speech and gesture applications: State-of-the-art systems and future research directions. *Human-Computer Interaction*, 15:253–322, 2000. 112

[117] S. Oviatt, M. MacEachern, and G.-A. Levow. Predicting hyperarticulate speech during human-computer error resolution. *Speech Communication*, 24(2):87–110, 1998. 36, 59

[118] D. B. Paul and J. M. Baker. The design for the Wall Street Journal-based CSR corpus. In *HLT '91: Proceedings of the Workshop on Speech and Natural Language*, pages 357–362. Association for Computational Linguistics, 1992. 13, 129, 252

[119] W. Press, S. Teukolsky, W. Vetterling, and B. Flannery. *Numerical Recipes in C*. Cambridge University Press, Cambridge, UK, 1992. 178, 204

[120] J. Price, M. Lin, J. Feng, R. Goldman, A. Sears, and A. Jacko. Motion does matter: An examination of speech-based text entry on the move. *Universal Access in the Information Society*, 4(3):246–257, 2006. 129, 156

[121] L. R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989. 218

[122] B. Reeves and C. Nass. *The media equation: how people treat computers, television, and new media like real people and places*. Cambridge University Press, New York, NY, USA, 1996. 29, 32

[123] T. Robinson. BEEP dictionary. `http://svr-www.eng.cam.ac.uk/comp.speech/Section1/Lexical/beep.html`, August 1996. Accessed June 5th, 2009. 12

[124] T. Robinson, J. Fransen, D. Pye, J. Foote, and S. Renals. WSJCAM0: A British English speech corpus for large vocabulary continuous speech recognition. In *Proceedings of the IEEE Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 81–84, Detroit, MI, 1995. 12, 254

[125] T. Robinson, J. Fransen, D. Pye, J. Foote, S. Renals, P. Woodland, and S. Young. WSJCAM0 cambridge read news. `http://www.ldc.upenn.edu/Catalog/CatalogEntry.jsp?catalogId=LDC95S24`, 1995. Linguistic Data Consortium, Philadelphia, PA, USA. Accessed June 5th, 2009. 124, 254

[126] C. Schmandt. The intelligent ear: A graphical interface to digital audio. In *Proceedings of the International Conference on Cybernetics and Society*, pages 393–397, Atlanta, GA, 1981. 31

[127] G. Schwarz. Estimating the dimension of a model. *Annals of Statistics*, 6:461–464, 1978. 176

[128] K. Seymore, S. Chen, S. Doh, M. Eskenazi, E. Gouvea, B. Raj, M. Ravishankar, R. Rosenfeld, M. Siegler, R. Stern, and E. Thayer. CMU Sphinx-3

English broadcast news transcription system. In *Proceedings of the 1998 DARPA Speech Recognition Workshop*, pages 55–59, 1998. 12

[129] J. Sherwani, D. Yu, T. Paek, M. Czerwinski, Y. C. Ju, and A. Acero. Voicepedia: Towards speech-based access to unstructured information. In *Proceedings of European Conference on Speech Communication and Technology*, pages 146–149, 2007. 195, 207

[130] B. Shneiderman. The limits of speech recognition. *Communications of the ACM*, 43(9):63–65, 2000. 110

[131] E. Shriberg, E. Wade, and P. Price. Human-machine problem solving using spoken language systems (SLS): Factors affecting performance and user satisfaction. In *Proceedings of the DARPA Speech and Natural Language Workshop*, pages 49–54, 1992. 35, 36, 40, 59, 63

[132] V. Siivola, T. Hirsimki, M. Creutz, and M. Kurimo. Unlimited vocabulary speech recognition based on morphs discovered in an unsupervised manner. In *Proceedings of European Conference on Speech Communication and Technology*, pages 2293–2296, 2006. 206

[133] H. Soltau and A. Waibel. On the influence of hyperarticulated speech on recognition performance. In *Proceedings of the International Conference on Spoken Language Processing*, pages 225–228, 1998. 35, 60, 63

[134] H. Soltau and A. Waibel. Phone dependent modeling of hyperarticulated effects. In *Proceedings of the International Conference on Spoken Language Processing*, pages 105–108, October 2000. 60, 63

[135] H. Soltau and A. Waibel. Specialized acoustic models for hyperarticulated speech. In *Proceedings of the IEEE Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 1779–1782, June 2000. 60, 63

[136] A. J. Stent, M. K. Huffman, and S. E. Brennan. Adapting speaking after evidence of misrecognition: Local and global hyperarticulation. *Speech Communications*, 50(3):163–178, 2008. 35, 60, 63

[137] A. Stolcke. Entropy-based pruning of backoff language models. In *Proceedings of DARPA Broadcast News Transcription and Understanding Workshop*, pages 270–274, 1998. 47, 82, 125, 223

[138] A. Stolcke. SRILM – an extensible language modeling toolkit. In *International Conference on Spoken Language Processing*, pages 901–904, Denver, CO, 2002. 12, 47, 128, 170, 183, 193

[139] B. Suhm, B. Myers, and A. Waibel. Multimodal error correction for speech user interfaces. *ACM Transactions on Computer-Human Interaction*, 8(1):60–98, 2001. 8, 15, 32, 107, 112, 155

[140] B. Suhm and A. Waibel. Exploiting repair context in interactive error recovery. In *Proceedings of the European Conference on Speech Communication and Technology*, pages 1659–1662, 1997. 55

[141] Z.-H. Tan and B. Lindberg. *Automatic Speech Recognition on Mobile Devices and over Communication Networks*. Springer Publishing Company, 2008. 153

[142] P. Taylor. Hidden Markov models for grapheme to phoneme conversion. In *Proceedings of the International Conference on Spoken Language Processing*, pages 1973–1976, September 2005. 205

[143] W. J. Teahan. *Modelling English Text*. PhD thesis, University of Waikato, 1997. 79

[144] O. Tuisku, P. Majaranta, P. Isokoski, and K.-J. Räihä. Now dasher! dash away! Longitudinal study of fast text entry by eye gaze. In *ETRA '08: Proceedings of the 2008 symposium on Eye tracking research & applications*, pages 19–26, 2008. 107

[145] S. Vermuri, P. DeCamp, W. Bender, and C. Schmandt. Improving speech playback using time-compression and speech recognition. In *CHI '04: Proceedings of the SIGCHI Conference on Human factors in Computing Systems*, pages 295–302. ACM, 2004. 32

[146] K. Vertanen. Efficient computer interfaces using continuous gestures, language models, and speech. Technical Report UCAM-CL-TR-627, Computer Laboratory, University of Cambridge, 2005. 67, 74, 75, 121

[147] K. Vertanen. Baseline WSJ acoustic models for HTK and Sphinx: Training recipes and recognition experiments. Technical report, Cavendish Laboratory, University of Cambridge, 2006. `http://www.inference.phy.cam.ac.uk/is/papers/baseline_wsj_recipes.pdf`. 12, 46, 47, 81, 124, 183, 252

[148] K. Vertanen. Speech and speech recognition during dictation corrections. In *Proceedings of the International Conference on Spoken Language Processing*, September 2006. 13, 36

[149] K. Vertanen. Combining open vocabulary recognition and word confusion networks. In *Proceedings of the IEEE Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 4325–4328, March 2008. 161

[150] K. Vertanen. Training, development and test set split of the CMU dictionary. `http://www.keithv.com/cmusplit/`, April 2009. Accessed June 4th, 2009. 172

[151] K. Vertanen and P. O. Kristensson. On the benefits of confidence visualization in speech recognition. In *CHI '08: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1497–1500. ACM, 2008. 9

[152] K. Vertanen and P. O. Kristensson. Parakeet: A continuous speech recognition system for mobile touch-screen devices. In *IUI '09: Proceedings of the* 14^{th} *International Conference on Intelligent User Interfaces*, pages 237–246. ACM, 2009. 111, 196

[153] K. Vertanen and P. O. Kristensson. Parakeet: A demonstration of speech recognition on a mobile touch-screen device. In *IUI '09: Proceedings of the 14th International Conference on Intelligent User Interfaces*, pages 483–484. ACM, 2009. 111

[154] K. Vertanen and P. O. Kristensson. Recognition and correction of voice web search queries. In *Proceedings of the International Conference on Spoken Language Processing*, pages 1863–1866, September 2009. 161

[155] E. Wade, E. Shriberg, and P. Price. User behaviors affecting speech recognition. In *Proceedings of the International Conference on Spoken Language Processing*, pages 995–998, 1992. 36, 60, 63

[156] D. Ward. *Adaptive Computer Interfaces*. PhD thesis, University of Cambridge, 2001. 107

[157] D. J. Ward, A. F. Blackwell, and D. J. C. MacKay. Dasher - a data entry interface using continuous gestures and language models. In *UIST '00: Proceedings of the 13th Annual ACM Symposium on User Interface software and technology*, pages 129–137. ACM Press, 2000. 65, 66

[158] D. J. Ward and D. J. C. MacKay. Fast hands-free writing by gaze direction. *Nature*, 418(6900):838, 2002. 66, 107

[159] F. Weng, A. Stolcke, and A. Sankar. Efficient lattice representation and generation. In *Proceedings of the International Conference on Spoken Language Processing*, pages 2531–2534, 1998. 74, 127, 128

[160] F. Wessel, R. Schltiter, K. Macherey, and H. Ney. Confidence measures for large vocabulary continuous speech recognition. *IEEE Transactions on Speech and Audio Processing*, 9:288–298, 2001. 30

[161] M. Wilson. MRC psycholinguistic database: Machine usable dictionary, version 2.00. `http://www.psy.uwa.edu.au/mrcdatabase/uwa_mrc.htm`, 1987. Rutherford Appleton Laboratory, Oxfordshire, England. Accessed June 5th, 2009. 41

[162] J. O. Wobbrock, D. H. Chau, and B. A. Myers. An alternative to push, press, and tap-tap-tap: Gesturing on an isometric joystick for mobile phone text entry. In *CHI '07: Proceedings of the SIGCHI Conference on Human factors in Computing Systems*, pages 667–676. ACM, 2007. 4, 149, 157, 212

[163] P. C. Woodland. Speaker adaptation for continuous density HMMs: A review. In *ITRW on Adaptation Methods for Speech Recognition*, pages 11–19, August 2001. 222

[164] P. C. Woodland, C. J. Leggetter, J. J. Odell, V. Valtchev, and S. J. Young. The 1994 HTK large vocabulary speech recognition system. In *Acoustics, Speech, and Signal Processing, 1995. ICASSP-95., 1995 International Conference on*, volume 1, pages 73–76, May 1995. 228

[165] P. C. Woodland, J. J. Odell, V. Valtchev, and S. J. Young. Large vocabulary continuous speech recognition using HTK. *Proceedings of the IEEE Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2:125–128, April 1994. 47

[166] A. Yazgan and M. Saraclar. Hybrid language models for out of vocabulary word detection in large vocabulary conversational speech recognition. *Proceedings of the IEEE Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 745–748, May 2004. 205

[167] S. J. Young, G. Evermann, M. J. F. Gales, T. Hain, D. Kershaw, X. Liu, G. Moore, J. J. Odell, D. Ollason, D. Povey, V. Valtchev, and P. C. Woodland. HTK version 3.4. `http://htk.eng.cam.ac.uk/ftp/software/HTK-3.4.zip`, March 2007. Accessed June 4th, 2009. 47, 183

[168] S. J. Young, G. Evermann, M. J. F. Gales, T. Hain, D. Kershaw, X. Liu, G. Moore, J. J. Odell, D. Ollason, D. Povey, V. Valtchev, and P. C. Woodland. *The HTK Book (for HTK Version 3.4)*. University of Cambridge, March 2009. 47, 183

[169] S. J. Young, G. Evermann, M. J. F. Gales, T. Hain, D. Kershaw, G. Moore, J. J. Odell, D. Ollason, D. Povey, V. Valtchev, and P. C. Woodland. *The HTK Book (for HTK Version 3.3)*. University of Cambridge, April 2005. 41, 46

[170] S. J. Young, G. Evermann, M. J. F. Gales, T. Hain, D. Kershaw, G. Moore, J. J. Odell, D. Ollason, D. Povey, V. Valtchev, and P. C. Woodland. HTK version 3.3. `http://htk.eng.cam.ac.uk/ftp/software/HTK-3.3.zip`, September 2005. Accessed June 4th, 2009. 46

[171] S. J. Young, J. J. Odell, and P. C. Woodland. Tree-based state tying for high accuracy acoustic modelling. In *HLT '94: Proceedings of the workshop on Human Language Technology*, pages 307–312, Morristown, NJ, USA, 1994. Association for Computational Linguistics. 220

[172] S. J. Young, N. Russell, and J. Thornton. Token passing: A simple conceptual model for connected speech recognition systems. Technical report, Cambridge University Engineering Department, 1989. 170, 225

[173] S. Zhai, P. O. Kristensson, and B. A. Smith. In search of effective text input interfaces for off the desktop computing. *Interacting with Computers*, 17(3):229–250, 2005. 109