2023

An Ambiguous Technique for Nonvisual Text Entry

Dylan C. Gaines
Michigan Technological University, dcgaines@mtu.edu

AN AMBIGUOUS TECHNIQUE FOR NONVISUAL TEXT ENTRY

By

Dylan C. Gaines

A DISSERTATION

Submitted in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

In Computer Science

MICHIGAN TECHNOLOGICAL UNIVERSITY

2023

This dissertation has been approved in partial fulfillment of the requirements for the Degree of DOCTOR OF PHILOSOPHY in Computer Science.

Department of Computer Science

Dissertation Advisor:	*Dr. Keith Vertanen*
Committee Member:	*Dr. Scott Kuhl*
Committee Member:	*Dr. Timothy Havens*
Committee Member:	*Dr. Ricardo Eiris*
Department Chair:	*Dr. Zhenlin Wang*

Contents

List of Figures

List of Tables

Author Contribution Statement

Chapter 2 of this dissertation is based on previously published work [23]. Parts of Chapters 3 and 4 have also previously been published [24]. Co-authors of [24] assisted in conducting the user study detailed in Chapter 4 as well as in reviewing the manuscript. Copyright on each of these published works are held by the author.

Acknowledgments

Thank you in particular to my committee members for their feedback throughout the project and to the National Federation for the Blind for their participation in these studies. Thank you to the Future Interactions Group and to Kenzie Baker for their contributions to this research.

This work was supported by NSF IIS-1909248 and by NSF Graduate Research Fellowship 2034833.

Abstract

Text entry is a common daily task for many people, but it can be a challenge for people with visual impairments when using virtual touchscreen keyboards that lack physical key boundaries. In this thesis, we investigate using a small number of gestures to select from groups of characters to remove most or all dependence on touch locations. We leverage a predictive language model to select the most likely characters from the selected groups once a user completes each word.

Using a preliminary interface with six groups of characters based on a Qwerty keyboard, we find that users are able to enter text with no visual feedback at 19.1 words per minute (WPM) with a 2.1% character error rate (CER) after five hours of practice. We explore ways to optimize the ambiguous groups to reduce the number of disambiguation errors. We develop a novel interface named FlexType with four character groups instead of six in order to remove all remaining location dependence and enable one-handed input. We compare optimized groups with and without constraining the group assignments to alphabetical order in a user study. We find that users enter text with no visual feedback at 12.0 WPM with a 2.0% CER using the constrained groups after four hours of practice. There was no significant difference from the unconstrained groups.

We improve FlexType based on user feedback and tune the recognition algorithm parameters based on the study data. We conduct an interview study with 12 blind users to assess the challenges they encounter while entering text and solicit feedback on FlexType, and we further incorporate this feedback into the interface. We evaluate the improved interface in a longitudinal study with 12 blind participants. On average, participants entered text at 8.2 words per minute using FlexType, 7.5 words per minute using a Qwerty keyboard with VoiceOver, and at 26.9 words per minute using Braille Screen Input.

Chapter 1

Introduction

In today's society, a large portion of mobile text entry takes place on touchscreen devices, whether it's a smartphone, tablet, or smartwatch. Users tap on a virtual keyboard that is shown on the screen and the exact location of each tap is recorded and used to produce text. However, there are times where precise touch locations are unavailable or impractical. If the user is blind or low-vision (BLV), they may not be able to see a virtual keyboard well enough (or at all) to provide precise input. This could impede their ability to enter text on touchscreen devices that lack the haptic feedback of key separations a physical keyboard has. Augmented and Virtual Reality keyboards share this problem, and it is exacerbated by mid-air keyboards lacking any tactile surface. While in this work we focus on accessible input for users who are BLV, sighted users may also wish to perform nonvisual text input in certain situations. If

users are multitasking or trying to be discreet, they may be entering text under a table or while their visual attention is directed at another task.

1.1 Existing Solutions

1.1.1 Nonvisual Text Entry Interfaces

The item selection technique used by Slide Rule [35] allows users to scan through a list of items by swiping their finger across the screen and select the item they are currently on by tapping a second finger in a different location. While the studies performed in this paper did not focus on text input specifically, the authors noted that their system did enable text input using a Qwerty keyboard. A similar technique can be used with an onscreen keyboard using Apple's VoiceOver[1] and Android's TalkBack[2]. Users can perform a double-tap gesture to enter a selected key or simply enter the selected key when they lift their finger, depending on their settings.

As an alternative to typing altogether, users can also dictate their text and have it be interpreted by a speech recognition algorithm. While speech recognition was first introduced on mobile devices by Fischer et al. [22], it was several more years before its

[1]https://support.apple.com/guide/iphone/use-the-onscreen-keyboard-iph3e2e3d1d/ios
[2]https://www.android.com/accessibility/vision/

performance as a nonvisual text input method was investigated. Azenkot and Lee [5] conducted the first research on speech input with users who were BLV, finding that they were able to use speech input to enter text at 19.5 words per minute. As we will see in Chapter 6, VoiceOver and dictation are the two primary ways that people with visual impairments enter text on their mobile devices.

Vertanen et al. [65] proposed a method where users would type an entire sentence, imagining a keyboard on a screen, without any feedback until the sentence was complete. The system would decode the entire sequence of taps into text at once so that it had all the data available. Users entered text at 23.3 words per minute while blindfolded, but with an 18.5% character error rate. Similarly, FlickType[3] allows users to enter text by tapping approximate character locations instead of definitively locating each key. Instead of entering a full sentence at once, FlickType users swipe to the right to signal to the system that they are finished entering a word. The system produces a best guess of the user's intended word from their tap locations, and the user can swipe through the list of suggested words.

Tinwala and MacKenzie developed a *Graffiti*-based approach [55] in which users would perform gestures similar to drawing each character on the screen. They found that users entered text with at an average of 10.0 words per minute with an error rate of 4.3%.

[3]https://www.flicktype.com/

No-Look Notes arranges characters into eight groups in a circle on the screen [8]. The groups are identical to the groups of letters found on a standard phone keypad and are explored in a manner similar to Apple's VoiceOver. The user explores the segments by dragging their finger on the screen until the audio feedback indicates the segment they want. Tapping a second finger on the screen selects the segment and brings up the characters in that segment to be explored and selected in a similar fashion. In a direct comparison, the authors found that participants entered text using No-Look Notes at 1.32 words per minute and only 0.66 words per minute using a keyboard with VoiceOver. Additionally, they found that users entered 0.11 incorrect characters per correct character in the target word using No-Look Notes, and 0.60 incorrect characters using VoiceOver.

1.1.2 Braille-Based Interfaces

Additional interfaces focus on using Braille-based gestures to map to characters. Perk-input [6] encoded characters as six-bit binary strings using the characters' Braille representation (found in Figure 1.1). These binary strings were entered using either three fingers on each hand or using two three-fingered touches with a single hand. The authors found that users were able to type an average of 17.56 words per minute with an uncorrected error rate of 0.14% using a single hand, or 38.0 words per minute with an error rate of 0.26% using the two-handed approach.

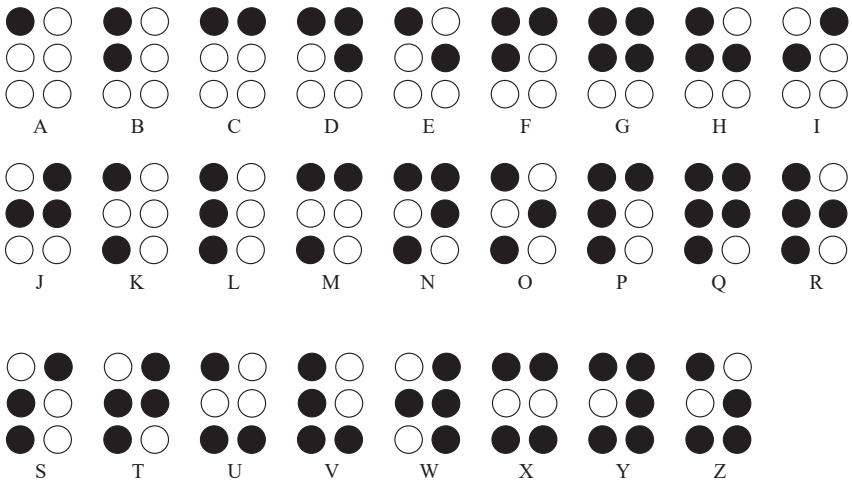


Figure 1.1: Braille representations for the English alphabet.

BrailleType places six targets on a touchscreen–one in each corner and one along each long edge–that correspond to the dot locations in a Braille character [47]. Users mark these dots by dragging their finger to each dot in the encoding, waiting for an audio confirmation at each location. Double tapping on the screen inputs the character represented by the currently marked dots, and swiping to the left clears any marked dots or deletes the last character if no dots are marked. In user studies, participants entered text at 1.45 words per minute using BrailleType compared to 2.11 words per minute using a technique identical to a Qwerty keyboard with VoiceOver. Error rates were on average 14.12% with VoiceOver and 8.91% with BrailleType.

Using BrailleTouch, a user holds the device with the screen facing away from them and uses the first three fingers on each hand to tap the Braille encoding for each character [53]. In user testing, the authors found that expert Braille typists obtained an average entry rate of 23.2 words per minute with a 14.5% error rate in their final

of five sessions using BrailleTouch on a smartphone. For comparison, they averaged 42.6 words per minute in the final session on a standard physical Braille keyboard with an error rate of 5.3%.

In contrast to Perkinput [6] and BrailleTouch [53] which split the 3×2 Braille matrix into left and right sides, TypeInBraille [43] allows users to enter characters one row at a time (i.e. using three actions instead of two). Users tap on the left or right side (or both) to indicate that that dot is raised, or with three fingers to indicate no dots in that row. Swiping to the right indicates the end of a character. In user studies, the authors found that participants were able to achieve an average of about 7 words per minute with just under 5% error rate using TypeInBraille.

In addition to these research solutions, commercial solutions have been developed with similar techniques. The MBraille keyboard[4] allows users to decide where to place each dot on the screen. Users enter characters by touching each dot in a character's encoding at the same time, and the character is typed when the fingers are released. Alternate modes allow for input with the screen facing away from the user, similar to BrailleTouch [53], and with all dots arranged horizontally, like bimanual entry in Perkinput [6].

The accessibility software developed by Apple and built into the iOS operating system

[4]https://mpaja.com/static/mbraille/help-en.htm

also allows for Braille-based text input with its Braille Screen Input (BSI)[5]. BSI also has a *screen away mode*, which operates similar to BrailleTouch [53], and allows the user to reposition the dots.

1.2 Drawbacks to Existing Methods

While these text entry methods all work, each has its own drawbacks. VoiceOver and No-Look Notes both require multiple user actions to enter a single character [8]. The *Graffiti*-based approach only required a single action, but that action was a gesture that the authors mentioned may not be properly recognized all the time [55]. This approach could also be more difficult for users to learn if they have been blind from a young age. These factors result in entry rates that are much lower than those reported for single-action Braille-based methods such as two-handed Perkinput [6] and BrailleTouch [53].

While the Braille-Based interfaces can work well for users familiar with Braille, they require users to learn the corresponding mapping for each individual character and to specify up to 6 dots to enter each character. Speech input is quick and doesn't require Braille, but Azenkot and Lee [5] found that users spent about 80% of their time editing the text, and that users had difficulty in noisy environments. This creates a need for

[5]https://support.apple.com/guide/iphone/type-onscreen-braille-iph10366cc30/ios

a non-Braille nonvisual text entry method that uses a short, single-action gesture, such as a tap, to input each character. A single tap would take a fraction of the time needed to enter a character using VoiceOver or No-Look Notes and would make nonvisual text entry much more efficient. Removing as much location dependency as possible removes the need for exploration and confirmation techniques such as that used by VoiceOver.

A non-Braille approach would also make this technique more accessible to a user with a situational impairment. A user that is walking down the street, for example, may wish to send a message without looking at their phone. A location-independent approach also has applications beyond a touchscreen. The main benefit of using a touchscreen is that precise touch coordinates can be captured. If we no longer need locations, we can map each group of characters to some gesture that can be detected another way (e.g. an instrumented glove or computer vision). This could have applications in augmented and virtual reality head-mounted displays, where there isn't a touchscreen to detect interaction coordinates.

1.3 Ambiguous Keyboards

We can meet these criteria by using an ambiguous keyboard with a small number of character groups. Ambiguous keyboards map multiple characters to a single input

gesture and use either a secondary gesture or a statistical process known as disambiguation to determine which letter out of a group the user intended to select. Disambiguation can be performed on every input gesture individually or on a full word or sentence. While there is theoretically no maximum length of text that can be disambiguated at once, the longer the text, the more possible combinations of characters there are to consider.

Before most commercial mobile phones had full sliding Qwerty keyboards or touchscreens, they had a nine-key (T9) ambiguous keyboard. An example of this can be seen in Figure 1.2. Users would repeatedly press a key to iterate through the characters assigned to that key until they reached their desired character, then they would proceed to the next character. Several studies through the years have studied this method, often reporting it as being slow, likely due to the need for multiple key presses for most characters [12, 32]. To remedy this, MacKenzie et al. [30] investigated a method that would search a dictionary for words that matched the sequence of key presses a user entered and return the most likely matching word. Using this method, users only needed to press each key once per letter.

More recently, this technique has been used to enter text on small form factor devices, such as smartwatches, that may lack the screen size for each character to have its own key. The Optimal-T9 keyboard [49] used 9 keys on a smartwatch keyboard. The keys were arranged to divide each row of a Qwerty keyboard into three keys, maintaining

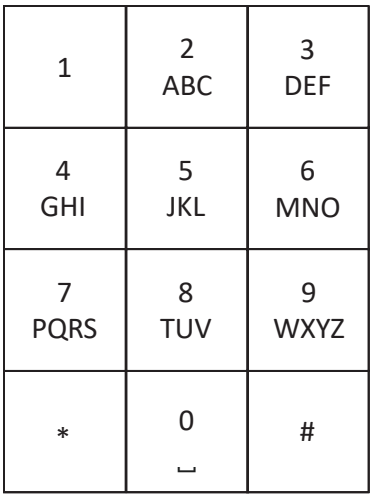


Figure 1.2: A T9 keyboard for keypad-based text input.

the order of letters. The authors found that this keyboard resulted in a 17% increase in entry rate from their users compared to a standard T9 keyboard.

The T18 keyboard [16] extended this to combine ambiguous and non-ambiguous keys into an interface with 18 keys (3 rows of 6 keys). The T18 keyboard still maintained the Qwerty arrangement of characters. Users typed with the T18 keyboard at 15.7 words per minute, compared to 11.3 words per minute with a keyboard based on the Optimal-T9.

Instead of having users tap directly on a touchscreen, TipText [66] allowed users to input text by using their thumb to type on an invisible ambiguous keyboard on the tip of their index finger. The input was then sent to a wearable device. The keyboard segmented a Qwerty configuration into a 2×3 grid of keys, with the top row containing letters from the top row of a Qwerty keyboard, and the bottom row containing letters

in the second and third Qwerty rows. Users wrote with TipText at about 13.3 words per minute.

In HiFinger [33], the authors instrumented a user's hand with pressure sensors to detect finger-to-thumb touch gestures for text input in virtual or augmented reality. The authors divided the alphabet and the ten numeric digits into six ambiguous groups of six characters each. Users selected their target character's group with their first gesture (out of six possibilities), and then performed a second gesture to select the specific character out of that group.

1.4 Project Overview

In this work, we explore creating and optimizing ambiguous keyboards for nonvisual text input. We begin with a prototype based on a Qwerty keyboard that has minimal dependence on the location of taps. We then use a long-span language model to optimize a fully location-independent ambiguous keyboard to reduce the expected disambiguation errors a user would encounter when entering text. We then compare two optimized keyboards in a longitudinal user study: one with freely optimized character groups to one with groups constrained to alphabetical order. We explore the possibility of presenting word predictions to users using simultaneous spatial audio, and we interview blind adults about their experiences with mobile text entry. Using

feedback from the user studies and the interview, we adjust our ambiguous keyboard interface and compare our text input method to users' typical nonvisual text input methods in a longitudinal user study. The resulting interface provides a non-Braille entry technique using a small group of quick input gestures. While in this work we investigate its performance on a touchscreen device, the location independence of this interface could enable its use on systems without a touchscreen.

Chapter 2

Evaluation of an Ambiguous Keyboard Prototype

In 2016, Vertanen proposed a text entry technique based on the number of fingers a user tapped on the screen at the same time [57]. Taps ranging from one to five fingers signified ambiguous groups of characters that could be based on a variety of mappings. After each word, the sequence of taps was fed to the VelociTap decoder [64] to determine the most likely intended word based on the taps and the context of what had already been typed. While some mappings were Braille-based, others were simply alphabetical and removed the need to know Braille encodings. In this chapter, we extend this work to create an input method using this technique that generates character groups from a Qwerty keyboard. We discuss the design of this interface

as well as its evaluation in a user study. The goal of this study was to determine if users are able to learn this new text entry technique and compare their performance to other nonvisual interfaces.

2.1 System Description

2.1.1 VelociTap Decoder

Our system needed a way to turn sequences of taps into text. Like Vertanen [57], we used the VelociTap decoder [64]. The VelociTap decoder uses n-gram language models to determine the probability of text sequences given the surrounding text [58]. An n-gram model considers sequences of up to n tokens to generate probability distributions based on the number of times those sequences occurred in a large quantity of training text. When performing character-at-a-time text entry, this means that the $n-1$ previous characters influence the distribution for the current character. For example, the 12-gram character model we use in this work would consider the previous 11 typed characters when evaluating each possibility for the 12^{th} character in the sequence.

The decoding process is slightly more complicated for an ambiguous keyboard like the one used here, since the prior characters in the sequence are not fixed and are being decoded at the same time. To perform this decode, the model considers all

of the possible sequences given the ambiguous groups selected and computes their probabilities given its training text.

The VelociTap decoder [64] we use in this chapter as well as future chapters was configured to combine a 12-gram character model with a 4-gram word model. Since the number of possible token sequences is quite high ($26^{12} = 9.5 \times 10^{16}$ for the character model, if only letters are considered), it is likely to run into a problem of sparsity, where exact sequences of tokens may not have been in the training text. To combat this sparsity, the model uses smoothing, where it reserves a small portion of the probability distribution for sequences that were not observed in the training text [58]. The word model functions on the same basic principles as the character model, but it examines sequences of words from its training text instead of sequences of characters.

Since our decoder is using two separate models, it needs to be able to combine the results that each model produces. The weight of each model is a configurable parameter within the decoder. The decoder's parameters also include a penalty for words not included in the decoder's vocabulary (in our work, we use a vocabulary of 100K words). This makes it possible, but less likely, for the decoder to return a word not in its vocabulary. These parameters are tuned based on past data from users typing. In Chapter 7 we re-tune the parameters using typing data from the study in Chapter 4 and investigate adding an input model to the decoder to model and correct for user

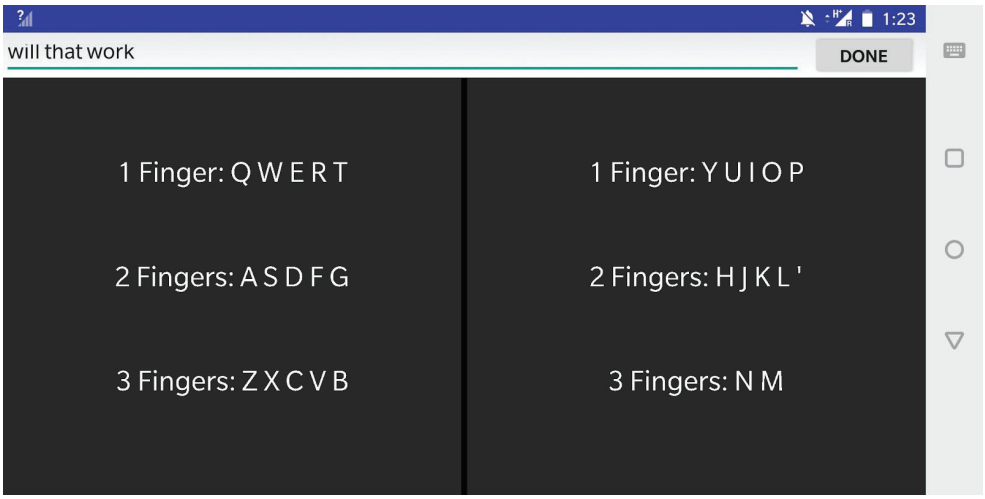


Figure 2.1: Tap123, the first ambiguous keyboard interface prototype used in this work.

input errors, but in this chapter we treat each input event as accurate.

2.1.2 Tap123 Interface

The Tap123 keyboard accepts taps containing between one and three fingers, inclusive. The number of fingers corresponds to the row of a Qwerty keyboard that the intended letter is in. In the original technique proposed by Vertanen [57], only the number of fingers was used to determine the group of characters. However, our interface uses the side of the screen on which the taps occur to further split each keyboard row into a left and right side. The exact groupings used can be seen on the interface depicted in Figure 2.1.

In addition to these multi-finger touch events to indicate groups of characters, a left

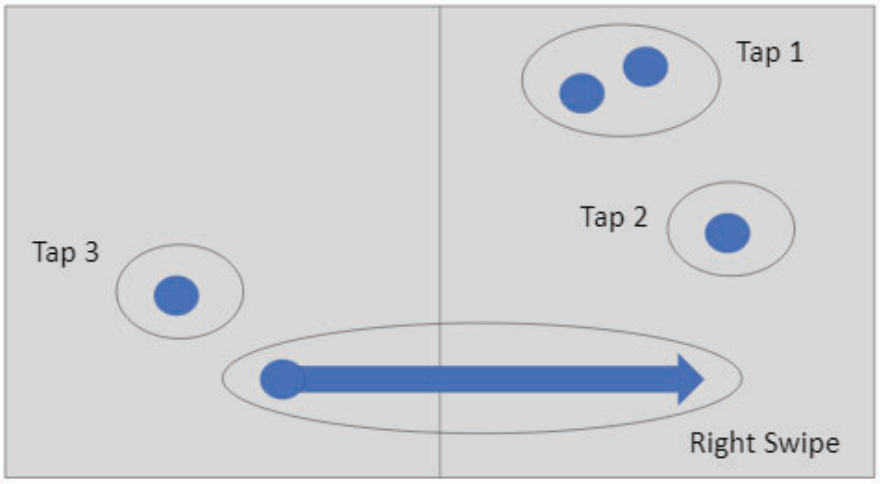


Figure 2.2: An example of a user's interaction sequence to enter the word "how" using Tap123. Each dot represents a finger touch location, and circled groups designate simultaneous touches.

swipe can be used to backspace a tap, while a right swipe sends the tap sequence to the VelociTap decoder [64] for disambiguation and inserts a space. If a left swipe is performed immediately after disambiguation, it will instead delete the entire word.

As in Vertanen's work [57], the VelociTap decoder is configured to treat each combination of finger count and side as a key with multiple possible letters. The decoder is configured to return a list of the 6 most likely words, which we will refer to as the n-best list, based on the current sequence of taps and the previously entered text (left context). If the first word returned is not the word the user intended, swiping up or down on the keyboard will iterate forwards or backwards through the n-best list, respectively.

An example event sequence of a word being entered is shown in Figure 2.2. The first

tap consists of two fingers on the right side of the screen, while the second tap is one finger on the right side. The third and final tap is one finger on the left side of the screen. The tap sequence is sent to the decoder when the interface receives the right swipe gesture. The most likely word recognized in this case is “how”, with the remainder of the n-best list containing “joe”, “hot”, “hit”, “low”, and “lot”.

In the absence of visual feedback, Tap123 instead provides audio feedback to convey information to the user. When any tap is performed, the keyboard speaks the letters in the group that corresponds to that tap. Upon recognition or an up or down swipe, the keyboard speaks the word that has been selected. If a word or tap is deleted, the keyboard will inform the user what they deleted. If the user taps and holds their finger on the screen for 600 ms or longer (long press), this will prompt the keyboard to read the contents of the text entry field. During the user study, a long press caused the application to reread the prompt text and then the contents of the entry field.

For the purposes of the user study, participants needed a way to signal that they were finished with a prompt and ready to proceed to the next. To close the keyboard interface when they were finished entering text, users were instructed to tap with six fingers at the same time (three on each hand since users are tapping bimanually). To make this a little easier to perform, the interface was configured to close if it detected five or more simultaneous fingers since this would not occur in normal text entry and we wanted the keyboard to close even if the device failed to recognize one of the

fingers. Tapping the screen after closing the keyboard allowed users to proceed to the next prompt.

2.2 Procedure

Our user study consisted of a series of sessions designed to introduce participants to the entry method slowly and allow them to learn the interactions. Though all four participants were sighted, the OnePlus 5T smartphone device on which entry occurred was obscured from the participant's view during all text entry tasks. In each session, we asked participants to transcribe text using Tap123. These prompts were read to the participants one at a time using Android's native text-to-speech software. Participants typed the prompt, then closed the keyboard and tapped to continue. Each participant completed 8 sessions consisting of about 40 minutes of text entry broken into about 10-minute segments (participants finished their current prompt before taking a break). All totaled, each session took about an hour. Since each session was time-based, the exact number of prompts completed varied, but across all participants there was an average of 86 prompts per session. Participants were asked to enter text bimanually with the device flat on the table.

We designed the first few sessions to build slowly and introduce the participant to the entry method. During the first session, the prompts we gave participants consisted

of single words ordered in small sets designed to obtain complete coverage of all available characters. The words were drawn from a list of the 2000 most frequently used words in English. The second session progressed to simple phrases with at most four words and a maximum word length of six characters. For the second session, we also removed any prompts that would require use of the n-best list. In the third session, participants still entered simple phrases, but they were introduced to the n-best list feature. For example, in the context of the session 3 sample prompt shown in Table 2.1, the tap sequence for "fine" initially recognizes the word "done". Prompts that participants entered in all subsequent sessions were only restricted to be six words or less to increase their memorability. All prompts in sessions 2 and later were drawn from the Enron mobile data set [61], a corpus of 623 phrases entered on mobile devices by Enron employees. Participants did not receive the same prompt more than once in a single session, but it was possible to receive a duplicate prompt in a subsequent session.

2.3 Results

We recruited four participants through convenience sampling. Three identified as female, and one as male. Three participants were 20 years old, and one was 24. Three participants were studying at a university, and all four rated the statement "I

Session	Prompt Description	Sample Prompt Text
1	single words, no n-best needed	leaves
2	simple phrases, no n-best needed	are you there
3	simple phrases, n-best allowed	should be fine
4	phrases, n-best allowed	he says he has some ideas
5	phrases, n-best allowed	too tiny
6	phrases, n-best allowed	so you're ignoring me
7	phrases, n-best allowed	i left a message
8	phrases, n-best allowed	this is the crew

Table 2.1

Sample prompts given to users in each session. Simple phrases had a maximum of four words, each with a maximum of six characters. All other phrases had a maximum of six words and no limit on word length.

am a fluent speaker of English" as 7 on a 7-point Likert scale. The same was true for the statement "I frequently enter text on a mobile device".

Since the sessions the participants completed were designed to train them to use the interface with increasing complexity, we wanted to evaluate performance after participants had learned the technique. We treated sessions 1 through 5 as practice and used sessions 6 through 8 for evaluation, computing our metrics as averages across those sessions. We excluded from all of our analysis a total of 25 prompts that participants expressed difficulty hearing or spelling during entry. We did this to more accurately represent the performance of the interface, since in these cases the issue was rooted in the transcription task itself and not in the entry method. In a real-world application the user would know what they are trying to type. Excluded prompts frequently included proper nouns, such as in "did you mean oxley". We evaluated the interface using three primary metrics.

To evaluate participants' performance with Tap123 we used the following metrics:

- **Entry Rate** — We measure entry rate in words per minute (WPM). We calculate this by dividing the text length by 5 (to standardize word length at five characters, including a space) and then dividing by the time it took participants to finish the prompt. Time for each prompt was measured from the start of the first tap until participants closed the keyboard, indicating that they were finished with that prompt.

- **Error Rate** — We measure character error rate (CER) as the minimum number of insertions, deletions, and substitutions required to obtain the reference text, divided by the number of characters in the reference text, multiplied by 100%.

- **Backspaces per Tap** — We measured backspaces per tap by dividing the total number of taps deleted by the total number of taps entered. This allows us to measure the accuracy of participants' initial taps instead of the accuracy of the final entered text.

As shown in Figure 2.3, participants averaged 4.3 WPM in session 1, and achieved an average 19.1 WPM across the final 3 sessions. While Participant 4 seemed to plateau in the final 3 sessions after a large increase in entry rate between sessions 5 and 6, the other three participants continued to increase their entry rate from session 6 to session 8.

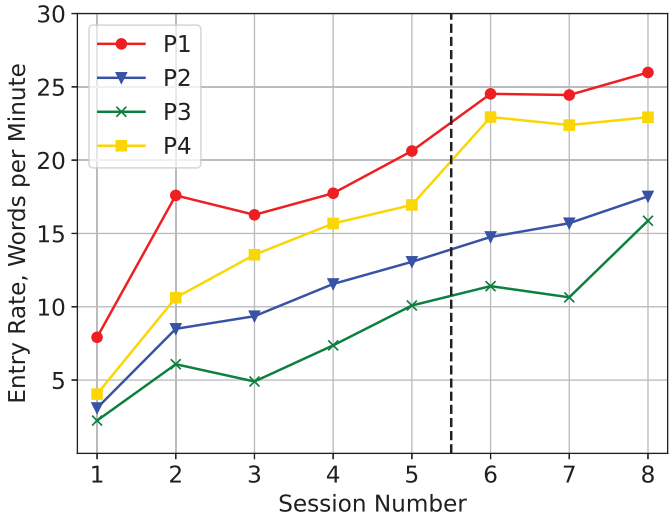


Figure 2.3: Entry rate by session using Tap123. Each line represents an individual participant. The dashed vertical line shows the separation between training and evaluation sessions.

Participants had an average 4.1% CER in session 1, which improved to 2.1% CER averaged across the final 3 sessions. The graph in Figure 2.4 shows that most participants obtained much better accuracy after initially struggling in session 1. While Participant 2 saw an initial drop in error rate from session 1 to session 2, their error rate increased again over the next few sessions. P2 averaged 4.9% CER over the final 3 sessions, pulling up the overall mean as the other three participants averaged only 1.2% CER.

Participants averaged 0.199 backspaces per tap in session 1, but as shown in Figure 2.5, remained relatively stable around the average of 0.068 backspaces per tap for the remainder of the sessions. Participant 2 did backspace slightly more in their final two sessions than in sessions 2 through 6, which perhaps helped lower their error rate

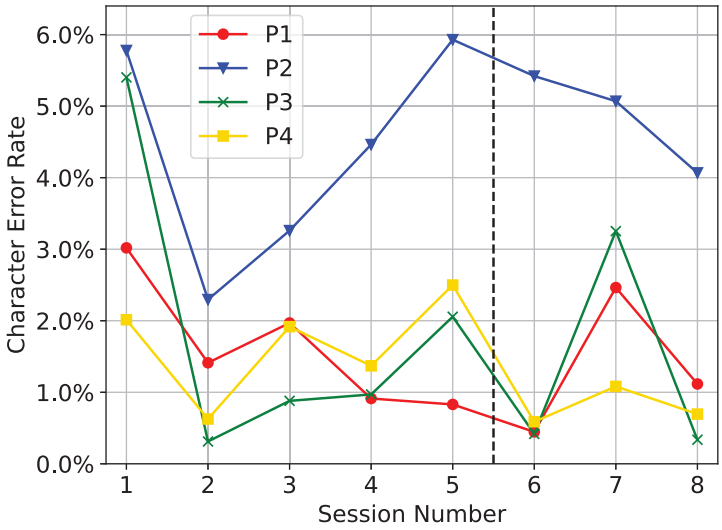


Figure 2.4: Character error rate by session using Tap123. Each line represents an individual participant. The dashed vertical line shows the separation between training and evaluation sessions.

from its peak in session 5.

Investigating this corrective behavior more, we found that in 28% of instances where participants backspaced, they backspaced multiple times in a row. Each of these multi-character backspaces averaged 4.4 characters in a row, demonstrating a need for a way to delete more than one character at a time.

2.4 Key Takeaways

To develop our next interface in future chapters, we make some changes from Tap123 in response to participant comments and our observations in this study. When asked

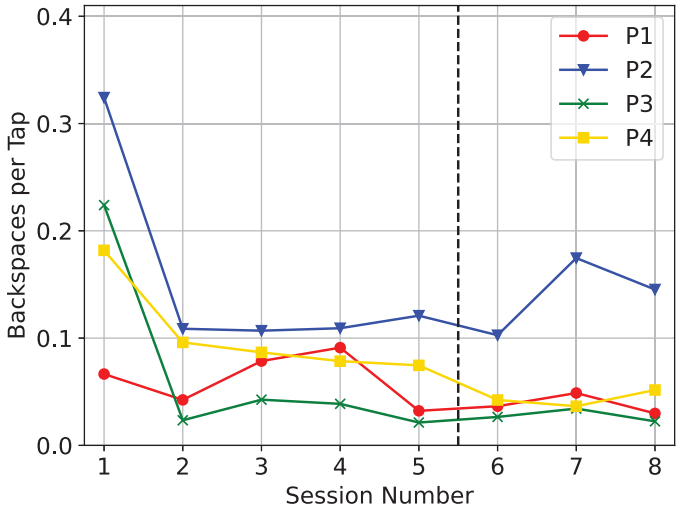


Figure 2.5: Backspaces per tap by session using Tap123. Each line represents an individual participant. The dashed vertical line shows the separation between training and evaluation sessions.

about unnatural interactions after the conclusion of their final session, Participant 3 noted that "Closing the keyboard was always an issue for me," so we improve the gesture to close the keyboard to make it easier and more consistent in our interface in Chapter 4. Because of our observations on multi-character backspaces, we also give users the ability to delete an entire word at once at any point during entry. In response to the same question about unnatural interactions, Participant 4 said, "Finding the correct word because my suggested words are above the keyboard on my phone." In Chapter 5, we explore ways to provide suggested words to users with simultaneous audio feedback.

After session 3, Participant 1 remarked that, "There is a bit of a mental delay when you're paying attention to the recognized word that prevents one from typing as

quickly as possible." Along similar lines, Participant 4 inquired about a way to go back and scroll through recognition alternatives after having moved on to the next word. These issues were caused by several common words that had identical tap sequences, causing the disambiguation algorithm to not always able to determine the correct one. This lead to users needing to slow down and verify each word was correct before moving on, or go back and delete the word if they began the next word before realizing the recognition was incorrect. In the next chapter, we explore optimizing the character groups to reduce the frequency of these collisions instead of simply using groups based on the Qwerty layout.

Chapter 3

Keyboard Optimization

In this chapter we will explore different ways to optimize an ambiguous keyboard for use in a nonvisual text entry interface. The primary goals of this line of work are to 1) determine if it is feasible to reduce the number of ambiguous groups from six to four, and 2) examine the disambiguation performance decrease from constraining the groupings alphabetically. Reducing the number of groups would allow users to enter text using a single hand, as opposed to Tap123's two-handed technique. This is more suitable for truly mobile text entry as users can then hold the device with their other hand. A four-group approach would also enable us to remove location dependency completely, with the user simply tapping with between one and four fingers, inclusive, to indicate the groups. This could also open the door to detection methods other than a touchscreen. Past work shows that constraining the groupings alphabetically makes

them easier to use without extensive training. However, it has the potential to reduce the benefit of the optimization, since the search space is narrower. We will combine ideas from past work on this topic as well as introduce novel ideas to create multiple proposed groupings that are likely to create a more efficient and accurate interface.

3.1 Related Work

3.1.1 N-Opt Algorithm

There has been a considerable amount of past research done on the topic of optimizing keyboard layouts for a variety of metrics such as finger travel distance, keystroke efficiency, and tap clarity (how well an interface can determine the intended key from a user's interaction). Several papers introduce ideas that are directly relevant to this particular application. The first of these papers, written by Lesher et al. [40], shows that fully enumerating and evaluating every possible set of groupings is computationally infeasible, but proposes an algorithm that can be used to efficiently find locally optimized groupings in ambiguous keyboards. The first step in their algorithm is to compute a confusability matrix for some corpus of English text. They do this by stepping through the corpus one character at a time and creating a list of the most likely characters to come next based on a character prediction algorithm (e.g. 'U'

would be quite likely after a 'Q'). They keep track of how frequently an incorrect character is predicted as more likely than the actual next character in the text. The authors define the mutual confusability of two characters α and β as

$$M(\alpha, \beta) = C(\alpha, \beta) + C(\beta, \alpha), \tag{3.1}$$

where $C(\alpha, \beta)$ is the number of times α was mistaken for β, and $C(\beta, \alpha)$ is the reverse. Further, for any number of characters placed in a single ambiguous group, the total mutual confusability for the group is the sum of the mutual confusabilities between each pair of characters in that group.

Once the confusability matrix is computed, Lesher et al. [40] run their n-opt algorithm. The algorithm starts with a valid arrangement of characters into groups. On each pass, it checks every possible tuple of n characters to see if any way of interchanging those characters' groups will result in a better overall arrangement according to whatever metric is being optimized (in this case, the confusability of the groups). For example, a 2-opt pass would check every pair of characters ('AB', 'AC', ..., 'YZ') and a three-opt pass would check every triple ('ABC', 'ABD', ..., 'XYZ') in alphabetical order. If any swaps are made during the course of a single pass, the pass repeats once it has finished, checking all tuples again. The algorithm continues until a pass completes with no swaps being made. The n-opt algorithm requires factorially increasing

computations for higher values of n; the highest pass completed by Lesher et al. was 5-opt [40]. Since the n-opt algorithm only finds local optima, the authors start with many initial arrangements and run the 2-opt algorithm. They take the result with the best performance and further improve it via the 5-opt algorithm.

An alternative approach to keyboard optimization is the use of genetic algorithms. Gong and Tarasewich [25] optimized an unconstrained ambiguous keyboard to minimize the number of keystrokes required on average to enter a character of input. They first generated a random population of keyboards from which to start. Each successive generation of keyboards was the product of reproduction, crossover, and mutations from the prior generation. Details on how these operations were performed were not provided. The authors produced keyboards with between one and twelve keys, and tested their disambiguation accuracy on three different corpora: one of written text, one of spoken text, and one of SMS (Short Message Service) text. They found that their keyboards had a 95% or greater disambiguation accuracy with six or more keys on the written and spoken corpora, and with ten or more keys on the SMS corpus.

Nivasch and Azaria [46] also used a genetic algorithm to optimize non-ambiguous keyboards based on the effort it would take a user to type with it. Since the Carpalx keyboard effort model [38] they used is expensive to compute, Nivasch and Azaria

trained a neural network model to compute a heuristic score for each successive generation of non-ambiguous keyboards. They found that their genetic algorithm was able to produce keyboards with similar effort scores to those produced by the Carpalx project [38] through stochastic simulation.

3.1.2 Constrained Keyboards

Gong and Tarasewich [25] also optimized an ambiguous keyboard that was constrained to be in strict alphabetical order and compared it to their freely optimized keyboard. Adding the alphabetic constraint allowed them to fully enumerate the possible constrained keyboards. They tested the genetic algorithm described in Section 3.1.1 on the constrained keyboards, and found that it produced results that were always within 0.33% of the optimal solutions found by enumeration. Additionally, the authors compared their eight-key optimized constrained and unconstrained keyboards in a user study. They found that novice users entered text with the constrained keyboard significantly faster, with no significant difference in error rate from the unconstrained keyboard. From this, they concluded that the constraint aided users' ability to learn the interface since it was more familiar. In a second session of their user study one week after the first, the authors found that participants entered text significantly faster using the constrained keyboard than in the first session. Participants did not use the unconstrained keyboard again in the second session, so the authors were

unable to analyze the learning effect of the unconstrained design.

Other authors found similar results when comparing freely optimized keyboards to ones with familiarity constraints. For example, Bi et al. [7] performed a study with a Qwerty keyboard, a Quasi-Qwerty keyboard, and a freely optimized keyboard. The Quasi-Qwerty and freely optimized keyboards were designed to minimize the travel distance for a user's finger in order to try to increase the entry rate. However, the Quasi-Qwerty keyboard had the constraint that letters could not move more than one row and one column from their initial Qwerty position. The authors found that while the Quasi-Qwerty keyboard had an improved movement efficiency from the standard Qwerty keyboard, it was not as efficient as the freely optimized keyboard. However, during user trials, the authors found that users took the longest to locate the initial letter of a word on the freely optimized keyboard, followed by the Quasi-Qwerty, and finally the standard Qwerty. The authors concluded that the Quasi-Qwerty keyboard was effective at obtaining an increased movement efficiency while not sacrificing too much time in the initial visual search. While through practice users would improve with the freely optimized keyboard, using more familiar keyboards reduces the amount of practice needed.

3.1.3 Multi-Parameter Optimization

Instead of simply enforcing a strict Qwerty restriction on their non-ambiguous keyboard, Dunlop and Levine [19] chose to optimize their keyboard on three different metrics:

- **Finger Travel Distance** — The distance between each pair of letters, multiplied by the number of times those letters were adjacent in the language corpus, summed over each pair of letters in a given keyboard. While this metric is relevant to many on-screen optimization problems and is common in related work, it does not apply to an ambiguous keyboard such as ours that is location-independent.

- **Tap Ambiguity** — The number of spatially adjacent characters in a keyboard that could frequently be swapped with each other to create valid words (e.g. I and O can be swapped between the words 'for' and 'fir'). We describe this in detail here since we build upon this parameter in this work. They first created a table of these commonly interchanged letters, which they referred to as bad bigrams, or "badgrams". After counting the frequency of badgrams in same-length words in a text corpus, they converted each frequency to a probability by dividing by the total number of badgram occurrences. The authors defined

their tap clarity metric for a given keyboard as

$$M_{tap\_clarity} = 1 - \sum_{\forall(i,j)\in\alpha} \begin{cases} p_{ij} & \text{if } i \text{ and } j \text{ are adjacent on the keyboard} \\ 0 & \text{otherwise} \end{cases} \tag{3.2}$$

where α is the set of all unique unordered letter pairs in the symbol set and p_{ij} is the badgram probability for letters i and j.

- **Familiarity** —The similarity of a given keyboard to the Qwerty keyboard, calculated by summing the squared distances between each key's position and its Qwerty position and then normalizing the results to the range between 0 and 1. This allowed for potentially high-scoring keyboards that had most of their letters near their Qwerty positions.

With these three metrics, Dunlop and Levine [19] used Pareto front optimization to find candidate keyboards. Pareto front optimization keeps track of the set of *Pareto optimal* keyboards, which are those that for all other keyboards in the set, there is no keyboard that is better on all three metrics. They defined a small set of randomly generated initial keyboards that were then taken through 2000 iterations of changes. At each iteration, one key was swapped and there was a 25% chance of swapping an additional key after each swap. The metrics were then recalculated and the keyboard was either added to the Pareto front or discarded. After the final Pareto front was formed, the authors selected the keyboard nearest the 45° line (the line with all metrics

equal), which they determined was the best overall keyboard given the metrics. All metrics were normalized to range from 0 to 1.

Qin et al. [49] also used a Pareto front to perform optimization, but they did so in two dimensions and with an ambiguous keyboard. They defined their clarity metric for a given word as the frequency of that word in the corpus divided by the total frequency of identical tapping sequences given a set of ambiguous groups. They then defined the clarity of a grouping as the sum over all words of the product of word frequency and word clarity. The second metric used in this work was a typing speed metric based on the relative location of frequent character combinations. As with Dunlop and Levine [19], this is not relevant to interfaces that remove location dependency. Instead of choosing the keyboard closest to the 45° line in the Pareto front as done by Dunlop and Levine [19], Qin et al. opted to select the keyboard that had the highest average of normalized metrics. Another difference between these works was that instead of optimizing on a third metric that indicated similarity to Qwerty, Qin et al. enforced a strict adherence to the Qwerty ordering of characters and split each row into three ambiguous groups, creating a total of nine groups.

Lee et al. [39] optimized five- and ten-key ambiguous keyboards for speed, accuracy, comfort, and confusability using Pareto front optimization. Their text entry method used fingernail-mounted sensors to track intra-hand touches and mapped each finger

to a subset of characters. The five-key keyboards utilized a single hand, while the ten-key keyboards used both hands. Their confusability metric was based on confusability matrices derived from a text corpus, similar to Lesher et al. [40].

While past work has optimized ambiguous keyboard groups in various ways, none factored in the use of a long-span language model (i.e., a language model that conditions its prediction on several previous words). The optimization performed by Lesher et al. [40] only predicted the next character to be entered, and did not consider the likelihood of each word as a whole. The word-level clarity metric used by Qin et al. [49] used only the frequency of each word in a text corpus, and did not take into consideration the context in which each word was used. With our optimization procedure, we aim to leverage long-span language modeling to determine likely disambiguation errors given the frequencies of words and contexts in which they are commonly used in a corpus of representative text. We then create optimized character groups by separating characters that are commonly confused in that corpus. We hypothesize that these optimized groups will reduce the number of disambiguation errors a user experiences on average, thereby making their text entry more efficient.

3.2 Procedure

To explore the feasibility of reducing the number of groups from the six-group (T6) configuration of Tap123 in Chapter 2, we optimized both four-group (T4) and six-group (T6) sets. This will also enable us to directly compare our optimized groups to the Qwerty-based groups used in Chapter 2. While the T4 sets only having four groups of characters may make disambiguation more difficult, they allow one-handed input to be performed more easily. On a touchscreen, this enables users to tap with between one and four fingers on a single hand to designate each group. In virtual or augmented reality, this would allow users to make a thumb-to-finger gesture as done by Bowman et al. [9] and by Jiang et al. [33]. Having four groups of characters enables us to assign one group to each of the four fingers that the thumb could touch on a single hand. This allows the user to utilize proprioception to execute the four distinct input actions with no visual feedback. This motion could be detected using wearable gloves [9], finger-mounted pressure sensors [33], or surface electromyography via electrodes on the forearm [4].

We developed our optimization procedure by combining some of the ideas used for optimization in past work [19, 40] and factoring in a long-span language model. Since the use case we investigate here is eyes-free text input on mobile devices, we performed our analysis on a corpus of text written on mobile devices. This particular corpus

was gathered and released by Vertanen and Kristensson [63] and consists of forum posts made by users on a mobile device.

3.2.1 Performance Metrics

To measure the number of disambiguation errors a user might expect to encounter during text entry with a given set of groups, we used *word error rate (WER) clarity*. To calculate WER clarity, we utilized the test set of text typed on mobile phones from the corpus released by Vertanen and Kristensson [63]. We first defined the groups that we were evaluating as an ambiguous keyboard in the VelociTap decoder [64], specifying which characters were in each group. We then iterated through each word in the first 250 phrases in the test set. We simulated the taps that would be needed to type each word given the current set of groups. We then used the decoder with a 4-gram word model and a 12-gram character model to find the probability of each word that fit the ambiguous group sequence given the words in the phrase to the left of the current word. We looked at the most probable word returned by the decoder, and determined if it was the true word. We aggregated all of the predictions to define WER clarity as:

$$Clarity_{WER} = 1 - \frac{\sum_{\forall word} correct(word)}{n_{words}}, \tag{3.3}$$

where $correct(word)$ returns 1 if the decoder's prediction matches the true word and 0 otherwise, and where n_{words} is the total number of words in the first 250 phrases of the mobile test set.

Because calculating WER clarity requires us to iterate through our test phrase set for each set of groups, it is difficult to test a large variety of groupings with this metric. To counter this, we devised a second metric, *badgram clarity*, by combining the ideas of Dunlop [19] and Lesher [40] with our long-span language model.

First, we performed an analysis on the corpus of text written on mobile devices gathered and released by Vertanen and Kristensson [63]. To perform this analysis, we adapted the ideas of Lesher et al. [40]. While they iterated through the corpus one character at a time, predicting the most likely next character, we iterated through the first 100,000 phrases in the training set one word at a time. We chose to use the training set in order to have a larger body of training data. We used software based on the VelociTap decoder [64] (with the same models used for WER clarity) to predict the most likely word given the preceding words in the phrase.

We queried the decoder for the top 100 predictions for each word, given the context of the previous words in the phrase, and stored each word deemed more likely than the true word. We restricted the decoder's search to words that matched the length of the actual word in the phrase since the disambiguation process would have that same restriction. We used these mispredicted words to build a character-level confusion

matrix for the corresponding letters between the incorrect and correct words. For example, if 'hello' was predicted before true word 'world', the matrix entries for the character pairs 'hw', 'eo', 'lr', and 'od' would be incremented.

Using this confusion matrix, similar to Lesher et al. [40], we calculated the mutual confusability between each pair of characters. We divided each of these by the total sum, as done by Dunlop and Levine [19], to obtain badgram probabilities p_{ij}:

$$p_{ij} = \frac{C(i,j) + C(j,i)}{\sum_{\forall i,j|i \neq j} C(i,j) + C(j,i)} \tag{3.4}$$

where $C(i,j)$ is the number of times i was mistaken for j in the corpus analysis (element i,j in the confusability matrix). We adapted Dunlop and Levine's tap clarity formula to define our badgram clarity metric as

$$Clarity_{badgram} = 1 - \sum_{\forall i,j|i \neq j} \begin{cases} p_{ij} & \text{if } i \text{ and } j \text{ are in the same group} \\ 0 & \text{otherwise} \end{cases} \tag{3.5}$$

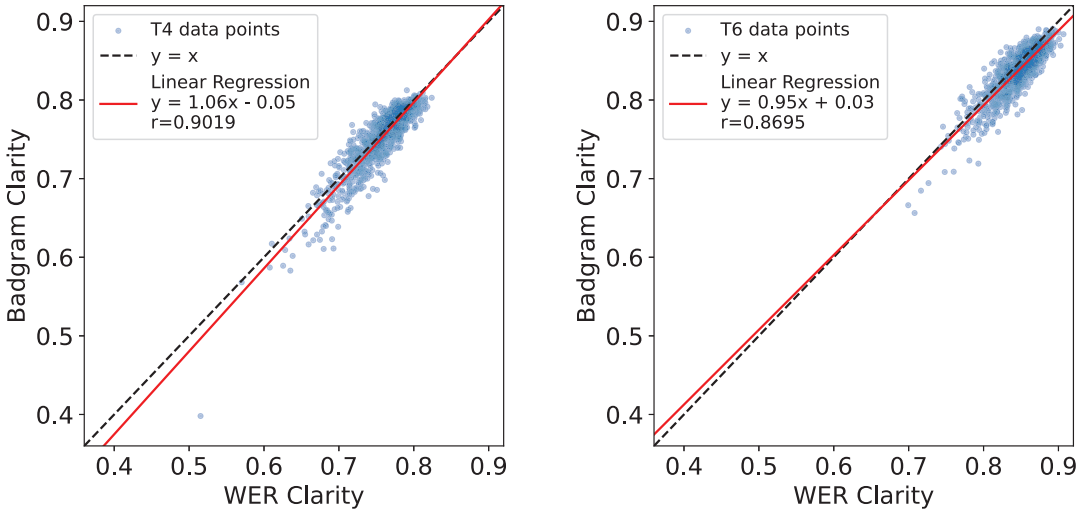
Badgram clarity is a simpler metric that may not as accurately model the disambiguation errors a user will face when using a set of groups in practice (i.e., when disambiguation is guided by a language model that conditions on a user's previously written text). However, since the confusion matrix is precomputed, it is much more

efficient to calculate than WER clarity.

To determine how well we can estimate performance using badgram clarity, we tested the similarity of the two metrics. We randomly generated 1,000 T4 and 1,000 T6 groupings. We calculated both clarity metrics on each grouping and fit a linear regression model to the data. The results can be seen in Figure 3.1. Both the T4 groupings and the T6 groupings produced results that were near the line $Clarity_{badgram} = Clarity_{WER}$ with relatively high correlation ($r = 0.9019$ and $r = 0.8695$ in the T4 and T6 groupings, respectively). From this, we can conclude that badgram clarity is a good, and computationally cheaper, estimate of a grouping's expected performance in a text entry system leveraging long-span language models for disambiguation.

3.2.2 Unconstrained Optimization

We began by optimizing groupings freely, without any familiarity constraints. The goal of this line of optimization was to determine the upper bound of performance that can be achieved, even if it takes additional training time for users (assuming we find the optimal groups, which is not necessarily guaranteed). Since, as Lesher et al. [40] showed, an exhaustive search is computationally infeasible, we used an n-opt approach based on their work. In their work, when they examined the possible swaps between



(a) T4 Groupings

(b) T6 Groupings

Figure 3.1: Scatter plots of different metrics on 1000 random groupings. The dashed line represents $Clarity_{badgram} = Clarity_{WER}$. The solid red line represents the best fit linear regression.

groups of characters, they only mentioned directly swapping the characters. However, since it could be the case that not all groups have an equal number of characters, we added some additional comparisons. For example, let Group A contain α (among other characters) and let Group B contain β (among other characters). For a 2-opt pass, we checked the grouping where we simply swapped α and β as Lesher et al. did. However, we also checked the groupings where both α and β were in the same group, either both in Group A or both in Group B.

To implement this, we first generated the set of all n-tuples in our set of characters (where n is the n in n-opt). For each tuple, we then generated all possible permutations. For example, for tuple (ab), we generated (ab, ba). For each permutation, we

examined all possible partitions into k groups (allowing empty groups), where k was the number of groups that contained any character in the permutation for the current groupings (e.g. if a, b, and c were each in a separate group, $k = 3$). An example of all partitions generated on the permutation (ab) is $(|ab,\ a|b,\ ab|)$. Note that there are $k - 1$ dividers, denoted in the example by $|$, to create k groups.

To increase efficiency, we tracked each partition created for each given tuple, and did not add duplicates to the final list of groupings. If we generated partitions on the permutation (ba) after already doing (ab), we would generate $(|ba,\ b|a,\ ba|)$. The first and last partitions are identical to the first and last partitions already generated from the permutation (ab), since ordering within each group does not matter (i.e. $(|ab)$ is equivalent to $(|ba)$). In this situation, only the middle partition $(b|a)$ would be added to our final list, since the others are already included.

3.2.2.1 WER Clarity

As we discussed in Section 3.2.1, the WER clarity metric is computationally expensive and requires iteration through 250 phrases from the mobile test set [63] to check each grouping. This, combined with the factorial growth of the n-opt algorithm, prevented us from running a large number of trials and from running anything larger than 2-opt. Using WER clarity, we ran 50 random initial groupings through our 2-opt algorithm. We did this separately for T4 and T6 groupings.

3.2.2.2 Badgram Clarity

We then started from scratch and optimized using badgram clarity, calculated as described in Section 3.2.1. Similar to Lesher et al. [40], we ran our version of the 2-opt algorithm on 50,000 random initial groupings. We then ran our version of the 5-opt algorithm on the best result from 2-opt. As with our WER-optimized groupings, we did this separately for T4 and T6 groups.

3.2.3 Constrained Optimization

In Chapter 2, we used groupings based on the Qwerty keyboard. However, as we move towards a more flexible input technique that may not depend on location, Qwerty-based groupings will likely have less of a positive impact on performance. Instead, in this section, we will focus on alphabet-based groupings. While Gong and Tarasewich [25] implemented a strict alphabetical constraint, we investigated allowing a small number of characters to shift outside of alphabetical order.

Constraining the groupings in this way drastically reduces the number of possibilities, which allowed us to fully enumerate and compare all of them. We first laid out the alphabet from A-Z with apostrophe on the end. We then used a similar partition function that we used on our unconstrained groupings, but this time we did not

allow any groups to be empty, as moving a letter from a group where it is alone will never increase performance. For a T4 keyboard, there are 2,600 possible partitions (3 dividers over 26 positions = $\binom{26}{3}$), and for a T6 keyboard there are 65,780 ($\binom{26}{5}$). For each partition, we selected every combination of s characters, where s was the maximum number of characters allowed to violate alphabetical order. We tested values of s from 0 to 4, inclusive, for T4 groupings, and s from 0 to 3, inclusive, for T6 groupings. We were unable to test $s = 4$ for T6 groupings due to the computational complexity. We then compared every possible way to shuffle the s characters in each combination. While our shuffles in the n-opt algorithm only allowed characters to move to groups occupied by another member of the combination, here characters could move to any group. The number of possible combinations was $\binom{27}{s}$, which for $s = 2$ was 351, for $s = 3$ was 2,925, and for $s = 4$ was 17,550. The number of possible shuffles for each combination was equal to t^s, where t was the number of groups in the set (4 or 6 for our work here). The total number of possible groupings was then equal to:

$$Groupings = \binom{26}{t-1} \times \binom{27}{s} \times t^s. \tag{3.6}$$

For the T4 groups, this equated to 2,600 possible groupings for $s = 0$, and 1.17×10^{10} for $s = 4$. For the T6 groups, this equated to 65,780 possible groupings for $s = 0$ and 4.16×10^{10} for $s = 3$. If we were to have run $s = 4$ for the T6 groups, there would

have been 1.50×10^{12} possible groupings. Due to this factorial growth of the number of possible combinations, we were only able to complete this optimization using the badgram clarity metric.

3.3 Results

3.3.1 Unconstrained

We first explored the groupings optimized using WER clarity and then proceeded to those optimized using badgram clarity.

3.3.1.1 WER Clarity

Among T4 groupings, we were able to achieve a peak WER clarity of 0.8542 with the following groups:

$$(a, d, f, h, k, q, y, '), (b, c, e, i, j, n, x), (g, l, o, s, v, w), (m, p, r, t, u, z)$$

For the T6 groupings, the best WER clarity was 0.9346, which was achieved by this set of groups:

$$(a, k, m, w, y), (b, l, q, s, v, z, '), (c, e, f, j, x), (d, i, n, p), (g, o, r), (h, t, u)$$

One observation that we made on these groupings was that wherever possible, vowels were placed into different groups. In the T6 grouping, each of the five main vowels occupied a separate group, although y shared a group with a. In the T4 grouping, there were not enough groups to separate all of the vowels, so e and u shared a group in addition to i and y.

While when optimizing with WER clarity we had hoped to see multiple random initial groupings converge to the same optimum, this was not the case. For both the T4 and T6 groupings we optimized, the best clarity was only achieved by a single grouping, likely because our sample size was not large enough. While this constraint was due to the computational complexity of the WER clarity metric, it leaves us less confident that were were able to find a global optimum.

3.3.1.2 Badgram Clarity

We next optimized a large number of initial groupings using badgram clarity to try to beat our best WER-optimized groupings. For the T4 groupings, the best achieved badgram clarity was 0.8191. There were 208 of the 50,000 initial groupings that when run through our 2-opt algorithm produced final groupings that matched this exact clarity. While the order of the groups and the order of the characters within each

group varied between these, alphabetizing each group and set of groups showed that these groupings were all identical. This best grouping was:

$$(a, h, k, n, s, x), (b, e, f, g, j, m, q, u), (c, d, o, t, v, z, '), (i, l, p, r, w, y)$$

When we ran this grouping through the 5-opt algorithm, there were no swaps made in the entire pass.

We found similar results in the 2-opt optimization of the T6 groupings. There were 3,509 of the 50,000 initial groupings that converged to a single grouping with a bad-gram clarity of 0.9067.

$$(a, d, l, q), (b, e, j, m), (c, g, k, o, w, z), (f, i, s, x, '), (h, n, p, v, y), (r, t, u)$$

Once again, when we ran this grouping through the 5-opt algorithm, no swaps were made. As we expected, the peak clarity of the T6 grouping was much higher than that of the T4 grouping.

We can note the same observation for these sets of groups that we did for the WER-optimized groupings: the vowels are separated wherever possible. We computed both clarity scores for these four groupings, which are shown in Table 3.1. While

Groups	Opt. metric	Badgram clarity	WER clarity
4	Badgram	0.8191	0.8209
4	WER	0.7990	0.8542
6	Badgram	0.9067	0.9106
6	WER	0.8955	0.9346

Table 3.1
Both clarity scores for the top unconstrained groupings. The groupings were optimized with respect to two different metrics and both results are reported.

our badgram-optimized groups had higher badgram clarity scores than our WER-optimized groups, they were unable to exceed their WER clarity scores. For comparison, the T6 Qwerty grouping used in Tap123 has a badgram clarity score of 0.8427 and a WER clarity score of 0.8418.

3.3.2 Constrained

We created a total of seven constrained groupings: four T4 and three T6. The difference between each grouping was the number of characters s that we allowed to stray from alphabetical order. The best groupings are shown in Tables 3.2 and 3.3, with out-of-order characters in bold.

While these groupings were all optimized based on their badgram clarity, both clarity scores for the final groupings are presented in Table 3.4. For the T4 groupings, there were only very small gains in badgram clarity for each additional character allowed out

s	Group 1	Group 2	Group 3	Group 4
0	a, b, c, d, e	f, g, h, i, j, k, l, m	n, o, p, q, r	$s, t, u, v, w, x, y, z,'$
1	a, b, c, d, e	f, g, h, i, j, k, l, m	$n, o, p, \underline{\mathbf{s}}$	$q, r, t, u, v, w, x, y, z,'$
2	a, b, d, e, f	$\underline{\mathbf{c}}, g, h, i, j, k, l, m$	$n, o, p, \underline{\mathbf{s}}$	$q, r, t, u, v, w, x, y, z,'$
3	a, b, d, e, f	$\underline{\mathbf{c}}, g, h, i, j, k, l$	$m, n, o, \underline{\mathbf{s}}, \underline{\mathbf{w}}$	$p, q, r, t, u, v, x, y, z,'$
4	a, b, e, f, g	$h, i, j, \underline{\mathbf{n}}, \underline{\mathbf{s}}$	$\underline{\mathbf{c}}, \underline{\mathbf{d}}, k, l, m, o$	$p, q, r, t, u, v, w, x, y, z,'$

Table 3.2
The best constrained T4 groupings (by badgram clarity) found by fully enumerating the search space. Characters out of alphabetical order are in bold and underlined.

s	Group 1	Group 2	Group 3	Group 4	Group 5	Group 6
0	a, b, c, d	e, f, g, h	i, j, k, l, m	n, o, p, q	r, s	$t, u, v, w, x, y, z,'$
1	a, b, c, d	e, f, g, h	i, j, k, l, m	n, p, q, r	$\underline{\mathbf{o}}, s$	$t, u, v, w, x, y, z,'$
2	a, c, d	$\underline{\mathbf{b}}, e, f$	g, h, i, j, k, l	$m, n, \underline{\mathbf{r}}$	o, p, q, s	$t, u, v, w, x, y, z,'$
3	a, b, c, d	e, f	g, h, j, k, m, n	o, p, q, s	$\underline{\mathbf{l}}, t, u, v$	$\underline{\mathbf{i}}, \underline{\mathbf{r}}, w, x, y, z,'$

Table 3.3
The best constrained T6 groupings (by badgram clarity) found by fully enumerating the search space. Characters out of alphabetical order are in bold and underlined.

of order after the first. When the WER clarity was measured, it actually decreased when a fourth character was allowed out of order due to the differences between the two metrics. If we had been able to optimize using the WER clarity, this would not have been the case. For the T6 groupings, each additional character allowed to move contributed to small gains on both clarity metrics. When compared to the Qwerty-based T6 groupings, all alphabetically-constrained T6 groupings had much higher clarity metrics, but none of the T4 groupings were particularly close. In all of the T6 constrained groupings produced here, the five core vowels were all separated, though one of them was paired with y in each grouping.

Groups	Out of order	Badgram clarity	WER clarity
4	0	0.8030	0.7872
4	1	0.8112	0.8046
4	2	0.8130	0.8093
4	3	0.8136	0.8170
4	4	0.8148	0.8143
6	0	0.8923	0.8874
6	1	0.8960	0.8886
6	2	0.8984	0.8975
6	3	0.9007	0.8994

Table 3.4
Both clarity scores for the top constrained groupings, with different numbers of characters allowed to be out of alphabetical order. All groupings were optimized with respect to Badgram clarity.

3.4 Discussion and Limitations

Optimizing for badgram clarity, our unconstrained T4 and T6 groupings were able to achieve scores of 0.8191 and 0.9067, respectively. For comparison, the badgram clarity score of the Qwerty-based grouping used in Chapter 2 was 0.8427. While our optimized T6 grouping was able to beat this, the T4 grouping was not. From this, we would expect to encounter more collisions using our freely optimized T4 groupings than we did with the initial Qwerty-based T6 groupings. However, the badgram clarity metric does not fully represent the collisions we could expect to occur. Using the WER clarity metric, the Qwerty-based T6 groupings registered a score of 0.8418. In this case, both our T4 and T6 freely-optimized groupings were able to outperform the baseline previous groupings (WER clarity scores of 0.8542 and

0.9346, respectively), showing that it is feasible to decrease from six groups to four.

For the alphabetically-constrained groupings, the gains seen in the clarity metrics for each additional character out of order were quite marginal. Since each character out of order would make the groups more difficult for a user to remember, it is our opinion that it would not be worth this additional cost to move any characters out of their alphabetical order.

From our observations on the separation of vowels in both our freely-optimized and constrained groupings, we conjecture that one of the factors leading to the poor performance of the Qwerty-based T6 groupings was the fact that four of the vowels (y, u, i, and o) were located in a single group.

Seeing such a large number of random starts ultimately converge to the same grouping in the badgram-based 2-opt algorithm supports the hypothesis that we were able to find the global optima for the badgram clarity metric. While we cannot prove this without fully enumerating all groupings, the fact that no additional swaps were made by the 5-opt algorithm adds further support to this claim.

One of the two main limitations in these studies was the need to iterate through the test set to compute WER clarity. Ideally, we would have been able to perform unconstrained WER-based 2-opt on more initial groupings in hopes of showing the convergence we did with the badgram clarity. Additionally, we were unable to use

WER clarity at all in our constrained optimizations since we were fully enumerating the search space, which resulted in factorial growth. It is possible that we would see more gains with increased numbers of characters out of order, especially in the T4 groupings. If we were optimizing on WER clarity, it would have been impossible for the metric to decrease with an additional character swap. However, based on the average computation time for WER clarity on a set of groups (about 5.5 seconds), constrained optimization with only two characters out of order would take about 2.5 years. We were also unable to optimize constrained T6 groups with four characters out of order, even with badgram clarity, again due to the factorial growth. Future work could explore bounding the search space using integer programming, as done by Karrenbauer and Oulasvirta [37], to be able to reduce the computation time required.

The other limitation is the assumption we make on user input. The user input is simulated and assumed to be perfect, which is not the case in practice. To fully explore the performance of these groupings, we need to test them with real users, which we do in Chapter 4. In Chapter 7, we then use that user data to tune our decoder to model the errors that users make. In these optimization experiments, we did not explore the possibility of users selecting suggested words in the middle of entering a word. To best explore this possibility, we could also add an optimization metric for keystroke savings that calculates the number of keystrokes needed for the true word to be the most likely word suggested by the decoder. Testing this in a user study would require further research in the design of nonvisual word predictions,

which we explore in Chapter 5.

Chapter 4

FlexType User Study

In the previous chapter, we optimized both alphabetically constrained and unconstrained groupings of characters into four and six groups. We showed that it would be feasible to decrease from the six groups used in Chapter 2 to four groups. In this chapter, we will explore this decrease, testing our best T4 constrained and unconstrained groupings in a longitudinal user study with the interface we call *FlexType*.

4.1 System Description

Based on our results in our offline experiments in Section 3.3, we opted to use the constrained grouping with no characters out of alphabetical order in our user study.

Due to the computational complexity required to calculate WER clarity for each possible set of groups, these were optimized with respect to badgram clarity. For our unconstrained groups, we chose the grouping with the best WER clarity metric since it is more representative of likely disambiguation accuracy when writing sequences of words with a recognizer that utilizes a long-span language model. Since our badgram-optimized grouping was unable to produce a higher WER clarity score than our top WER-optimized grouping, we chose to proceed with the WER-optimized grouping. While past work has shown that users perform better initially on familiar keyboards, in this study we wanted to evaluate how long this benefit would persist and how large it might be in a location-independent ambiguous interface. We also wanted to evaluate the extent to which the differences in the WER clarity metric would impact real user performance.

We used a OnePlus 5T smartphone for the user study. To enter text with the Flex-Type interface, users tapped with between one and four fingers to designate one of four character groups. The characters in each group differed between the two experimental conditions. In our CONSTRAINED condition we used the following set of groups:

$$(a, b, c, d, e), (f, g, h, i, j, k, l, m), (n, o, p, q, r), (s, t, u, v, w, x, y, z, ')$$

For our UNCONSTRAINED condition, we used the groups that were optimized on the

WER clarity metric:

$$(a, d, f, h, k, q, y, '), (b, c, e, i, j, n, x), (g, l, o, s, v, w), (m, p, r, t, u, z)$$

The FlexType interface occupied the entire 68 mm × 137 mm screen and the phone displayed solely a solid black background. No visual feedback was provided during text entry to ensure nonvisual text entry. During the input of a word, the following gestures were available:

- Tap with 1–4 fingers — Selects a character from the corresponding group. After each tap, the device reads all characters in the selected group via text-to-speech.
- Swipe left with one finger — Backspaces a single character.
- Swipe left with two fingers — Backspaces all characters in the current word.
- Swipe right with one finger — Indicates that the current word is finished and disambiguates the tap sequence.

Disambiguation was performed by software based on the VelociTap decoder [64]. As described in Section 2.1.1, we used a 12-gram character language model and a 4-gram word language model with a 100K vocabulary. The decoder produced the six most likely words (referred to as the n-best list) that matched a user's exact tap sequence. The most likely result was read via text-to-speech and added to the currently written

text, which was invisible to the user. Immediately following disambiguation, the following gestures were available, in addition to the gestures listed previously:

- Swipe up with one finger — Iterates forward in the n-best list, reading the next most likely word and replacing the current word in the text entry area.
- Swipe down with one finger — Iterates backwards in the n-best list, reading the previous word in the list and replacing the current word in the text entry area.
- Swipe down with two fingers — Signals that there is no more text to enter and advances to the next text entry task.

An example input sequence for the word 'man' can be seen in Figure 4.1.

4.2 Procedure

Since our aim was to compare the learning curves of participants on two different sets of character groups, we opted for a between-subjects design to avoid order effects and the risk of participants confusing the two sets. We had a total of sixteen participants, with eight each in the CONSTRAINED and UNCONSTRAINED conditions.

Each participant completed a total of eight sessions. Each session lasted approximately one hour and participants received US$10 per session as compensation. No

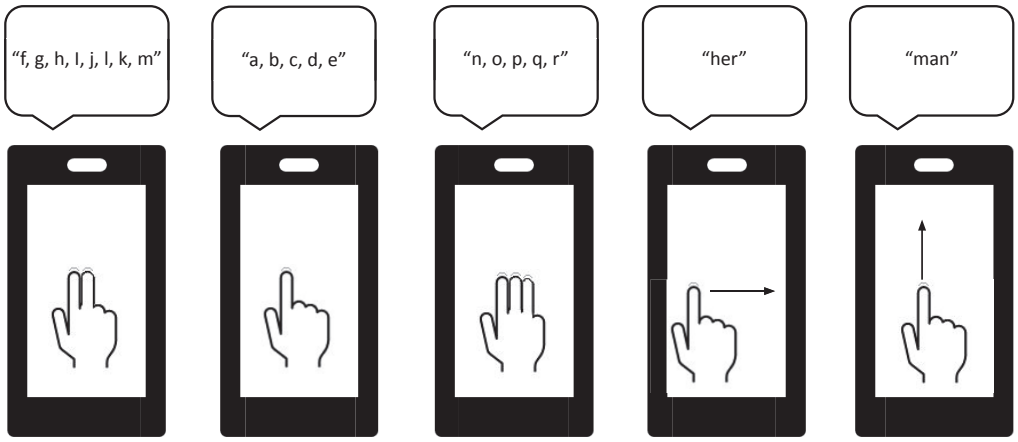


Figure 4.1: An example input sequence for the word ‘man’. From left to right, the user enters each character by tapping with two fingers, one finger, and then three fingers. The letters corresponding to a group are read out after each tap. The user then swipes to the right and the word ‘her’ is recognized as most likely and read out. The user then swipes up to change ‘her’ to the second most likely word, ‘man’.

participant completed more than one session on a single day, and participants had no more than two days between sessions. Within each session, participants took a short break approximately every ten minutes to reduce fatigue.

In each session, participants transcribed text using the experimental interface on a smartphone device. Each prompt was both displayed on the screen and read via text-to-speech. The prompt was displayed in order to provide the users clarity on spelling and to reduce errors resulting from the transcription task itself as opposed to the text entry method. Once the user selected the ‘Start’ button, the FlexType interface blocked the entire display. The prompts that the participants received varied based on the session they were completing. Since one of the character groupings we were

testing was unfamiliar to participants, we slightly modified the progression of sessions from Chapter 2, adding a session where participants entered single-letter prompts. The full progression of sessions was as follows:

- Session 1 — Participants received single-letter prompts. The goal of this session was to teach the participants the groups. Participants did not perform disambiguation.

- Session 2 — Participants received single-word prompts with full alphabet coverage. This was designed to continue teaching them the groupings while familiarizing them with the disambiguation interaction. These prompts were pruned to remove any words that did not appear as the first result in the decoder's n-best recognition results (assuming the participant made no mistake while entering the word).

- Session 3 — Participants received phrase prompts that were pruned to contain no more than four words, each no longer than six characters. The prompts were no longer pruned to remove words not appearing as the first result in the decoder's recognition results.

- Sessions 4–8 — Participants still received phrase prompts. Prompts were only restricted to be no longer than six words to aid participants in remembering them correctly.

All phrase prompts were drawn from the Enron mobile data set [62].

After finishing each transcription, participants swiped down with two fingers to complete the input. The participant was shown a summary screen with the prompt text, their entered text, and their entry and error rates before they advanced to the next prompt.

4.3 Results

We recruited twenty participants for our between-subjects study via convenience sampling. Four participants withdrew from the study prior to completion for a total of sixteen complete participants (eight in each condition). The incomplete participants' data was excluded from analysis. Participants in the CONSTRAINED condition were 19–22 years old (mean 21). One identified as male and seven identified as female. Participants in the UNCONSTRAINED condition were 20–45 years old (mean 23.88). Six identified as male and two as female. All participants were students or staff at a university and rated the statement "I consider myself a fluent speaker of English" a 7 on a 7-point Likert scale where 7 was strongly agree.

Our independent variable was the groupings of characters, CONSTRAINED or UNCONSTRAINED. As our dependent variables, we measured different metrics of user

Condition	Entry rate (WPM)	Error rate (% CER)
CONSTRAINED	12.0 ± 4.4 [7.4, 20.8]	2.03 ± 1.31 [0.66, 4.53]
UNCONSTRAINED	13.5 ± 3.3 [9.9, 19.8]	1.81 ± 0.91 [1.01, 3.82]
Statistical test	$t(14)$ = -0.75, $p = 0.46$	$t(14) = 0.40$, $p = 0.69$
Effect size	$d = 0.38$ (small)	$d = 0.20$ (small)

Table 4.1
The main results from sessions 5 through 8 of the user study. Results are reported in the format *mean ± sd [min, max]*. The statistical tests reported are independent means t-tests and effect sizes are measured by Cohen's d.

Condition	Backspaces per char	N-best explorations per word
CONSTRAINED	0.090 ± 0.043 [0.037, 0.154]	0.318 ± 0.111 [0.180, 0.468]
UNCONSTRAINED	0.111 ± 0.037 [0.069, 0.174]	0.288 ± 0.080 [0.186, 0.447]
Statistical test	$t(14)$ = -1.06, $p = 0.31$	$t(14) = 0.61$, $p = 0.55$
Effect size	$d = 0.53$ (medium)	$d = 0.30$ (small)

Table 4.2
The corrective action results from sessions 5 through 8 of the user study. Results are reported in the format *mean ± sd [min, max]*. The statistical tests reported are independent means t-tests and effect sizes are measured by Cohen's d.

performance and behavior. Since the beginning sessions were designed to train the users on the interface, the metrics that we calculated represent the average of the final four sessions. We excluded from analysis 105 tasks (out of 16,383) across the entire study in which technical issues impacted either participants' ability to complete the task or the data logging for that particular task. No more than 6 tasks were excluded from any one session, and no more than 19 tasks from any one participant.

4.3.1 Entry Rate

First, we measured participants' entry rate in words per minute (WPM), where a word is assumed to be five characters, including a space. Since participants only entered single characters in the first session, we cannot calculate an entry rate. As shown in Figure 4.2, participants' entry rates increased through the sessions but may have started to plateau towards the end. Participants in the CONSTRAINED condition were able to achieve an average of 12.0 words per minute across their final four sessions, while participants in the UNCONSTRAINED condition averaged 13.5 words per minute. An independent means t-test showed that this difference was not significant. Details can be found in Table 4.1.

4.3.2 Error Rate

The next metric we measured was error rate. We report character error rate (CER) as the number of insertions, deletions, and substitutions required to transform the input text to the reference text, divided by the length of the reference text. We computed this on participants' final text, after they made corrections. As shown in Figure 4.3, participants' error rates varied throughout their sessions with most participants having a CER of less than 5% in most sessions. Across the final four

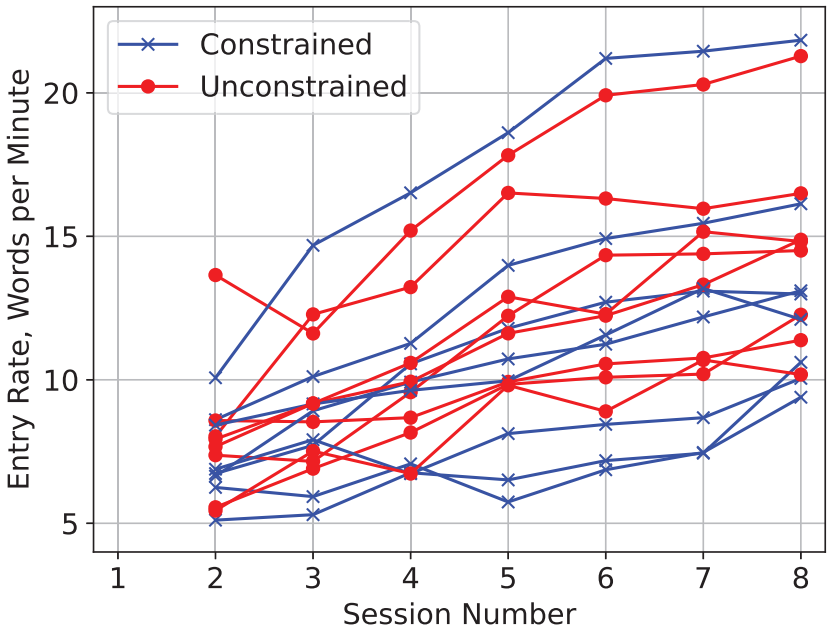


Figure 4.2: Entry rates of each participant throughout the study. Session 1 entry rates are not plotted since participants only entered single letters in this session.

sessions, participants in the CONSTRAINED condition averaged 2.03% character error rate while participants in the UNCONSTRAINED condition averaged 1.81%. As with entry rate, this difference was not significant. Details can be found in Table 4.1.

To compare individual participants' entry rates to their error rates, we created the scatter plot shown in Figure 4.4. While we expected to see a trade-off between speed and accuracy, we actually saw the opposite. In general, as entry rate increased, participants' error rates decreased. The two participants with the fastest entry rates in the final four sessions, 20.8 and 19.8 WPM, had among the lowest error rates (both 1.0% CER), with only two other participants having lower error rates. The participant with the slowest entry rate (7.4 WPM) also had the highest error rate, at

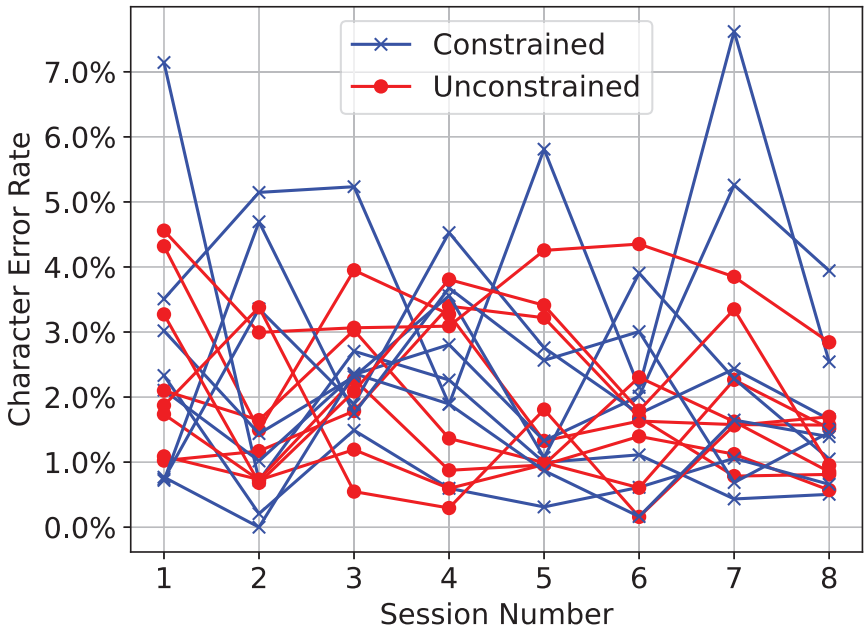


Figure 4.3: Character error rates of each participant throughout the study.

4.5% CER.

4.3.3 Backspaces per Character

As a metric of corrected taps, we measured backspaces per character (BPC), which is the total number of characters backspaced divided by the final number of output characters. This metric takes into account the total characters deleted by both single character (one-finger) and word-at-a-time (two-finger) backspaces. As shown in Figure 4.5, the BPC was quite high in the first session at 0.162 and 0.714 in the CONSTRAINED and UNCONSTRAINED conditions, respectively. The BPC dropped in the second and third sessions and remained relatively constant for the remainder of

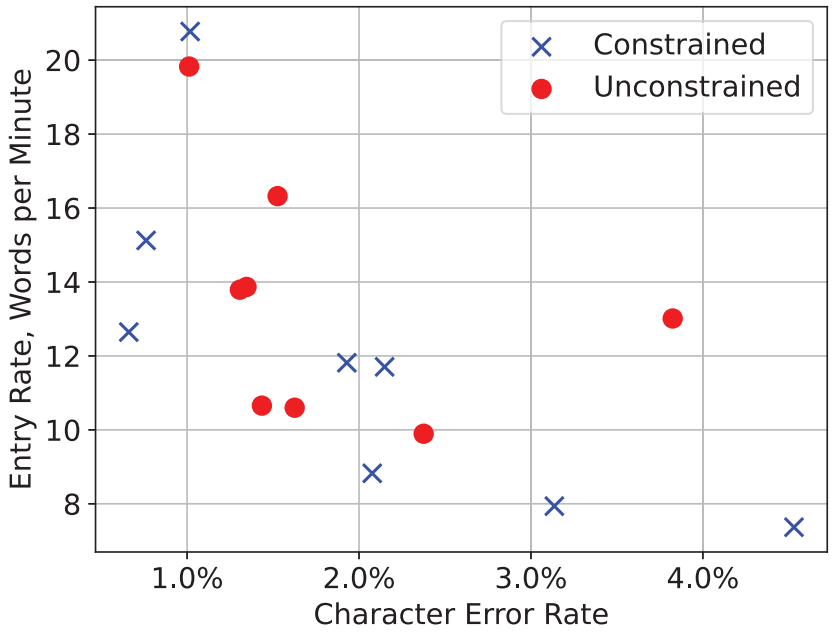


Figure 4.4: Scatter plot showing each participant's entry rate compared to their error rate for the final four sessions.

the study. As expected from prior work on familiarity constraints, the participants in the UNCONSTRAINED condition had a significantly higher BPC in the first session ($t(14) = -4.66, p < 0.001$) with a large effect size (Cohen's $d = 2.33$). There were no significant differences in any of the remaining sessions, or in the combined evaluation sessions, where participants averaged 0.090 backspaces per character in the CONSTRAINED condition and 0.111 in the UNCONSTRAINED condition (see Table 4.2).

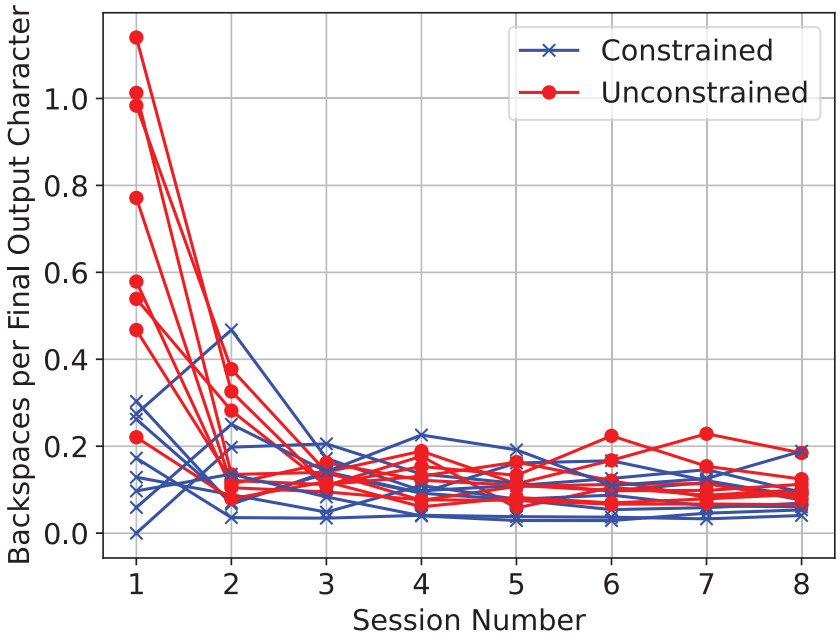


Figure 4.5: Backspaces per final output character for each participant during the user study.

4.3.4 N-Best Exploration

To analyze participant behavior following disambiguation, we totaled the number of up and down swipes participants used to iterate through the n-best list and normalized based on the final number of words that were input (including words that were later deleted). We measured this from session 3 on since the n-best list was not used in the first two sessions. As shown in Figure 4.6, participants' total swipes per word in each session was relatively stable and hovered around 0.2–0.5 swipes per word. In sessions 5–8, participants using the UNCONSTRAINED groups iterated through the n-best list slightly less at 0.288 swipes per word compared to 0.318 for the CONSTRAINED

participants, though this difference was not significant (see Table 4.2).

We conducted further analysis measuring the number of swipes following words that were entered with the correct tap sequence. Since the goal of this was to evaluate the disambiguation algorithm, we eliminated cases where there was an error in a previously entered word, since this would impact the disambiguation results. This metric was nearly identical between the two conditions, with an average of 0.169 swipes per properly entered word in the CONSTRAINED condition and 0.168 swipes in the UNCONSTRAINED condition. This was not significantly different ($t(14) = 0.03$, $p = 0.97$) and the effect size was negligible (Cohen's $d = 0.017$). Because this metric was so similar between the two conditions, it appears as though the UNCONSTRAINED groups were unsuccessful in reducing the exploration through the n-best list.

4.3.5 N-Best Distribution

Finally, we analyzed each word entered with the proper tap sequence and context to determine the distribution of the target word's position in the n-best list. We averaged the proportion of words found at or before each position (e.g. position 2 includes words found in either position 1 or position 2) for each participant. The graph of this cumulative distribution can be seen in Figure 4.7. As we expected from our optimization experiment, this proportion was higher for the UNCONSTRAINED

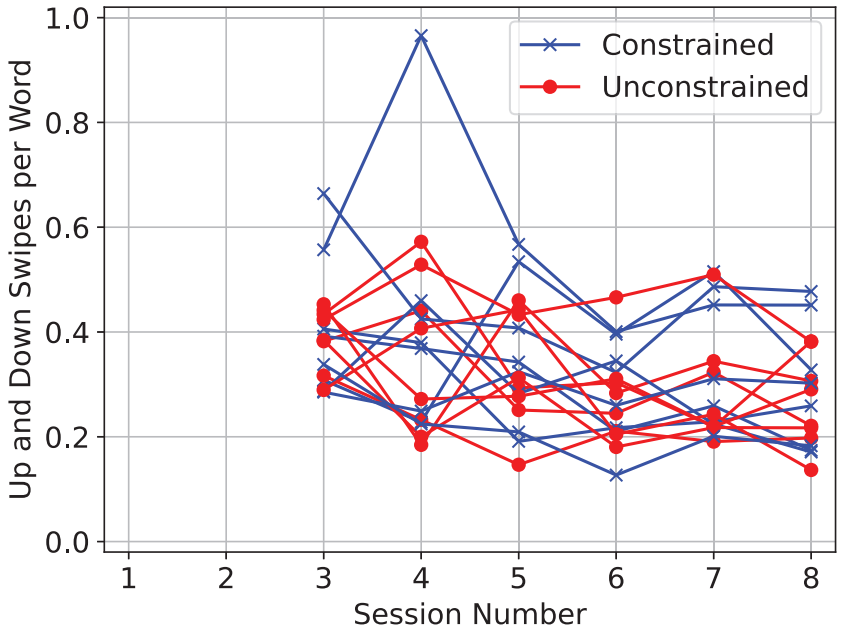


Figure 4.6: Total number of explorations through the n-best list per final input word for each participant. Sessions 1 and 2 did not use the n-best list and are not plotted.

condition for position 1 (the top disambiguation result). It was interesting to note that it was very similar for the remainder of the positions. The full non-cumulative distribution is reported in Table 4.3.

Using this distribution, we calculated the expected number of swipes per correct word users in each condition would perform if they swiped perfectly each time (never passing their target word in the list). We found this to be 0.172 for the CONSTRAINED condition and 0.152 for the UNCONSTRAINED condition. Although it was a small difference, it is interesting to note that the 0.169 swipes per word participants in the CONSTRAINED condition performed was less than the expected 0.172 swipes per word given the prompts that they entered. This means that on average, CONSTRAINED

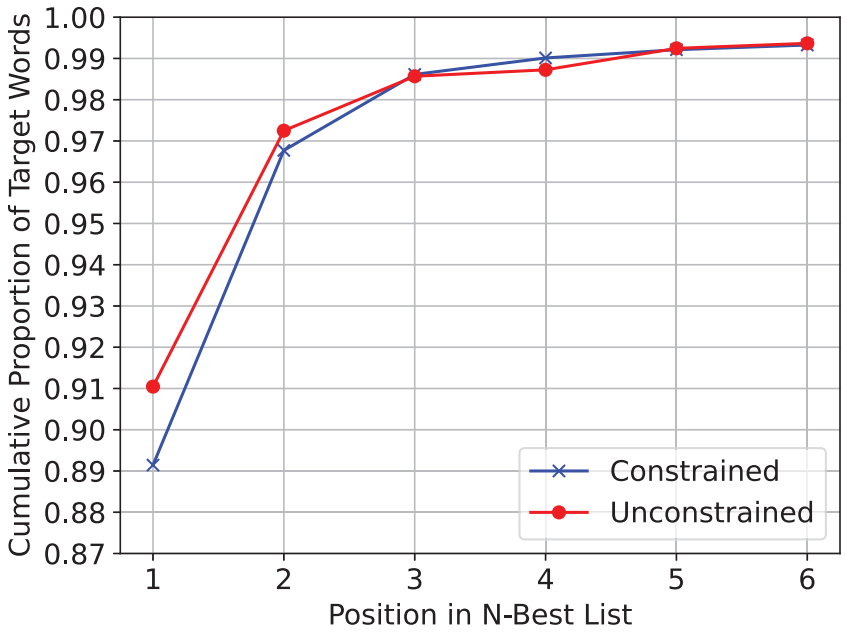


Figure 4.7: The cumulative proportion of intended words located at or before each position in the n-best list.

participants did not navigate far enough through the n-best list to reach their target word or to conclude that it was not present. One possible explanation for this is that participants may have immediately backspaced a word without exploring the n-best list if they thought that they made an error. Another possible explanation is that participants may have not bothered exploring the n-best list after typing a word that they had previously encountered and learned would not be returned by the decoder.

A reason that users in the UNCONSTRAINED condition swiped more than expected may have been that they explored the n-best list too quickly and needed to go backwards in the list to get to their intended word. While 9.1% of swipes from users in the UNCONSTRAINED condition were down swipes, only 4.5% of swipes from users in

Condition	Pos. 1	Pos. 2	Pos. 3	Pos. 4	Pos. 5	Pos. 6	Not in list
CONSTRAINED	89.14%	7.63%	1.84%	0.40%	0.20%	0.11%	0.67%
UNCONSTRAINED	91.05%	6.20%	1.32%	0.16%	0.52%	0.12%	0.63%

Table 4.3
The distribution of intended word positions in the n-best list when they were entered with the correct tap sequence and context.

the CONSTRAINED condition were down swipes. This difference was significant ($t(14)$ = -2.34, p = 0.035) with a large effect size (Cohen's d = 1.17), and does show that users in the UNCONSTRAINED condition navigated backwards through the n-best list more often.

4.4 Discussion

The goal of this work was to compare user performance between our constrained and unconstrained optimized groups from Chapter 3. As we expected to find, participants struggled with the unconstrained groups more in the beginning as evidenced by the significantly higher backspaces per character metric in their first session. However, Figure 4.5 shows that from session 2 on, all of the participants had quite similar backspace rates. This suggests that the benefit of the familiarity of the constrained groups may be reduced after the first hour of practice. While after the final session one participant in the UNCONSTRAINED condition stated, "Some letters like 't' and 'x' were hard for my brain to remember", another commented, "At first it was just a

matter of memorization and then it was totally natural." The latter comment shows that some participants were more open to learning the UNCONSTRAINED groups than others.

The theoretical benefit of a particular ambiguous set having a higher WER clarity metric is that during entry, participants will need to navigate through the n-best list less often. As we showed in Table 4.3, the UNCONSTRAINED set did have a slightly higher proportion of words that did not require exploration of the n-best list. Interestingly, participants in both conditions explored the n-best list similar amounts, with participants in the UNCONSTRAINED condition using it slightly less. While again this difference was not significant, it does seem to align with the slight difference shown in the optimization metrics.

A limitation of our interface was that the n-best list did not always contain the users' target words. If a user's intended word was not in the decoder's vocabulary, or simply less likely than other words with an identical tap sequence given the context, users were unable to enter that word correctly. As shown in Table 4.3, this occurred in about 0.65% of words across both conditions where the user entered the proper tap sequence with the correct prior text. To remedy this issue in our final user evaluation in Chapter 8, we will add a mode where users can enter text one letter at a time, swiping up and down after each tap to designate the exact desired character from their selected group. While this will likely slow entry rates, it will provide a means

for accurate entry of words that that are hard for the decoder to predict, such as proper names.

While FlexType could be implemented using a variety of sensors, we used a touchscreen in our user study. This led to some ergonomic issues, with one participant remarking that "The four finger tap was always a little bit of a stretch. I tended to need to shift my hand position to make the gesture." Three other participants also mentioned tapping with four fingers when asked about interactions that felt unnatural or were hard to learn. For future studies involving a touchscreen, it may be useful to explore a different gesture for selecting the fourth group, or to optimize a set of three groups of characters. Holding the phone in a different orientation could also make this gesture a little more natural. Gestures that do not require a touchscreen could include finger-to-thumb touches, detected by either pressure sensors [33] or gloves with conductive fabric [9].

Due to the longitudinal nature of the user study, we were unable to run all of the participants at one time. Because of this, the participants were assigned to alternating conditions in the order that they were recruited. By chance, this led to an imbalance of male and female participants between the groups. While this could create a potential confound, we do not have reason to believe there is a performance difference driven by gender identity.

Across both conditions, participants averaged 12.8 words per minute and 1.92% character error rate using single-handed text entry without visual feedback. It can be difficult to make direct comparisons between studies due to differences in experimental procedure and the amount of practice participants have with each interface. That being said, these performance metrics are similar to those of other eyes-free text entry methods (e.g., Graffiti: 10.0 WPM [55], Perkinput: 17.6 WPM with one hand [6], and TipText: 13.3 WPM [66]). Our final user evaluation in Chapter 8 will conduct an experiment with blind participants that directly compares FlexType to users' typical nonvisual text entry methods.

4.5 Takeaways

In a multi-session user study, we evaluated the optimized groups from Chapter 3 and found that our unconstrained groups, while they had a slightly better clarity metric, did not perform meaningfully better than alphabetically-constrained groups. Since the unconstrained groups did not produce a noticeable benefit in our long-term evaluation, we believe it is not worth the higher barrier to entry that they create. This barrier is exemplified by users of the unconstrained groups backspacing over four times more often in their first hour of use. In open feedback following the final session, one participant in the CONSTRAINED condition remarked, "I wouldn't be opposed to using this as a keyboard option on my own phone, it was fun to use." This sentiment

highlights the potential for widespread adoption of this technique in situations where visual feedback is not available or motor gestures are limited. In Chapter 6 we solicit feedback on this text entry method from blind users and incorporate that feedback into the next iteration of the interface in Chapter 8.

Chapter 5

Audio Word Suggestions

5.1 Motivation

In our initial study of the Tap123 interface in Chapter 2, participants noted difficulty selecting their intended word from the list of likely words returned by the decoder. Our optimization work in the previous two chapters aimed to increase the likelihood of predicting the correct word. However, the correct word is still not a guarantee and there will still be instances that require users to select a word other than the one initially returned.

Commercial touchscreen keyboards place a number of word suggestions, commonly three or four, just above the top row of keys. This allows the user to quickly glance

and check them without distracting too much from the task of typing. However, if these words are read sequentially using standard text-to-speech software, it would take quite some time to finish; the user would likely be able to type the next key or the remainder of the word by the time the list was finished being read.

In FlexType, suggested words and word completions could be offered with each keystroke prior to recognition, or if the user pauses. If the user hears the word they are typing they could select it immediately, without needing to finish the remainder of the taps. This would operate similar to many commercially available standard touchscreen keyboards, but using audio instead of visual feedback.

To attempt to present suggested words more efficiently, we investigate the possibility of providing the user with audio-only word suggestions using simultaneous voices. This would enable the interface to provide word suggestions much quicker and help to further increase the efficiency of this entry technique. In this work, we will begin with a perceptual study to measure user performance when listening for a target word among several potentially similar words. We want to determine if users are able to detect when their target word is spoken and identify which voice said it. If this method shows promise, we could then integrate this suggestion technique into a text entry interface to see how it impacts users' text entry performance.

5.2 Related Work

There have been several studies on the human ability to perceive speech amid other speech, the first dating back to Cherry in 1953 [13]. Cherry conducted a series of experiments on the *Cocktail Party Effect*, with the aim of identifying the factors that contributed to the phenomenon. The results of the studies showed that participants were able to clearly listen to either of two spoken messages played into opposite ears via headphones (each was recorded by the same speaker). However, participants were not able to notice when the language of the message in the ear that they were not listening to (the “rejected" ear) changed from English to German. This prompted further studies, in which the author found that participants were able to correctly identify if the rejected message was normal human speech (as opposed to reversed speech or an oscillating tone), as well as whether it was a male or female voice. They were not, however, able to identify any words in the rejected message or what language it was in.

This line of work was continued by Egan et al. [20], who played sentences simultaneously for participants. Each of these sentences contained one of two “call signs", and the participants were instructed to write down the words following a particular call sign, ignoring the sentence that contained the other call sign. In one experiment,

the authors' results indicated that high-pass filtering of either message allowed participants to better understand the target message. This suggests that users will be able to understand simultaneous speech better if there are differences in the sound frequencies. Another experiment confirmed the results found by Cherry [13] that participants were better able to understand the target message when the two messages were played in opposite ears instead of the same ear.

Brungart [11] conducted experiments using phrases of a set structure. They each contained one of eight call signs, followed by one of four colors, and finally one of eight numbers. Two phrases were played simultaneously, and participants were instructed to listen for the phrase with a particular call sign (the target phrase) and report the corresponding color and number. The author varied the speakers of the target phrase and the other (masking) phrase, finding that participants were best able to distinguish the two when the speakers were of different sexes. Participants performed the worst when the same speaker was used for both the target and masking phrases, and performance was between these two extremes when the phrases were spoken by different speakers of the same sex. Further work by Darwin et al. [15] attribute this performance difference to the combination of differences in vocal tract length and fundamental frequency between speakers.

Brungart and Simpson [10] experimented with two, three, and four speakers in different spatial arrangements around the listener. They used digital signal processing and

head-related transfer functions to "move" the stimuli in space around the users' heads. They used the same phrase structure (call sign, color, number) that Brungart did in their earlier work [11]. Their results showed that participants were 92% accurate with two speakers, 72% with three speakers, and 62% with four. Additionally, they found that participants scored higher in spatial configurations where the speakers were more spread out, and specifically when the target phrase came from either the left or right side as opposed to middle.

Guerreiro and Gonçalves [27] adapted this work to the field of accessible computing, testing how well blind people could understand concurrent speech using screen readers. They used both different-sex voices as well as spatial separation to increase the users' ability to understand the phrases. The authors also increased the speech rates of the voices with the aim of presenting as much information to the user as quickly as possible (while still maintaining comprehension). They found that using two or three voices with slightly faster speech rates had a larger information bandwidth than using a single voice with a much faster speech rate. They reported that the best combination of comprehension and speed was using two voices at either 1.75 or 2 times the default speech rate.

While these past studies have explored various factors that contribute to the clarity of simultaneous speech, they have all done so using phrases or full sentences. When designing a text entry interface for word suggestions, we are more concerned with

the clarity of single words. Additionally, the problem is complicated by the fact that word suggestions tend to be more similar to one another than randomly selected words would be (i.e. if a user has begun to type a word, several of the suggestions will likely start with that prefix). Nicolau et al. [45] investigated concurrency while exploring the design space of nonvisual word completions. However, they reported qualitative feedback from users and did not report any investigation into the quantitative accuracy of users with this type of audio. In their analysis of users' feedback, Nicolau et al. identified the theme that word discrimination was the biggest drawback to concurrent word suggestions.

5.3 Study 1

To quantify this drawback, we developed a study to determine how well users were able to comprehend single words among other simultaneous words. We asked participants to wear headphones for the duration of the study. In each trial, we presented users with a target word. They then listened to between two and four voices each speak a word and selected which voice, if any, they thought spoke the target word. All of the voices we used originated from the Android operating system's built in text-to-speech service:

- Voice 1 — English US, Male 1

- Voice 2 — English US, Female 1
- Voice 3 — English US, Male 3
- Voice 4 — English US, Female 2

There were an equal number of trials in each condition with two, three, and four voices, but the order they occurred within each condition was randomized. There were three counterbalanced conditions in our within-subjects study:

- 180Degree — Voice 1 and Voice 2 played entirely in the participants' left ears, while Voice 3 and Voice 4 played entirely in the participants' right ears (creating 180 degrees of separation between each pair).
  - For trials with two voices, we used Voice 1 and Voice 4.
  - For trials with three voices, we used Voice 1, Voice 2, and Voice 4.
  - Trials with four voices used all four voices.

  The aim of this ordering was to maximize both pitch and spatial differences between playing voices. All voices began playing simultaneously.
- 60Degree — Voice 1 and Voice 4 remained entirely on the left and right, respectively. Voice 2 was located 30 degrees to the left of center and Voice 3 was 30 degrees to the right of center (creating 60 degrees of separation between each voice).

- VOICE 1 used only the left audio channel.
- VOICE 2's location was simulated by splitting the audio into left and right channels, adjusting the volume of each, merging then back together, and then normalizing the result to the same total volume as the other voices. The individual channel volumes were adjusted by factors of 0.67 on the left and 0.33 on the right.
- VOICE 3's location was simulated in the same manner as VOICE 2. The individual channel volumes were adjusted by factors of 0.33 on the left and 0.67 on the right.
- VOICE 4 used only the right audio channel.

The same combinations of voices were used as in 180DEGREE and all voices began playing simultaneously.

- SEQUENTIAL — All voices were located directly in front of the participant (equal volume in both ears). Voices were added and played in numerical order (e.g. if there were only two voices, VOICE 1 and VOICE 2 were used). Each voice waited for the previous voice to finish completely before beginning to speak.

Prior to the study, the trials were created based on phrases taken from the Enron mobile data set [61]. To create the words spoken for each trial, a random phrase was chosen from the set, along with a random character position in that phrase. We fed the portion of the phrase up to and including that position (including any partial

word) into the VelociTap decoder [64]. The n-best word completions returned by the decoder (the n words with the highest probability given the context and any prefix) became the words spoken by each voice, where n was the number of voices in that particular trial. We did not allow the random position to be the beginning of the sentence since the decoder would just return the most likely words to start a sentence. If there were not enough words returned by the decoder to fill every voice in a trial a new phrase and position were chosen. We felt that this procedure would produce words that were representative of a user receiving audio word predictions in a real-world text entry task. The partially completed word or next word (if the position fell between words) became the target word. Once the words were selected, which voices spoke them were randomized so that Voice 1 was not always the most likely word returned by the decoder. It is important to note that the target word was not always present in the words that were spoken, as would be the case in real-world text entry.

5.3.1 Procedure

We developed a web application to conduct this study remotely. The application first walked participants through informed consent and a demographic questionnaire. It then provided instructions for the first of the three counterbalanced conditions; the ordering of conditions was predetermined based on participant number. To familiarize participants with the voices and their locations for each condition, we included a

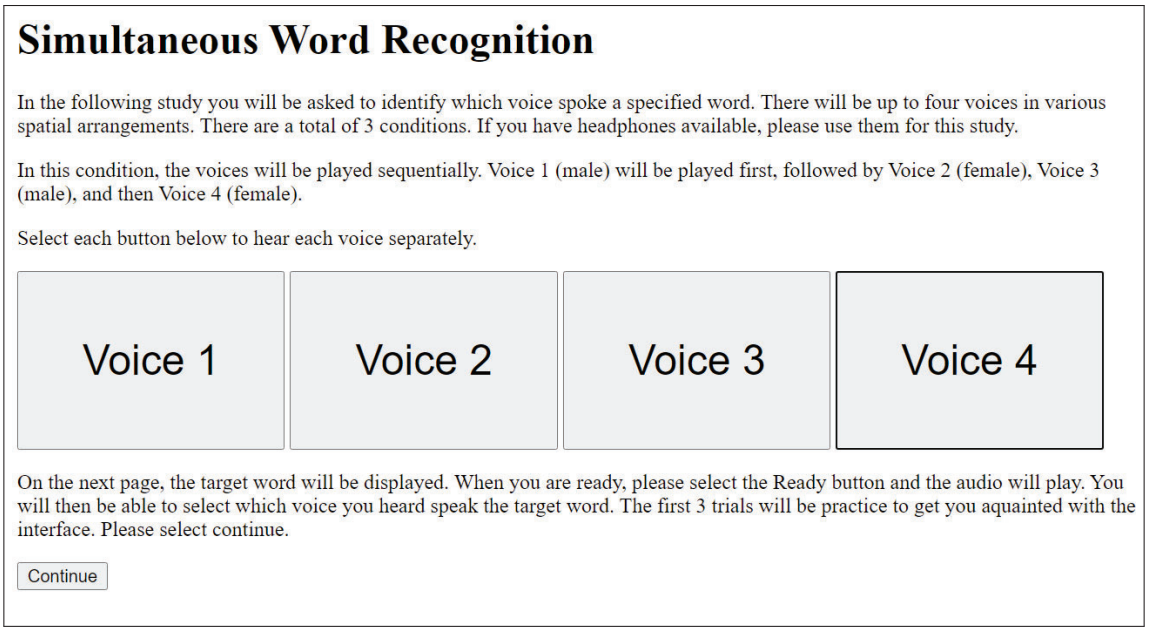


Figure 5.1: The web interface at the beginning of the SEQUENTIAL condition. The text below the Voice buttons as well as the Continue button do not appear until all four Voice buttons have been clicked. This screen is meant to familiarize users with which voice corresponds to which number.

button for each voice on the instruction page, as shown in Figure 5.1. When clicked, these buttons played via text-to-speech 'Voice N', where N was the number of the voice clicked. We synthesized this text-to-speech using the corresponding voice, and spatially located each voice in its position for the current condition. We required participants to click all four buttons before they were allowed to proceed, but they were able to listen to each voice as many times as they wanted. We included this instruction page at the start of each condition.

After listening to each voice on the instruction page, participants then progressed to six practice trials. They were first shown the target word (see Figure 5.2). When the participant clicked the 'Ready' button, the audio for the trial was played and

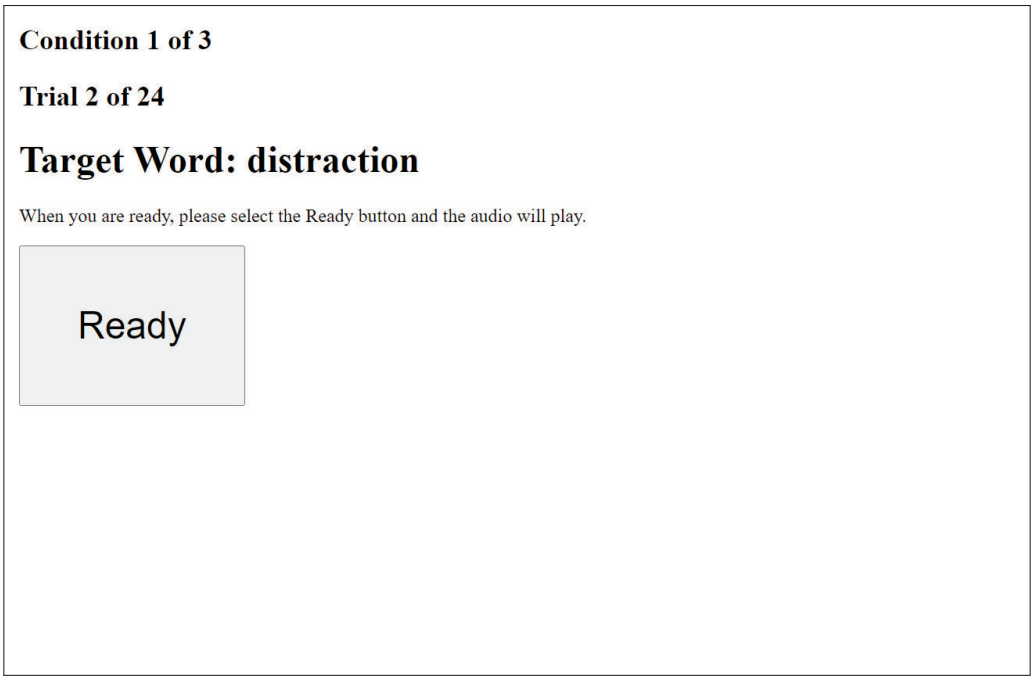


Figure 5.2: The web interface at the beginning of a trial. It displays the target word and allows the participant to prepare to listen. The audio will play once they click 'Ready'.

they were shown six buttons, as shown in Figure 5.3. These corresponded to 'Not Present', the four voices, and 'I Don't Know'. If they answered correctly (Figure 5.4), they proceeded to the next trial. Otherwise, the correct response was shown and the participant repeated the practice trial until they responded correctly. The purpose of this was to ensure that participants were familiar with the voices that were being presented. Practice trials were not included in any of the data analysis. Once the practice trials were complete, participants then completed 24 evaluation trials. The interface was identical to that used in the practice trials, but participants were not able to retry any incorrect trials. At the end of each condition, participants completed a brief questionnaire. They then repeated this process, beginning with the instructions

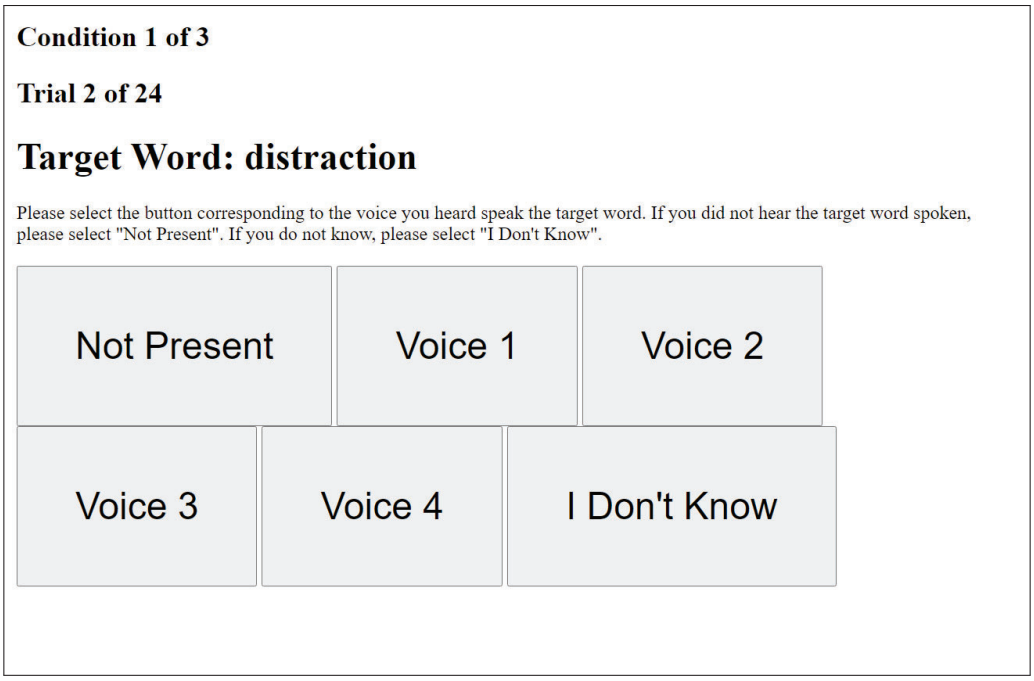


Figure 5.3: The web interface once the participant has clicked 'Ready' and the audio has played. The participant now selects which voice they heard say the target word.

page, for the next condition.

5.3.2 Results

We recruited 24 participants by word-of-mouth. A total of 5 participants did not fully complete the study due to technical issues. We removed their data and recruited new participants to obtain a total of 24 complete participants. They were each paid $10 for their participation. Participants ranged from 18–25 years of age. 9 identified as male, 14 as female, and 1 as other. None of the participants reported having uncorrected

Figure 5.4: The web interface after the participant has indicated their response. Clicking 'Continue' will take them to the next trial.

hearing or visual impairments.

There were two independent variables in this study: spatial arrangement and number of voices. The dependent variable was response accuracy, which we computed as the number of correct responses divided by the number of total responses. The mean response accuracy for each combination of spatial arrangement and number of voices can be seen in Figure 5.5. For the 180Degree condition, participants had an average accuracy of 85%, 65%, 41% with two, three, and four voices, respectively. The results in the 60Degree condition were similar, with an accuracy of 84% for two voices, 61% for three voices, and 43% for four voices. As we expected, the Sequential condition did not have the same decrease in accuracy when more voices were added.

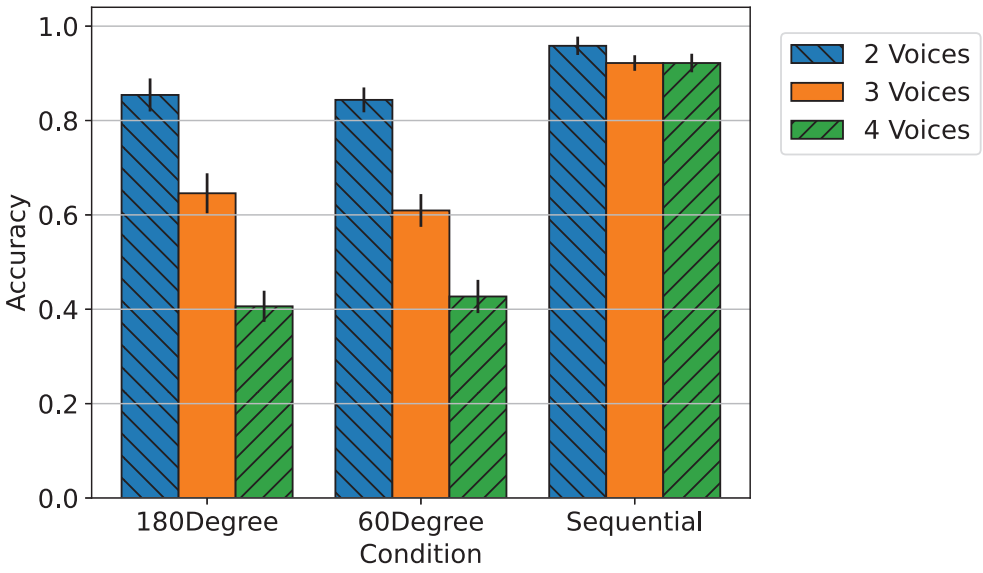


Figure 5.5: The average accuracy of participants in Study 1. Error bars represent standard error of the mean.

Participants had accuracies of 96% (two voices), 92% (three voices), and 92% (four voices).

Mauchly's sphericity test showed that both main effects met the assumption of sphericity ($W = 0.96, p = 0.66$ for spatial arrangement and $W > 0.99, p = 0.99$ for number of voices). Additionally, the interaction also met the assumption of sphericity ($W = 0.82, p = 0.90$), so we did not need to correct the degrees of freedom. A type III ANOVA for our 3 (spatial arrangement) $\times$ 3 (number of voices) design showed a significant difference between spatial arrangements, $F(2, 46) = 83.80$, $p < .001$, as well as number of voices, $F(2, 46) = 102.35$, $p < .001$. More importantly, it showed a significant interaction between our independent variables, $F(4, 72) = 19.38$, $p < .001$.

This indicates that the number of voices had a different effect on participants' accuracy depending on the spatial arrangement used, so we did not summarize within each independent variable.

We ran the pairwise *post hoc* t-tests with Bonferroni correction to determine where exactly the significant differences were found. The results of these tests are shown in Table 5.1. Notably, within both 180Degree and 60Degree the difference between each number of voices was significant, but none of these differences were significant within Sequential. Between the two simultaneous conditions, 180Degree and 60Degree, all differences were significant only where the number of voices was not equal. Finally, the difference between each simultaneous condition with two voices and Sequential with all numbers of voices was not significant, but three and four voices within each simultaneous condition were significantly different from Sequential with all numbers of voices.

5.3.3 Discussion

This first study showed that performance was quite poor in both simultaneous conditions when there were more than two voices, while performance did not significantly change for the control Sequential condition. We were hoping to have success similar to past work by creating both spatial and pitch differences in our speakers. However,

		Condition								
		180Degree			60Degree			Sequential		
Condition	Voices	2	3	4	2	3	4	2	3	4
180Degree	2	-	-	-	-	-	-	-	-	-
	3	**.002**	-	-	-	-	-	-	-	-
	4	**<.001**	**<.001**	-	-	-	-	-	-	-
60Degree	2	1.00	**.010**	**<.001**	-	-	-	-	-	-
	3	**<.001**	1.00	**.007**	**<.001**	-	-	-	-	-
	4	**<.001**	**.010**	1.00	**<.001**	**.024**	-	-	-	-
Sequential	2	.536	**<.001**	**<.001**	.055	**<.001**	**<.001**	-	-	-
	3	1.00	**<.001**	**<.001**	1.00	**<.001**	**<.001**	1.00	-	-
	4	1.00	**<.001**	**<.001**	.786	**<.001**	**<.001**	1.00	1.00	-

Table 5.1
The p-values from pairwise post hoc t-tests in Study 1. Bold values indicate a significant difference ($p < 0.05$).

we conjecture that the difference arose from the short length and similarity of the stimuli. All of the previous studies we examined earlier in this chapter used sentence or phrase stimuli that may have allowed participants to adjust to hearing multiple speakers. Additionally, many previous studies, like those by Brungart [11] and Brungart and Simpson [10], used a very finite set of words in their phrases that were not easily confusable. This was not the case in our study, and that may have also contributed to the poorer performance observed in the simultaneous conditions.

5.4 Study 2

We wanted to further explore audio word suggestions to see if we could provide more than two suggestions. We devised another experiment that introduced a small amount

of temporal spacing between words in the simultaneous conditions. This would still allow a series of suggestions to be presented to the user more quickly than fully sequential, but with potentially better comprehension than fully simultaneous. A similar technique was used by Nicolau et al. [45], but they used a static 250 ms delay and did not report any quantitative comparisons. We used the same voices and trial creation process as in the Study 1. All conditions used the same spatial arrangement as the Study 1's 60Degree condition. Voices were also added in the same order as 60Degree to maximize spatial separation (Voice 1, Voice 4, Voice 2, then Voice 3). The conditions in this study were as follows:

- DelayShort - Voices began playing in numerical order. After each voice began, there was a 0.15 s delay before the next voice began playing.
- DelayLong - Voices began playing in numerical order. After each voice began, there was a 0.25 s delay before the next voice began playing.
- Sequential - Voices played in numerical order. Each voice waited for the previous voice to fully finish before beginning to play.

5.4.1 Procedure

This study used the same web interface as Study 1. Participants completed informed consent, a demographic questionnaire and received instructions before completing six

practice trials and 24 evaluation trials in each condition. There was also a short questionnaire after each condition. To recruit a larger and more diverse set of participants in this study, we recruited 96 participants by posting Human Intelligence Tasks (HITs) on Amazon Mechanical Turk, a crowdsourcing platform. They were paid $3.50 for their time. Similar to the crowdsourcing procedure used by Vertanen et al. [59], we eliminated workers that failed to meet a control standard. The standard we chose was 50% accuracy in the SEQUENTIAL (control) condition. Since there was no overlap between the voices in the SEQUENTIAL condition, this should have been an easy accuracy to achieve. We returned rejected HITs to the pool until we had 96 participants that met the standard. No participant was able to complete more than one HIT in this study, and, to our knowledge, no participants had previously completed Study 1 since we recruited from different pools for each study.

5.4.2 Results

The 96 participants in this study ranged from 21 to 69 years of age. 68 identified as male, and the remaining 28 as female. Given that a total of only three participants identified as low vision, it was somewhat surprising that eight of the participants reported using some form of screen reading software, either on mobile or desktop devices.

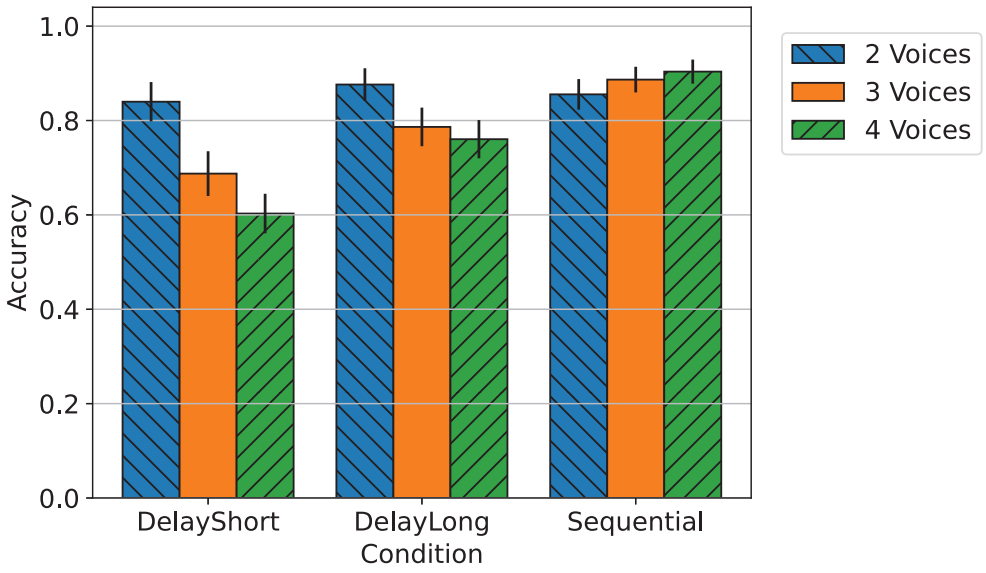


Figure 5.6: The average accuracy of participants in Study 2. Error bars represent standard error of the mean.

In this study, our independent variables were the temporal arrangement and the number of voices. As with the previous study, our dependent variable was accuracy. The mean accuracy results broken down by each combination of temporal arrangement and number of voices can be seen in Figure 5.6. In the DelayShort condition, participants had average response accuracies of 84% with two voices, 69% with three voices, and 60% with four voices. Increasing the delay to 0.25 s resulted in accuracies of 88%, 79%, and 76% with two, three, and four voices, respectively. The baseline Sequential condition yielded accuracies of 86% (two voices), 89% (three voices), and 90% (four voices). It was interesting to note that accuracy increased with the number of voices in the Sequential condition. One possible explanation for this is that participants may have selected Voice 2, for example, if the second voice spoke their word, even if only Voices 1 and 4 were speaking, as was the case when only two

voices were present.

As in the previous study, Mauchly's sphericity test showed that both main effects met the assumption of sphericity ($W = 0.99, p = 0.65$ for temporal arrangement and $W = 0.99, p = 0.64$ for number of voices). Additionally, the interaction also met the assumption of sphericity, $W = 0.86, p = 0.11$, so once again we did not need to correct the degrees of freedom. A type III ANOVA for our 3 (temporal arrangement) $\times$ 3 (number of voices) design showed a significant difference between temporal arrangement, $F(2, 190) = 78.42$, $p < .001$, and between number of voices, $F(2, 190) = 37.60$, $p < .001$. Again, there was a significant interaction between our independent variables, $F(4, 380) = 30.13$, $p < .001$, showing that the number of voices had a different impact on accuracy for different temporal arrangements.

We ran the pairwise *post hoc* t-tests with Bonferroni correction to locate the significant differences. These results are presented in Table 5.2. Interestingly, mean accuracy increased slightly with number of voices in the SEQUENTIAL condition, though none of these differences were significant. In the DELAYLONG condition, participants' accuracy with two voices was significantly higher than with three or four, but there was no significant difference between three and four voices. In the DELAYSHORT condition, accuracy significantly decreased with each voice added. The most important difference from Study 1 was that three voices in the DELAYLONG condition was not statistically significant from two voices in SEQUENTIAL. While we cannot

		Condition								
		DelayShort			DelayLong			Sequential		
Condition	Voices	2	3	4	2	3	4	2	3	4
DelayShort	2	-	-	-	-	-	-	-	-	-
	3	**<.001**	-	-	-	-	-	-	-	-
	4	**<.001**	**.042**	-	-	-	-	-	-	-
DelayLong	2	1.00	**<.001**	**<.001**	-	-	-	-	-	-
	3	.228	**.002**	**<.001**	**<.001**	-	-	-	-	-
	4	**.004**	.063	**<.001**	**<.001**	1.00	-	-	-	-
Sequential	2	1.00	**<.001**	**<.001**	1.00	.063	**.004**	-	-	-
	3	1.00	**<.001**	**<.001**	1.00	**<.001**	**<.001**	1.00	-	-
	4	**.015**	**<.001**	**<.001**	1.00	**<.001**	**<.001**	.255	1.00	-

Table 5.2
The p-values from pairwise post hoc t-tests in the Study 2. Bold values indicate a significant difference ($p < 0.05$).

conclude from this that there was no difference between these conditions, the large number of participants in our study suggests that any difference in accuracy is likely to be small. It may be worth further investigating if the faster playback is worth any accuracy difference that may exist in a text entry task.

To measure the time saved by using simultaneous audio, we averaged the duration of each trial's audio file. This can be found, broken down by condition and number of voices, in Table 5.3. As expected, this was the lowest in DelayShort, with an average playing time of 0.95 seconds across all numbers of voices. Each trial had one fewer number of delays than it did voices (e.g. 2 delays for a trial with 3 voices), and the difference between the short and long delays was 0.1 seconds. From this, we would expect the DelayLong condition to have a 0.2 second higher average duration than DelayShort. We observed just that, with the average playing time of 1.15

Voices	DelayShort	DelayLong	Sequential
2	0.78	0.87	1.15
3	0.95	1.16	1.76
4	1.13	1.43	2.40
Avg	0.95	1.15	1.77

Table 5.3
The average playing time for the trials in each condition in seconds.

seconds for the DelayLong condition. The trials in the Sequential condition had an average playing time of 1.77 seconds, a 54% increase from the DelayLong.

5.4.3 Discussion

Adding the slight delay before starting to play each subsequent voice produced much more favorable results than the spatial and pitch distance alone, especially for greater numbers of voices. While we did not directly compare the DelayShort and DelayLong conditions to the 60Degree condition from Study 1, the mean accuracies were much higher: 60.2% and 76.0% with four voices in DelayShort and DelayLong, respectively, compared to 42.7% in 60Degree from Study 1. Most importantly, as noted in Section 5.4.2, DelayLong with three voices did not have a statistically significant difference in accuracy from Sequential with two voices. This shows that we were able to provide an additional suggested word in approximately the same amount of time (1.16 seconds compared to 1.15 seconds) without significantly decreasing performance.

5.5 Limitations and Future Work

While the results shown in these studies (particularly the latter) are promising, they are still only a simplification of a real-world text entry task. They allowed participants to focus all of their attention on listening to the suggestions for the target word. In a more complete task, users attention will be split between not only the word suggestions, but also the content that they are writing and the act of performing the input itself. An additional limitation is that in these studies, participants were instructed to wear headphones. We were not able to detect if participants followed this instruction. Additionally, in a real-world task, wearing headphones may not always be practical, which could impair users' ability to hear separate left and right channels. In the next chapter, we interview 12 blind adults about their experiences with nonvisual text input. During the interview, we play sample audio word suggestions from the DelayLong condition and solicit feedback on this method.

Chapter 6

Interview with Blind Users

When researching any Human-Computer Interaction (HCI) topic, it is imperative to consider the specific needs of the target users. When touchscreen smartphones first became commercially popularized, many studies were conducted that interviewed people who were BLV [5, 35, 52]. As technology changes, so too can the needs of the users. In this chapter, we aim to assess the current user needs in nonvisual text input to improve the FlexType interface and to provide recommended areas of future research to the field. We interview 12 legally blind adults about their past and present experiences with mobile text input and gather their thoughts on different interaction techniques, including FlexType from Chapter 4 and the simultaneous word suggestions from Chapter 5.

6.1 Related Work

Early work on accessible technology incorporated interviews and case studies to learn more about how people who were BLV interacted with technology [52]. The work done by Kane et al. [35] focused on formative interviews on their interactions with touchscreens. This led to the development of their Slide Rule selection mechanism, which allows users to explore a list of items by swiping their finger on screen and select an item by tapping a second finger while it is in focus.

Further work by Azenkot and Lee [5] sought to investigate how people who were blind used speech input on mobile devices compared to people who were sighted. They conducted a survey with 169 people (of which 64 were BLV) and found that people who were BLV were more satisfied with speech input and felt it was faster compared to people who were sighted. Ahmed et al. [3] interviewed 14 people with visual impairments to discuss how they protect their privacy while using their devices and what privacy concerns they weren't able to mitigate. They found that many participants were wary of other people overhearing the audio feedback from their device or being able to spy on the screen unnoticed.

Recent work by Karimi et al. [36] conducted interviews with 20 people who were BLV to explore how they text while traveling. They found that texting on-the-go often

required users to switch between applications to provide them with information about their surroundings. Abdolrahmani et al. [1] interviewed eight people who were BLV to learn more about how they use their devices in certain situations, such as when their hands are occupied or when they are on crowded public transportation. While these recent studies focused on nonvisual text entry in specific scenarios, we wanted to investigate users' experiences with their typical, everyday text entry methods. Similar to Kane [35], we also wanted to incorporate feedback from our interface's target users into the design process.

6.2 Procedure

We recruited a total of 12 legally blind adults for this study from the National Federation for the Blind mailing list in the United States. Participants were selected based on the order that they responded to the advertisement. The interviews took approximately 30–60 minutes, and participants received a US$20 Amazon gift card as compensation.

At the beginning of each interview, we obtained oral consent from the participant and then asked a series of demographic questions about their age, gender, and blindness. We asked each participant about their typical use cases for text input on mobile devices, and which languages they entered text in. We then asked about the main

technique that each participant used, and what they liked and didn't like about it. We followed this by asking about any other techniques that they use now or have used in the past. We asked the participants about common sources of error in their entered text, and their experiences detecting and correcting those errors.

To provoke discussion on various types of interfaces, we asked participants questions about their knowledge of Braille, as well as any experiences they may have had with Braille-based input on mobile devices. We then gathered participants' opinions on the FlexType interface we developed in Chapter 4, which we described as follows:

"FlexType would remove all dependence on the location of your taps. Essentially, it divides the letters into four groups and you tap with the number of fingers corresponding to the group containing your letter. For example, group 1 contains A through E, so for any of those letters you would tap with one finger. Group 2 contains F through M, so for any of those you'd tap with two fingers. You'd tap with 3 fingers for N through R, and with 4 for S through Z and apostrophe. Once you do all of the taps for a word, you swipe to the right and the interface uses the sequence of taps as well as the context of what you have already written to determine a list of the most likely matching words. You would then swipe up and down to navigate through this list until you hear the word you were trying to type."

We concluded the interview by discussing participants' experiences with word predictions, how their main interface currently presents them (if at all), at what rate they

listen to text-to-speech (TTS), and how frequently they use earbuds during text input. We played audio samples from Chapter 5, both from Study 1 without a delay and from Study 2 with a 0.25 s delay, to solicit feedback on simultaneous word suggestions and evaluate the potential of further research on that presentation method.

6.3 Results

Participants ranged from 38 to 66 (mean 50.25) years of age. Six identified as female, five as male, and one did not identify with either gender. None of the participants were currently studying at a university. Five of the participants were completely blind, with one more having only minimal light perception. Further details on the causes of their blindness and other participants' visual acuities can be found in Table 6.1. Eight participants reported being blind since birth, and all participants had been blind for a minimum of 19 years.

6.3.1 Types of Text

When asked what types of text they typically wrote on mobile devices, all 12 participants reported sending text messages, 8 reported composing emails, and 7 reported performing web searches. Three participants said that they interact with social media

Participant	Visual Acuity	Duration of Blindness	Cause of Blindness
P1	0	Legally 45 yr Completely 26 yr	Retinitis Pigmentosa
P2	20/3000 5° visual field	About 20 yr	Retinitis Pigmentosa
P3	0	Since birth	Microphthalmia
P4	0	Since birth	Retinopathy of Prematurity
P5	20/10000	19 yr	Leber Hereditary Optic Neuropathy
P6	0 in one eye 20/400 in one eye	Since birth	Cataracts
P7	Minimal light perception	Since birth	Leber Congenital Amaurosis
P8	0 in one eye 20/400 in one eye	Since birth	Retinopathy of Prematurity
P9	0	42 yr	Retinal detachment
P10	20/400	Since birth	Leber Congenital Amaurosis
P11	20/300	Since birth	Brain cyst, nystagmus
P12	0	Since birth	Dark corneas, cataracts

Table 6.1
Details of the visual acuity and duration and cause of blindness for each participant.

from their mobile devices, and another three fill out forms or surveys. One participant reported composing short stories or novels entirely on their mobile device. Five of the participants reported entering text in languages other than English, though four of them specified that it was rare.

6.3.2 Input Methods

Which participants reported using each input method can be found in Table 6.2. Dictation was the most common primary text input method, with 7 of the participants reporting it as their primary way of entering text and another 3 using it as a secondary method. Many cited its speed as the main benefit of inputting text by voice. The other commonly used method was the onscreen keyboard with VoiceOver, with four participants using it as their primary method, and six more as a secondary method. Of these ten participants, four set their keyboard to select the focused character when they released their finger, two selected a character by double tapping the screen while the character was focused, and one selected a character by tapping the screen with a different finger when the character was focused (split tapping). The other three did not report specifically how they selected characters.

6.4 Identified Themes

When reviewing the results of our interviews, we looked for common themes in the responses of our participants. We sought to identify experiences or opinions that many of our participants shared to help guide the direction of nonvisual text input research. We identified a total of eight themes. The most prevalent theme that we identified

Input Method	Primary	Secondary	Previously Used
Dictation	P1, P2, P3, P6, P7, P9, P11	P4, P5, P10	-
Onscreen keyboard w/ VoiceOver	P4, P5, P10, P12	P1, P3, P6, P7, P9, P11	-
Onscreen keyboard w/o VoiceOver	-	P8	-
Wireless physical keyboard	-	P5, P8, P9	P6, P12
Braille Screen Input	P8	P4	P12
MBraille	-	-	P4
FlickType (or similar method)	-	-	P10, P12

Table 6.2
The participants that reported using each input method.

was **the poor accuracy of dictation**. All ten participants that reported using dictation as either a primary or secondary text input method specifically mentioned that they had issues with the speech recognizer's ability to correctly determine what they said. This presented in a variety of ways to the different participants, some citing background noise, accents, uncommon words, or artifacts in their own speech such as coughs or hiccups as the reasons for its inaccuracies. Supporting quotes for each theme can be found in Table 6.3.

This led to the next theme that we identified: participants frequently mentioned having **difficulty entering text in noisy environments**. Users of both dictation and a soft keyboard with VoiceOver mentioned having these difficulties; dictation users mentioned that the speech recognizer often picked up on background noise, while soft keyboard users struggled to hear the audio feedback from their device. In

public places such as restaurants or on public transit, another common theme emerged that **participants were concerned about their privacy**. Participants were wary of other people hearing their dictated messages, or hearing the audio feedback from their devices. The participant that used Braille Screen Input primarily used screen away mode, but was frustrated that the text they were composing was displayed on the screen for anyone to see.

Theme	**Supporting Quotes**
Poor accuracy of dictation	"I used it to write a text message to my mom one time and the text came out all wrong. So I think that there's some issues with the microphone picking up the right words and things." (P11) "It doesn't understand all of what I'm saying, and it writes the wrong word, and then I have to go back and edit it several times to make it say the right thing." (P7)
Difficulty entering text in noisy environments	"When I'm in a noisy environment I have the the same issue that dictation does, which is hearing VoiceOver speak. So a lot of the characters kind of blend into each other when I'm trying to type something in a noisy environment." (P1) "If it's crowded, the dictation will hear other voices, and then it will get confused about what I'm saying." (P9)

Continued on next page

Table 6.3 – continued from previous page

Theme	Supporting Quotes
Participants were concerned about privacy	"I have one ear bud so I can always be listening to stuff on my phone and I can interact with it, and nobody else has to hear my VoiceOver that way, because it's a privacy thing." (P5) "One thing that drives me nuts about BSI is that there's no way to make it not show the words you're typing. I am a little bit creeped out that since I have my phone facing away from me, anybody can just read what I'm typing." (P8)
Difficulty correcting errors in entered text	"I haven't figured out an easy way to navigate the text and edit it." (P2) "It's pretty easy to detect an error when it reads it back to me. It's just it's harder to go back and correct it." (P3)
Wireless keyboard too cumbersome to carry	"It was much faster than typing it on the phone keyboard, but I just haven't used that recently. No particular reason, it's just another piece of equipment and I don't really like equipment." (P6) "The thing that I find frustrating with that is like, I usually don't have those devices with me and connected when I wanna send, you know, like a text or something." (P12)
Third party apps are no longer supported	"I really loved that app a lot, but then they changed it a bunch and it didn't work as well anymore." (P12)
Steep learning curve with new methods	"I feel like I would have trouble onboarding and it feels like for me, I think I could see myself getting really frustrated really fast." (P8) "I'm generally not a fan of a text entry method that involves me having to sort of relearn text entry. That's always the thing that sort of kills it for me is that I have to now have to enter this new thought process into my head." (P10)

Continued on next page

Table 6.3 – continued from previous page

Theme	Supporting Quotes
Word predictions are disruptive to use	"Those suggestions weren't just physically disrupting with typing, which they were, they were also really mentally disruptive." (P8)
	"I can tell you I've never used it, and when it's come up by mistake, I've hated it and had to redo it. You know it's it's just been a complete hindrance to me." (P6)

Table 6.3
Identified themes with supporting quotes from participants.

While most participants noted that it was typically fairly easy to detect when an error had occurred through the interface reading their text back to them, one of the other themes we identified was that participants had **difficulty correcting errors they detected**. A few were unsure of how to move the cursor back in the text field to get to the error, opting instead to delete their text and start over or to send a follow-up message to their recipient correcting the error. Some participants elaborated that when moving the cursor through the text field, it was difficult to know whether the cursor was at the beginning or end of the word spoken by the screen reader.

Of the five participants that talked about using a Bluetooth wireless keyboard, three mentioned that the reason they don't use it more frequently is that while it is quite fast, it is **too cumbersome to carry a physical keyboard with them**. The participants that discussed MBraille and FlickType mentioned that often times updates to their devices' operating systems would make **third party applications not work as well as they used to**, and it was difficult for developers to continue to offer long term support for their input methods.

Participants had mixed enthusiasm about the FlexType interface, but eight of the twelve were interested in trying it, and only two were firmly against the idea. One participant did not take a firm position either way, and the final participant was hesitant at first, but concluded their interview by stating “it sounds really promising." (P8). The chief concern that participants voiced was the **steep learning curve involved with learning the new method** and memorizing the groups of characters. Participants were also concerned about the accuracy of the algorithm in determining their intended word. One of the limitations of FlexType in Chapter 4 was that users were unable to enter words that didn’t appear in the top six predictions for that combination of group sequence and context. When asked how they would envision entering uncommon words or proper names that might not be in the system’s vocabulary, the most common responses participants gave were to allow users to define a custom dictionary and to have a letter-at-a-time mode that would remove the ambiguity from characters.

When we asked participants about their usage of word predictions, seven of them either did not use them at all or had the option disabled, one reported having used them very rarely, and the other four said they actively used them. All four participants that actively used their word suggestions said that they were located on the screen and they needed to explore them with their screen reader and then select them like they would a key. Participants that did not use them often said that the **word predictions were disruptive to use**, and that it was easier or faster for them to just finish typing the word.

Text-to-speech (TTS) speeds from our participants ranged from the default speed (denoted as 50% in VoiceOver) to 95% out of a maximum possible 100%. It is

Participant	TTS Speed	Earbud Usage
P1	Not reported	Hearing aids connected via Bluetooth
P2	Not reported	Not reported
P3	70%	When privacy is a concern
P4	70–75%	In the office, but not at home
P5	95%	80% of the time, single earbud
P6	Not reported	Not reported
P7	85%	75% of the time
P8	90%	10% of the time
P9	Not reported	None
P10	75%	90% of the time, single earbud
P11	Default (50%)	When around other people
P12	65%	Nearly all the time

Table 6.4
The text-to-speech speed and earbud usage reported by each participant. TTS speeds are reported on the scale used by iOS where 0% is the minimum, 100% is the maximum, and 50% is the default.

worth noting that only one participant reported using TTS at the default speed. The other seven participants that reported their TTS speed had at least some acceleration enabled. Participant use of earbuds while entering text was also mixed, with some saying they would use them only when around other people and others using them the majority of the time. Two participants mentioned only using a single earbud at a time so that they were still able to hear their surroundings, such as listening for their stop on public transit. Details on TTS speeds and earbud usage can be found in Table 6.4.

6.5 Directions for Future Research

6.5.1 Direction 1: Improvements to Dictation

Our interviews show that dictation is the preferred text input method of many people who are BLV, even in spite of the concerns they raised about its accuracy. This is due mainly to the speed at which users are able to enter text, even if they need to dictate their message multiple times to correct errors. While this work primarily focuses on the development of FlexType and improvements to dictation are out of the scope of this project, we recommend future work on the models behind the speech recognizer, specifically to make them more robust when the input contains background noise or accents. By reducing the error rate of dictated text, we can enable its use in more circumstances and improve the rate at which people who are BLV can input text on mobile devices.

Another area of dictation that could be improved is how it treats pauses in speech. P9 voiced their experience as, “If I slow down or stop, you know, trying to compose a sentence, it will try to send the message right away without me completing my thought or my sentence." It can be difficult to know the full message that you wish to send before you begin typing it, especially in the case of longer messages or emails. When typing on a keyboard it is easy to pause to think, but further work on dictation is needed to alleviate these difficulties for people who are BLV. A potential solution to this problem would be to allow users to customize the length of silence before the system registers the end of a message.

6.5.2 Direction 2: Less Reliance on Audio Feedback

While dictation has room to improve significantly, an excess of background noise or privacy concerns could still call for the use of an alternative text input method. Both of these scenarios could benefit from a text input method that has less reliance on audio feedback. FlexType moves in this direction by only requiring users to receive audio feedback after each word recognition as opposed to for each character. One participant suggested somehow using vibration to convey information to the user in addition to or in lieu of audio feedback. While there has been some work on this in the form of a Braille-based wearable glove for people who are deaf-blind [14], there is still an opportunity for further research in this area.

6.5.3 Direction 3: Improved Error Correction

Our next recommendation for the direction of future research is error correction. Errors are an inevitable part of text input, whether they originate from the system or from the user. Being able to efficiently correct errors is vital to achieving acceptable overall entry rates. Recent work by Zhang et al. [68] proposed three ways that sighted users could go about correcting errors without navigating the cursor to the error's location. While two of these involved dragging a correction to the location of the error and would likely not be well-suited to nonvisual text input, the *Magic Key* technique used a recurrent neural network to determine the most likely errors in a user's text. This method could be adapted to make nonvisual error correcting more efficient.

This research direction is supported by other work as well. A study by Rodrigues et al. [50] identified 'clunky editing' as a commonly discussed problem on community forums for people with visual impairments. A 12-week longitudinal field study by Nicolau et al. [44] showed that users spent 13% of their time correcting errors and corrected most of the errors that occurred in entry. The authors posited that increases to the efficiency of error correction would have a significant impact on overall text input rates.

6.5.4 Direction 4: Reduce the Barrier to Entry

Our final recommendation is not necessarily a standalone topic, but rather guidance for all other research. While only two of the twelve participants were not interested in even trying the FlexType interface, many were hesitant to fully adopt it because they were concerned about the barrier to entry created of learning a new technique. We highly recommend that future research take into consideration as much as possible ways that can reduce the barrier to entry for their input method. One possibility for this is to maintain some elements of consistency with current popular input methods, so users do not feel they need to enter an entirely new thought process when entering text, like P10 was concerned about. An example of this is the *SHARK* system developed by Zhai and Kristensson [67], which allowed users to draw gestures through the the letters on a keyboard to select them more quickly instead of tapping each individual letter. However, the *SHARK* system still allowed users to tap letters on the keyboard if they weren't comfortable with the gesture for a particular word yet. This research direction further supports our selection of the constrained character groups for our final user evaluation of FlexType in Chapter 8, since Section 4.3.3

showed a much higher initial backspace rate for users with the unconstrained groups.

6.6 Discussion and Limitations

The primary aim of this work was to gather information about the day-to-day use of current nonvisual text input methods and to assess the needs of users who are BLV. In the prior section, we recommended four directions for future research. Intentionally omitted from these directions was further work on word predictions. While four participants reported having them enabled on their interface, only one said that they used them frequently, with two using them about half the time and the fourth using them only when they were unsure about the spelling of a word. While further research on nonvisual word predictions could be beneficial to a small portion of the population, the results of our interviews do not suggest that it would have a widespread positive impact across users who are BLV.

Looking back on the surveys done by Azenkot et al. [5] on early speech input, we found it interesting that individuals who were BLV rated the accuracy of the method just under 4 on a 5-point scale. While it is difficult to make direct comparisons between separate participant pools, we found it noteworthy that a decade later, nearly all of our participants had concerns with the accuracy of speech input. One possible explanation for this level of satisfaction is that while speech input has improved since the original interview, the expectations of users have also grown. Even if that is the case, we feel that Direction 1: Improvements to Dictation has strong potential to have a large impact on the field of nonvisual text input.

One possible limitation to this work arises from how we recruited our participants. People who both registered for the National Federation of the Blind mailing list and were the first to respond to our email advertisement could have more-than-average experience with technology, and may not necessarily be representative of all adults who are blind. Additionally, all of our participants had been blind for a minimum of 19 years, which was before the 2007 release of the first iPhone that removed most physical buttons. People who have become blind more recently may have different experiences with nonvisual text input that stem from them using a touchscreen mobile device prior to losing their vision.

Chapter 7

Tuning Decoder Parameters

The disambiguation process of the VelociTap decoder [64] depends on different free parameters. When we ran our user study in Chapter 4, we did not have examples of users entering text using FlexType, so the decoder parameters we used were an estimate based on data from other interfaces. In this chapter, we use the data from Chapter 4 to test decoder models with different parameters to determine which would likely result in the best performance for the users.

7.1 Decoder Models

Each decoder model used a slightly different combination of parameters to control how it executed its search. The Baseline model treated all user input as fixed, and did not allow for insertions, deletions, or substitutions. That is, it restricted its search to only words that exactly matched the group sequence entered by the user.

The BASELINE model used the following parameters:

- **Character Scale Factor** — The scale factor for the log probabilities output by the character language model.
- **Word Scale Factor** — The scale factor for the log probabilities output by the word language model.
- **OOV Penalty** — The penalty assessed to the likelihood of hypotheses that are out-of-vocabulary (OOV) for the decoder's 100K vocab.
- **Beam** — One of the primary control parameters for pruning the search. The beam sets the maximum allowable log likelihood difference from the decoder's current best hypothesis before a hypothesis is pruned from the search. A lower beam value will result in a narrower search that is completed quicker, but potentially misses probable hypotheses.

Each other decoder model included all of the parameters used by the BASELINE model, with some additions.

The RNN model included an **RNNLM Scale Factor**, which which scaled the log probabilities output by a recurrent neural network language model (RNNLM). This RNNLM, trained by Adhikary et al. [2], contained 512 LSTM units, a character embedding size of 64, two hidden layers, a learning rate of 0.001, and a dropout probability of 0.5. It was trained on web text from Common Crawl[1] that was selected by the BERT language model [17] as likely to be from a set of crowdsourced messages

[1]https://commoncrawl.org/

similar to messages composed using alternative and augmentative communication (AAC) systems [60].

Our next model tuned penalties for inserting or deleting characters in the users' input. The INS/DEL model added three parameters to the baseline:

- **Insertion Penalty** — The penalty assessed to the log probability of a hypothesis for each character in the hypothesis that does not have a corresponding user input observation (group selection).
- **Deletion Penalty** — The penalty assessed to the log probability of a hypothesis for each user input observation that does not have a corresponding character in the hypothesis.
- **Apostrophe Insertion Penalty** — The penalty assessed to the log probability of a hypothesis for each apostrophe in the hypothesis that does not have a corresponding user input observation. We separated the parameter for apostrophe insertions since users may be more likely to omit an apostrophe than a letter while typing quickly (e.g. typing "dont" instead of "don't") [58].

Next we tested a variety of substitution models, without insertions or deletions, to isolate the effect of modeling substitution errors. The SIMPLESUB model had a single tunable **Substitution Penalty** parameter (in addition to the BASELINE parameters) for the penalty assessed to a hypothesis for each substitution, regardless of the specific character groups involved.

The CONFUSIONMATRIX model used the training data (derived from the CONSTRAINED condition in Chapter 4) to generate a confusion matrix between the groups

	Target Group				
Entered Group	Group 1	Group 2	Group 3	Group 4	Total
Group 1	14390	59	18	12	14479
Group 2	85	14655	36	6	14782
Group 3	9	40	12368	41	12458
Group 4	8	12	35	14388	14443

Table 7.1
The confusion matrix from users' group selections in the user study in Chapter 4. The target group was determined based on the transcription prompts given to users.

entered by users and the groups users should have entered given the target text. This matrix can be found in Table 7.1. The ConfusionMatrix model had the same tunable parameters as the Baseline model, but it assessed penalties to hypotheses based on the substitution probability estimates $P(substitution|entered)$ learned from the confusion matrix. For example, if a hypothesis substituted a character from group 2 where the user had entered group 1, the likelihood of the hypothesis was assessed a penalty of $P(s = 2|e = 1) = 59 \div 14479 = 4.1 \times 10^{-3}$. A different hypothesis that substituted a character from group 4 where the user had entered group 1 would be assessed a penalty of $P(s = 4|e = 1) = 12 \div 14479 = 8.3 \times 10^{-4}$.

In the confusion matrix, we observed that the number of entered taps decreased with distance from the target group. For example, for a target of Group 1, users entered Group 2 more than ten times as often as they entered Group 4. Since the groups were ordered alphabetically, this makes sense that users would be less likely to enter the group containing the end of the alphabet for a character towards the beginning of the alphabet.

Using this observation, we developed a more complex substitution model. The DECAYSUB model assessed a penalty to a hypothesis substituting a character from group s where the user entered group e to be:

$$P(s|e) = d^n, \tag{7.1}$$

where n was the number of groups between s and e, and d was a tunable **Decay Factor** in the range $(0, 1)$.

The last model we tested was the INS/DEL/DECAY model, which combined the parameters of the BASELINE model with the **Insertion Penalty**, **Deletion Penalty**, and **Apostrophe Insertion Penalty** of the INS/DEL model, and the **Decay Factor** of the DECAYSUB model.

7.2 Procedure

Using the data from the first FlexType study in Chapter 4, we first extracted the input traces for each task in session four and later from the eight participants in the CONSTRAINED condition, which had identical character groups to those we use in the FlexType interface in Chapter 8. With one of the goals of this process being to model user errors in group selection, we included traces for all words that were recognized, even if they were subsequently deleted. In those cases, we included each trace through entered words as its own training example. For example, if the user entered three words ("see you monday"), deleted one ("monday"), and then entered

two more (“tuesday morning"), we included both “see you monday morning" and “see you tuesday morning" as separate examples in the training data. To be able to align re-entered words with an adequate level of confidence, we only considered examples in which users’ final input had the same number of words as the prompt. Characters that were deleted prior to word recognition were not considered, since these were errors that were corrected by the users and were never seen by the decoder. This procedure left us with a total of 3792 training examples.

To choose the best decoder model, we evaluated each model option from Section 7.1 using leave-one-out cross validation. From our training data we created eight tuning folds, each withholding the examples from a single participant to be used as a test set. We used this method in order to evaluate how the models would perform on an unseen participant.

For each tuning fold of the training data, our tuning algorithm used the VelociTap [64] decoder to make predictions and compute an average character error rate (CER) for all the examples in the tuning fold. At each iteration, the tuning algorithm perturbed a single parameter both up and down by a random factor between 0 and 50% of its current value. The tuning algorithm compared the average CER with these parameter values, as well as the current value, and selected the value with the best CER. The algorithm perturbed each parameter in this manner, one at a time in a cyclic pattern. After each one-hour time period, the program completed an epoch of tuning and restarted with parameters perturbed from their initial values by up to 50% in either direction. The tuning algorithm terminated after three training epochs and selected the parameter values with the best CER on the tuning fold across all three epochs. This procedure produced eight sets of parameter values for each decoder model, one

for each tuning fold.

For each set of parameter values selected by the tuning algorithm, we then computed the character error rate (CER) and word error rate (WER) for each example in the tuning fold's corresponding test set (i.e. the examples from the participant held out from the tuning fold). For each decoder model, we combined the resulting error rates for each example in the training data (since each was present in exactly one test set). We ensured that the examples were combined in the same order for each model so that we could compare the performance of different decoder models using paired examples.

7.3 Results

For each model, we calculated the mean of all the samples in the test results. We then drew 100,000 samples from the test results, with replacement, to generate a bootstrapped estimate of the 95% confidence interval (CI). The mean error rates and bootstrapped CIs for each model can be found in Table 7.2.

The Baseline model yielded an 8.74% character error rate (CER) and a 16.90% word error rate (WER). Since the RNNLM component of the Rnn model was computationally expensive and slowed the evaluation of each hypothesis, we doubled the length of each tuning epoch to two hours for the Rnn decoder model. On average, the Rnn model resulted in a CER of 10.13% and a WER of 19.21%.

Model	% CER	% WER
Baseline	8.74 ± 0.44	16.90 ± 0.73
Rnn	10.13 ± 0.47	19.21 ± 0.79
Ins/Del	8.71 ± 0.43	16.82 ± 0.74
SimpleSub	8.71 ± 0.43	16.87 ± 0.73
ConfusionMatrix	8.75 ± 0.43	16.89 ± 0.74
DecaySub	8.33 ± 0.43	16.10 ± 0.72
Ins/Del/Decay	8.55 ± 0.43	16.48 ± 0.73

Table 7.2
The mean error rates and bootstrapped confidence intervals for each decoder model. Results are reported in the format *mean* $\pm$ 95% *CI*.

Model	Δ %CER CI	Δ %WER CI
Rnn	[1.197, 1.578]*	[1.971, 2.660]*
Ins/Del	[−0.086, 0.019]	[−0.134, −0.011]*
SimpleSub	[−0.085, 0.024]	[−0.071, 0.027]
ConfusionMatrix	[−0.008, 0.028]	[−0.021, 0.003]
DecaySub	[−0.573, −0.260]*	[−1.025, −0.569]*
Ins/Del/Decay	[−0.304, −0.085]*	[−0.573, −0.254]*

Table 7.3
The bootstrapped 95% confidence intervals for the pairwise differences between each model and Baseline. Significant results are indicated with an asterisk (*). Positive differences indicate an increase in error rate compared to Baseline.

To compare each model to the Baseline, we bootstrapped estimates of the 95% confidence interval for the difference between the models. We sampled the same 100,000 tasks from the test results of each model, with replacement, and used the pairwise differences in our calculations. The complete results can be found in Table 7.3.

Compared to the Baseline, the Rnn decoder model showed an average increase in CER by 1.39% $\pm$ 0.23% and in WER by 2.32% $\pm$ 0.34%. Since the entirety of the bootstrapped 95% confidence intervals fell on the positive side of zero, we concluded

that the RNN model performed significantly worse than the BASELINE and chose not to include RNNLM re-scoring in our other models. One reason this model may have performed worse is that the RNNLM was only trained on 100M characters of text since the training process was computationally expensive. For comparison, the character and word language models used were trained on over 20B characters and 8B words, respectively.

The INS/DEL decoder model produced a mean CER of 8.71% and a mean WER of 16.82%. When compared to the BASELINE model, the bootstrapped CI did not show a significant difference in the character error rate. The mean pairwise difference was -0.034% $\pm$ 0.053%. However, the bootstrapped CI indicated a significant improvement in word error rate, with a mean pairwise difference of -0.072% $\pm$ 0.062%.

The SIMPLESUB model was able to achieve means of 8.71% CER and 16.87% WER, both of which were slight improvements over the BASELINE. As shown in Table 7.3, however, we found no significant difference between the BASELINE and SIMPLESUB models. The mean pairwise differences were -0.030% $\pm$ 0.055% CER and -0.022% $\pm$ 0.049% WER.

The CONFUSIONMATRIX model had very similar performance to BASELINE model, with an average CER of 8.75% and WER of 16.89%. Again, we found no significant difference between these models, with a mean pairwise CER difference of 0.010% $\pm$ 0.018 and a mean pairwise WER difference of -0.009% $\pm$ 0.012%.

The DECAYSUB model showed a significant improvement over the BASELINE. The average CER for this model was 8.33%, and the average WER was 16.10%. In comparison to the BASELINE model, the DECAYSUB model had a mean pairwise CER

difference of -0.42% ± 0.16%. The mean pairwise WER difference was -0.80% ± 0.23%. Both of these differences were significant, since the entire bootstrapped 95% confidence intervals fell below zero.

Since we saw significant improvement over the BASELINE for WER with the INS/DEL model and for both CER and WER with the DECAYSUB model, we tuned the combination of their parameters in the INS/DEL/DECAY model. This model had an average CER of 8.55% and average WER of 16.48%. Both of these error rates also showed significant improvement over the BASELINE model. However, a pairwise comparison of INS/DEL/DECAY to the DECAYSUB model (without the insertion and deletion parameters) yielded bootstrapped 95% confidence intervals of $[0.097\%, 0.348\%]$ for the CER and $[0.205\%, 0.563\%]$ for the WER, both showing significantly higher error rates for INS/DEL/DECAY. A pairwise comparison of INS/DEL to DECAYSUB also showed that INS/DEL had significantly higher error rates (bootstrapped CIs of $[0.225, 0.541]$ and $[0.490, 0.961]$ for CER and WER, respectively), so we selected DECAYSUB as our final model.

Having selected the DECAYSUB decoder model, we reran our tuning procedure with all of the training data, not withholding any participants, to select the final parameter values for that model. These values were:

- **Character Scale Factor** — 0.4301
- **Word Scale Factor** — 0.7230
- **OOV Penalty** — 2.4630×10^{-7}
- **Beam** — 3.0587

- **Decay Factor** — 0.0685

In Chapter 8, we incorporate these decoder parameters into the FlexType interface for use in a longitudinal user study. In the user study's training sessions, we wanted to avoid potentially unexpected behavior (e.g. the decoder performing a substitution during recognition) while the participants were learning the technique and the character groups. To address this, we also performed a final tuning with all of the training data for the BASELINE model. The final parameter values were:

- **Character Scale Factor** — 0.3097
- **Word Scale Factor** — 0.8363
- **OOV Penalty** — 1.6505×10^{-7}
- **Beam** — 1.1118

Chapter 8

Final User Evaluation

In this chapter we incorporate feedback from users in the original FlexType study in Chapter 4 and participants in the interview study in Chapter 6 into our interface. We then deploy the updated FlexType to blind users' smartphone devices to evaluate if participants are able to enter text more quickly and/or accurately using FlexType than using their typical onscreen text entry method.

8.1 System Description

In this study, participants completed each session remotely on their own device. Since the vast majority of our interview participants in Chapter 6 used an iPhone as their primary mobile device, we ported the Android version of FlexType used in Chapter 4 to iOS (the iPhone operating system). All participants used a device running at least iOS 16.0. The primary functions of the interface remained the same, using one to

four-fingered taps to select from the four character groups. In Chapter 4, we did not find a significant difference in entry or error rate between the CONSTRAINED and UNCONSTRAINED groups. Because our interview participants suggested lowering the barrier to entry for novel text input methods, we selected the CONSTRAINED groups for the updated FlexType interface:

$$(a, b, c, d, e), (f, g, h, i, j, k, l, m), (n, o, p, q, r), (s, t, u, v, w, x, y, z, ')$$

After entering the full sequence of taps for each word, participants swiped right to use the VelociTap decoder [64] to disambiguate the sequence. The decoder ran on a remote server, and was queried for recognition results through a REST API. The decoder operated the same as described in previous chapters, with a 4-gram word model and 12-gram character model, but used the parameters tuned in Chapter 7. To avoid potentially unexpected behavior (e.g. the decoder performing a substitution during recognition) while the participants were learning the technique and the groups, the first three sessions used the BASELINE model parameters from Chapter 7, while the remainder used the DECAYSUB model parameters. The method of word selection following disambiguation remained the same as used in Chapter 4, and participants were still able to swipe left with one finger to backspace a character and with two fingers to backspace a word. The version of the interface used in this chapter also added a three-finger left swipe gesture to clear the entire text field.

By default, the interface began each entry task in *word entry mode*, which is the mode that we have described thus far. With a two-fingered swipe up, participants were able to toggle to *letter entry mode*, the other new interface feature introduced in this chapter's version. This mode allowed participants to enter text a single letter at

a time to solve the issue faced by some participants in Chapter 4 where their target word was not in the n-best list produced by the decoder. In letter entry mode, each group selection would enter (and speak) the character in the middle of the selected group. Participants could then swipe up or down to iterate forwards or backwards through the characters in the group, the same way they could to explore the n-best list in word entry mode. A two-fingered swipe up while in letter entry mode would toggle back to word entry mode.

We opted to implement letter entry mode as opposed to increasing the size of the n-best list produced by the decoder. Our main reasoning was that increasing the size of the n-best list would still not be guaranteed to solve the problem, and it would lead to participants spending more time exploring deeper into the list. The results we found in Section 4.3.5 showed that each position deeper into the n-best list contained a smaller portion of the words entered, with that portion roughly halving between the fourth and fifth positions, and again between the fifth and the sixth. Even if we assume that subsequent positions in the n-best list would maintain the same portion of words seen in the sixth position (which is unlikely given the trend of the data), we would need to double the size of the n-best list to include all of the words seen by our users in Chapter 4. When we considered additional out-of-vocabulary words such as proper nouns specific to individual users' lives that aren't in our phrase lists, the catch-all method of letter entry mode was the clear choice.

A long press (over 600 ms) with one finger would cause the interface to read the prompt again as well as what the user had entered so far. A long press with two fingers would cause the interface to speak the available gestures.

8.2 Procedure

Prior to the study, we tested and evaluated the FlexType app by employing a third party accessibility expert in the blind/low-vision field. We used this expert's feedback to refine the length of each session and to make the app more accessible for users with visual impairments.

The study app was made available to participants through Apple's TestFlight, a system designed for developers to test preliminary versions of their applications. This allowed us to restrict access to only the participants that were recruited for the study.

Upon downloading the app, participants were presented with an informed consent form, followed by an introductory questionnaire. At the beginning of each session, participants were shown an instructions page that included a description of what was new in that session, followed by a reminder of the available gestures introduced in previous sessions. Participants were instructed to hold the device in portrait orientation in one hand, using the other hand to tap the screen.

As with the studies we conducted in Chapters 2 and 4, each participant completed eight sessions designed to progressively get harder and introduce interface features. The first four sessions were designed as practice, while we used the final four to compare the FlexType interface to each participant's typical text entry method.

- Session 1 — Participants received single-character prompts and were instructed to tap with the number of fingers corresponding to that character's group. After

each tap, the characters in the selected group were read, followed by feedback on whether the selection was correct. If incorrect, participants were instructed to try again until it was correct. If correct, participants were informed as such and the system automatically advanced to the next prompt. A long press in this session would cause the system to read the target character as well as its word from the NATO phonetic alphabet to help distinguish between similar sounding characters. This component was suggested by our accessibility expert. Participants entered each character (A–Z and apostrophe) four times, in a randomized order.

- Session 2 — Participants received 54 single-word prompts from a list of common words. These prompts were chosen to ensure that each participant encountered each character at least three times. The prompts were pruned to remove any containing words that would require participants to explore the n-best list. Due to a bug, one word ("they'd") did require use of the n-best list. This prompt was excluded from analysis for all participants.

- Session 3 — Participants received 20 phrase prompts that were pruned to contain no more than four words, each no longer than six characters. The prompts were not pruned to remove words not appearing as the first result in the n-best list, but they were pruned to remove words that did not appear at all in the n-best list. The first prompt each participant received contained a word that would require participants to explore the n-best list. All phrase prompts were drawn from the Enron mobile data set [61].

- Session 4 — Participants received 20 phrase prompts that were pruned to contain no more than four words, each no longer than six characters. The prompts were no longer pruned to remove words not appearing in the n-best list, and the

first prompt each participant received contained a word that would not appear in the n-best list. This allowed participants to practice using letter entry mode immediately after it was described.

- Sessions 5–8 — Participants received 20 phrase prompts in each of two conditions, FlexType and Baseline. The FlexType condition used the same entry process as the previous sessions, and the Baseline condition instructed participants to use their typical onscreen text input method, and to use the same method in each session. The order in which participants completed the conditions alternated between sessions and between participants (e.g. Participant 1 completed the FlexType condition first in sessions 5 and 7 and last in sessions 6 and 8, while Participant 2 completed the FlexType condition last in sessions 5 and 7, and first in sessions 6 and 8). Prompts were only pruned to contain no more than six words to aid participants in remembering them correctly.

Based on feedback from participants in Chapter 4, the corresponding character group was only read after each tap in the first two sessions. In all subsequent sessions, a key tap sound was played instead. In all sessions after the first, participants swiped down with two fingers to indicate when they were finished entering a prompt. They were then presented with their accuracy and speed for that prompt and were able to tap to continue to the next prompt. Participants completed a questionnaire at the end of each of the first four sessions, and after each condition in the final four sessions.

In each session using FlexType, the app asked participants to disable VoiceOver (if the system detected it was enabled) prior to receiving the first prompt. This was a necessary step since VoiceOver, if enabled, would intercept screen touch events and

prevent FlexType from interpreting them. We designed the app to provide its own audio feedback in instances where VoiceOver needed to be disabled. We discuss some of the drawbacks of this design consideration in Section 8.4.

Participants were instructed to complete each session with only one session in a single day and at most one day off between two sessions. If participants allowed the FlexType app to send notifications, they received one 24 and 48 hours after completing a session if they had not yet begun the next session. They also received one if 15 minutes passed mid-session without them completing a task.

8.3 Results

We recruited a total of 25 participants from the National Federation for the Blind email list. Of these participants, four never downloaded the app for the study and eight more did not finish all eight sessions. While at the time of writing, four of the incomplete participants appeared to have dropped out of the study (no sessions in over two weeks), the other four still seemed to be completing sessions slower than the instructed pacing.

From the 13 complete participants, one participant displayed signs of having significant difficulty with the word-at-a-time entry technique utilized by FlexType. The majority of this participant's input in the training sessions consisted of disambiguating a single character at a time instead of a full word, leaving spaces between each character. Because this was a remote and asynchronous study, we were unable to help clarify how this technique was intended to work. This participant used exclusively

letter entry mode for all input in the FlexType conditions of sessions 5–8, so we excluded them from the bulk of analysis. However, since their results could provide interesting insight into the performance of FlexType in letter entry mode, we will refer to this participant as "Participant X", and discuss their results specifically in Section 8.3.4.

The 12 remaining participants were ages 21 to 62 (mean 44). Three identified as male, and nine as female. Seven participants reported being blind since birth, and all participants had been legally blind for more than five years. All participants rated the statement "I am a fluent speaker of English" a 7 on 7-point Likert scale, where 7 represented "strongly agree".

8.3.1 Entry Rate

We measured participants' entry rate in words per minute (WPM), considering every five characters to be a word. The results, shown in Figure 8.1, seemed to depend on the individual participants. Some participants showed great improvement, with Participant 1 improving from 5.4 WPM in session 2 to 12.8 WPM in session 8, and Participant 5 improving from 10.1 WPM to 17.0 WPM. Other participants did not see the same improvement, with Participant 10 regressing from 2.6 WPM to 2.3 WPM over the course of their sessions. Overall, participants averaged 8.2 WPM over the course of their final four sessions. This had quite a bit of variability, with a range from 2.2 WPM to 14.2 WPM and a standard deviation of 2.9 WPM. We found the participants at both ends of the range to be outliers, more than 1.5 times the interquartile range from their respective nearest quartiles.

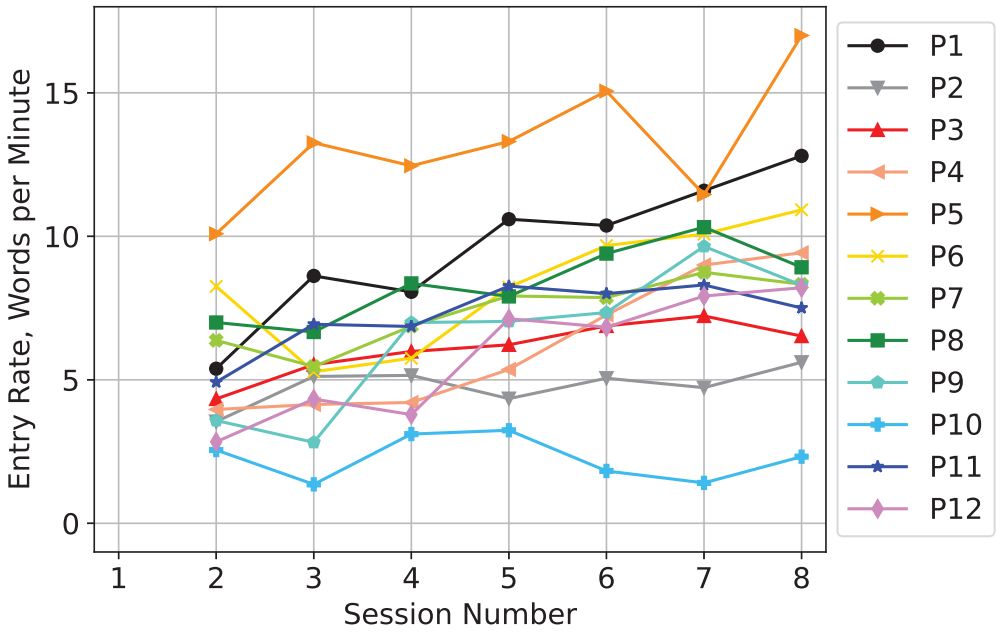


Figure 8.1: Entry rates of each participant on FlexType throughout the study. Session 1 entry rates are not reported since participants only entered single-character prompts.

When comparing participants' results with their typical text entry methods, we chose to look at each baseline method independently. Of our 12 participants, 8 used a Qwerty keyboard with VoiceOver in their Baseline condition. These eight participants had a mean FlexType entry rate of 7.0 ± 2.4 WPM, and a mean Baseline entry rate of 7.5 ± 3.0 WPM. Since a Shapiro-Wilk test showed no violations of the normality assumption ($W = 0.99$, $p = 0.99$), we used a dependent t-test to check significance. This test did not show that this difference was significant ($t(7) = -0.42$, $p = 0.69$, negligible effect size: Cohen's $d = 0.16$).

Three participants (P1,P5,P9) used Braille Screen Input as their typical text input method. These participants had a mean FlexType entry rate of 11.2 ± 3.1 WPM, and a mean Baseline entry rate of 26.9 ± 8.7 WPM. Again, a Shapiro-Wilk test

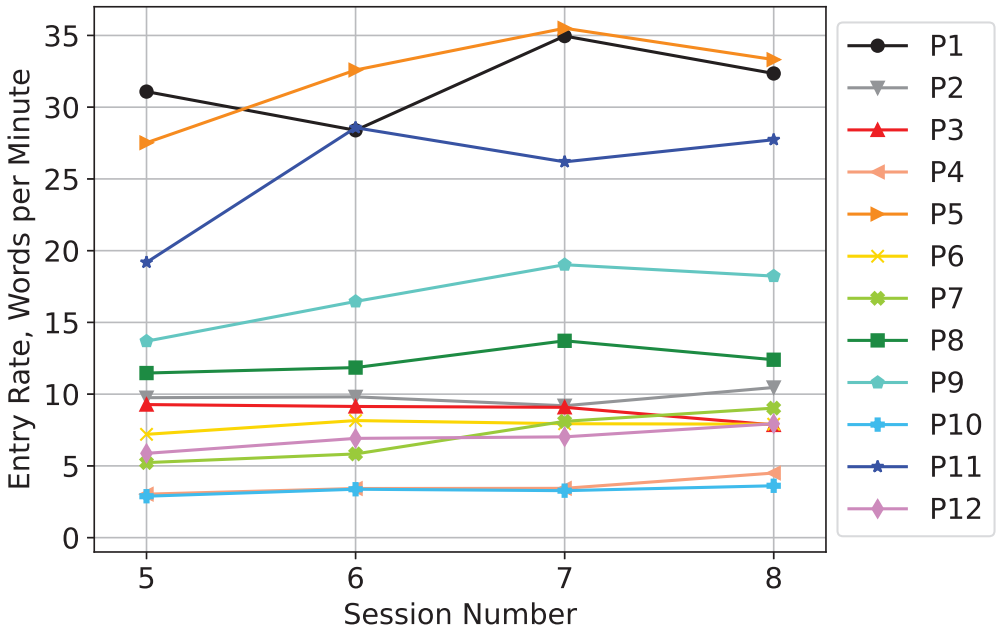


Figure 8.2: Entry rates of each participant with their Baseline text input method. Participants only used this method in sessions 5 and later.

showed no violations of the normality assumption ($W = 0.89$, $p = 0.36$), but the dependent t-test did show that the BASELINE was significantly faster ($t(2) = -4.44$, $p = 0.047$, large effect size: Cohen's $d = 1.16$).

Although participants were instructed to use their typical onscreen keyboard, the final participant (P11) used a Bluetooth keyboard in their BASELINE condition. While we were unable to conduct statistical tests with a sample size of only one, this participant had a mean FLEXTYPE entry rate of 8.0 WPM and a mean BASELINE entry rate of 25.4 WPM.

Each participant's BASELINE entry rates for each session can be seen in Figure 8.2. While most participants were fairly static throughout the sessions, some (P5,P9,P11) seemed to get slightly faster in later sessions. We hypothesize that this was due to

the participants gaining familiarity with the study app and the process of submitting their text, as opposed to improvement in their use of their typical text entry method.

8.3.2 Error Rate

To evaluate our participants' error rates in session 1, we used the error rate of their taps, since they were only being asked to select the group containing a character, and they were not entering text. For example, if a participant required three attempts to select the correct group, this would yield a 67% error rate for that prompt.

For the remainder of the sessions, we used Character Error Rate (CER). We computed the number of insertions, deletions, and substitutions required to transform the participant's final text (after any corrections) to the reference text (the prompt), divided by the length of the reference text. Since it was possible for the length of the input to exceed the length of the prompt, it was possible to obtain a CER that exceeded 100%.

As shown in Figure 8.3, we did observe a CER over 100% for Participant 6, who had a misunderstanding about how the interface worked in session 2, which led to an average CER of nearly 386%. The participant was tapping each group multiple times to get to the desired character (e.g. tapping with two fingers five times to get to 'J', the fifth letter in group two). This led to excessively long tap sequences that the decoder then failed to disambiguate since there were no words matching those sequences. The participant reached out to us following that session and we were able to clarify that the disambiguation process would choose the letters from the groups

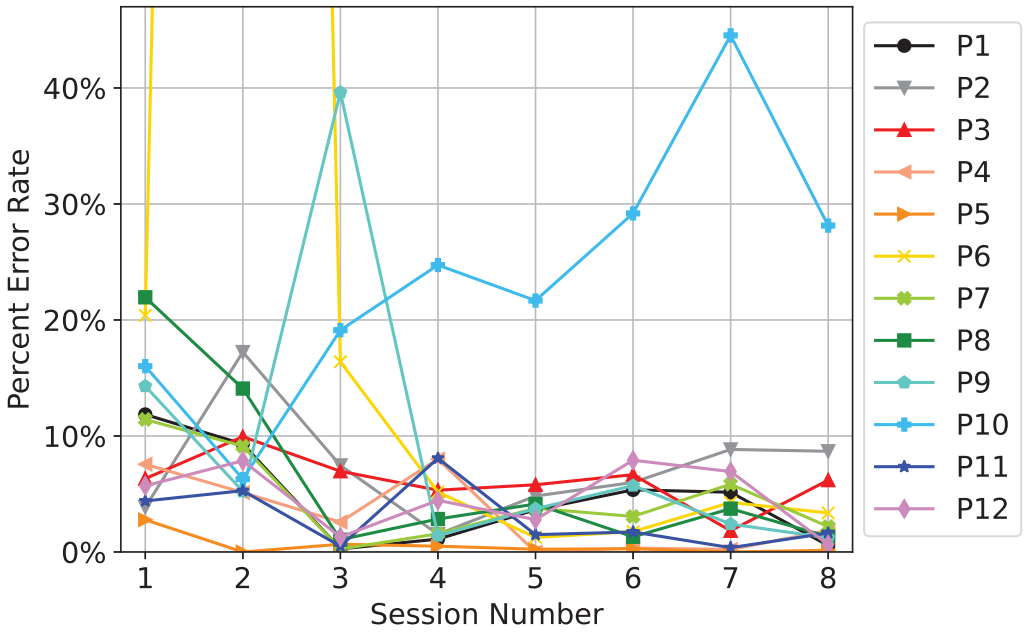


Figure 8.3: The error rate of each participant in the FlexType condition throughout the user study. Session 1 results depict tap error rate, and subsequent sessions depict character error rate. In session 2, P6 had a CER of 386%.

after the word was completed. Participant 6 went on to average 2.7% CER with FlexType over the final four sessions.

Overall, participants had slightly higher error rates in the first couple sessions that then dropped down as they became more familiar with the text entry technique. Across the final four sessions, participants had an average FlexType CER of 5.5%, though this was inflated by Participant 10, who had an average CER of 30.9% in their final four sessions. We will discuss more about why this may have been in Section 8.4. More resistant to this outlier, the median CER of participants in their final four sessions was 3.5%.

The differences in character error rate between the FlexType and Baseline conditions for the eight participants that used a Qwerty keyboard with VoiceOver violated the normality assumption ($W = 0.73$, $p = 0.004$). While these participants had a mean FlexType CER of 7.2% ± 9.8%, and a mean Baseline CER of 2.9% ± 2.0%, a Wilcoxon signed-rank test did not find this difference to be significant ($V = 28$, $p = 0.20$, moderate effect size: $r = 0.50$).

For the three Braille Screen Input users, we found a mean FlexType CER of 2.4% ± 1.9%, and a mean Baseline CER of 0.5% ± 0.1%. The differences did not violate normality ($W = 0.81$, $p = 0.15$), and a dependent t-test did not show a significant difference ($t(2) = 1.77$, $p = 0.22$, small effect size: Cohen's $d = 0.39$). The Bluetooth keyboard user had a mean FlexType error rate of 1.3% CER and a mean Baseline error rate of 0.2% CER. Once again, we were unable to conduct significance tests due to the sample size of one.

8.3.3 Backspaces per Character

For each session, we also calculated the number of characters each participant backspaced per character in their final text. For this metric, we used the total characters deleted, summed across all methods of backspacing (single character, full word, full input). As shown in Figure 8.4, most participants had a fairly static number of backspaces per character (BPC) throughout their sessions. The exception to this was, again, P10, who backspaced quite frequently in the later sessions, and over 6 times per final character in session 7. For the final four sessions, participants had a mean BPC of 0.41 ± 0.71, with a median of 0.21 BPC.

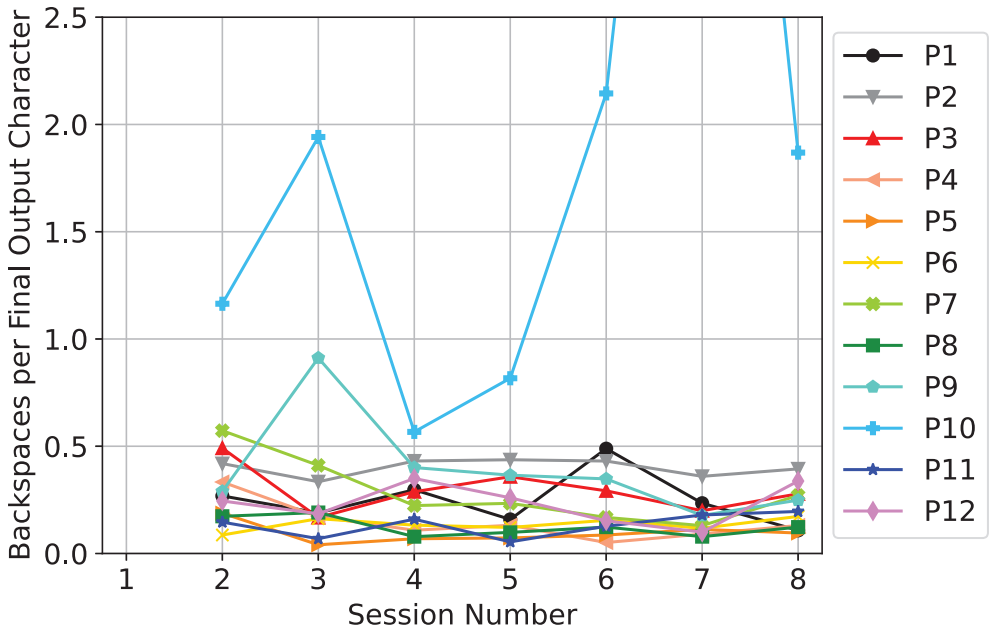


Figure 8.4: Each participant's backspaces per final output character in the FlexType condition. This is not reported for session 1 since participants were not able to backspace. In session 7, P10 had a BPC of 6.1.

In Chapter 4, we saw a relatively high BPC in session 1, followed by a fairly constant and low BPC in the remaining sessions. In this study, we adjusted session 1 to have participants try again if they selected the incorrect group without the need (or ability) to backspace. Because of this, we don't have a measure of participants' BPC in session 1. However, if we assume that participants backspaced each incorrect tap in session 1, this would yield a theoretical mean BPC of 0.32 ± 0.29 for that session, with a median of 0.20 BPC. Interestingly, this was actually lower than for the final four sessions. The medians were quite similar, however, suggesting that the mean for the final sessions was inflated by the outlier.

In the Baseline condition, since participants used their typical text input methods and not something we had developed, we were not able to log input events and were

only able to access the final text for each prompt. Because of this, we weren't able to measure any sort of corrective action, such as backspaces per character, for the Baseline condition with which to compare our FlexType results.

8.3.4 Participant X

As we mentioned previously, one participant (Participant X) used exclusively letter entry mode for their final four sessions. While they were excluded from the analysis we have conducted so far, we will discuss their results here. It is possible that letter entry mode could have an additional application in situations where users are not able to receive audio feedback. While that was not the case for this participant, the deterministic nature of this mode could, with enough practice and familiarity with the groups, make it so that the audio is not needed.

In the final four sessions, Participant X entered text using FlexType at 3.0 WPM, which was only slightly behind their Baseline method (Qwerty keyboard with VoiceOver) at 3.4 WPM. While Participant X was not the slowest with entry on FlexType, they were second only to Participant 10, who reported consistent difficulty using the app.

Participant X's error rate in the FlexType condition at 5.3% CER was also slightly higher than their error rate in the Baseline condition (4.1% CER). However, their FlexType error rate fell just under the average observed from the rest of the participants. This error rate was inflated by session 5, where Participant X had a 15.2% CER, before averaging 2.0% for the final three sessions. This high error rate seemed

to be caused by extra letters inserted within several entries (e.g. “thaknjks goocd job” for the prompt “thanks good job”) and one prompt that Participant X skipped entirely. Participant X averaged 0.35 BPC in their final four sessions, which again, fell just below the mean of the other participants.

Subjective feedback from Participant X shifted throughout the duration of the study. After session 3, Participant X remarked, “This is tedious program, which I would never ever use." The introduction of letter entry mode seemed to shift this view, and after the FlexType condition in session 6, PX commented, “This part was easy peazy." Participant X’s final comment after the conclusion of the study was, “I actually grew to like FlexType more than the Carti keyboard that I used to normally enter text. The letter mode really was cool."

8.4 Interface Limitations

As we saw in Section 8.3, Participant 10 had the lowest entry rate and both error rates and BPC that were quite elevated in the FlexType condition. In post-session comments, P10 noted a variety of factors that likely contributed to these poor outcomes. P10 specifically stated that they had trouble getting their fourth finger down to select group 4, and that once the feedback on which group had been selected went away, it was difficult for them to know if it had been recognized properly. In this study, we removed the audio feedback that read the selected group after each tap in session 3 and later. This decision was based on the feedback we received from participants in Chapter 4, who stated that they found it quite annoying in later sessions when they were familiar with the groups and typing faster. P10’s experiences in this study

suggest that the group feedback should be a configurable setting for users so they can enable it if it is useful to them. A possible compromise option would be to read the group number after each tap, as suggested by Participant 1. While some participants in Chapter 4 noted some difficulty tapping with four fingers (e.g. P13 in the previous study said, "tapping with all 4 fingers is difficult when tapping fast"), none expressed so much difficulty that we felt necessitated change. Seeing P10's consistent comments that they struggled to select group four, it is likely worth considering a reduction to three groups or an alternate way to select group four.

Based on the results of our interview study in Chapter 6, we chose not to include word suggestions prior to the completion of each word. However, after their BASELINE condition in session 5, P8 commented that "some keyboards have auto correct, and some keyboards have suggested words which could lead to shortcuts". If some participants were using these features with their typical text entry method, that could have lead to increased performance in the BASELINE condition. While we asked participants to report the text entry method they used for the BASELINE condition, we did not ask about their use of word suggestions or autocorrect. In future development of FlexType, it could be beneficial to implement word suggestions. One possible implementation would be to play the most likely prediction after each tap and allow users to swipe right to select it. This would allow users to know what result they will get when they swipe right and potentially increase their entry rate by eliminating keystrokes. However, this could confuse users if their intended word has a similar starting group sequence to the words being read, without necessarily having similar letters (e.g. "help" and "fair" have the same tap sequence without sharing any letters).

Another limitation of our interface was that FlexType was not integrated with

VoiceOver. Since VoiceOver, if enabled, intercepts screen touch events and processes them itself, participants needed to disable VoiceOver in the FlexType condition. While we developed the app to provide its own audio feedback, both during entry and to read each prompt, Participant 3 noted after a Baseline condition that "Being able to use VoiceOver enabled me to actually read the words I was supposed to be typing individually and go character by character, so I was able to know how certain things were spelled". This was not available in the FlexType condition, and may have contributed to higher error rates when compared to participants' Baseline methods.

8.5 Study Limitations

The largest limitations with this study arose from it being conducted entirely remotely. Since participants were using their own devices for the study, they had a variety of devices and operating system versions. This led to some bugs that we did not encounter during testing. For example, P10 noted that whenever they tried to switch to letter entry mode, the app would crash. P6 also noted crashes that occurred when they tapped with three fingers, since it conflicted with gestures used by their screen magnification software. The remote study also made it more difficult to clarify any misunderstandings of the instructions. While some participants reached out via email to ask questions, others did not, which led to poorer performance. An in-person study with the ability to standardize the device would have allowed participants a smoother experience with the interface, but it would have been more difficult to recruit a large and diverse participant pool.

Another difficulty of the remote study was enforcing our desired session pacing, since the study was conducted asynchronously. While seven participants followed our pacing instructions, three participants had a large time gap within a single session. Participant 9 had an overnight 14-hour gap part way through session 4, and Participant 12 had a one-day gap a few prompts into session 2. Participant 6 encountered a bug within the final session, and it took a few extra days for us to find a fix, but they were otherwise compliant. One participant (P2) had a 4-day gap between their first and second session, and the remaining participant (P8) completed all four of their final sessions within a 14-hour total time span. While most of these occurrences were minor and we don't have reason to believe they impacted the final results, P8's results may have been affected by the fatigue of doing the sessions in quick succession.

The other limitation with the study was the number of participants that completed the final session. While we recruited 25 participants, we only had 12 for our analysis. This led to limited power for our statistical tests, especially for the users of Braille Screen Input, where we only had a sample size of three. While we did recruit more participants that used BSI, not all of them finished the study. This also created a slight imbalance in the ordering of conditions. Since all three participants that used BSI had odd participant numbers they completed the FlexType condition first in session 5. Since we swapped the order of conditions in alternating sessions (odd numbered participants completed the Baseline condition first in sessions 6 and 8), we believe the effect of this order imbalance on the final data to be minimal.

8.6 Discussion

While we were hopeful that we would see FlexType outperform users' typical text input methods, this was not the case. When comparing our participants' performance on FlexType to their baseline text entry methods, the only significant difference we found was that Braille Screen Input users were faster with their typical method than they were with FlexType. This was supported by participant comments, with P9 remarking, "after using my normal text input method, I found this method to be especially slow", after the FlexType condition in session 6. However, compared to their typical text input methods, participants had very little practice with FlexType. After the final session, P9 was hopeful for the future of FlexType, stating, "I believe right now that FlexType is slower than my regular text input method, but, I am sure with improvements it can become faster than braille screen input." Conversely, Participant 1 provided the feedback that they don't think FlexType will ever be faster than BSI for them, but that it would be great for people who don't know Braille.

While we found no significant difference between a Qwerty keyboard with VoiceOver and FlexType, the majority of subjective user feedback seemed to favor FlexType. Of the eight participants that used a Qwerty keyboard with VoiceOver as their Baseline text input method, five expressed that they wanted to learn more about the development of FlexType in the future. In their final questionnaire, Participant 8 left the comment, "I really hope that this entry method becomes widely available because I am honestly sad that I don't get to use it anymore. I think that a lot of blind people would appreciate this method since it is so much easier than using the typical typing interface." Participant 4 preferred FlexType to their typical method,

stating, “When I had to go back to entering on the QWERTY keyboard on screen, it felt slow and clumsy.” Comments from these users that were negative of FlexType were mostly related to difficulty getting used to the technique, bugs with the app for the study, and issues with tap recognition. Participant 10 suggested the possibility of using VoiceOver to select the groups, which would alleviate the issue they were encountering with selecting group four.

Chapter 9

Conclusions

9.1 Discussion

In this work we presented three iterations of an ambiguous interface for nonvisual text entry. This interface was enhanced by an offline keyboard optimization study, an interview study with target users, and offline decoder model tuning. The resulting FlexType interface provides a non-Braille entry technique using a small group of quick input gestures. With this interface, users reached parity with their performance using a Qwerty keyboard with VoiceOver. While in this work we investigated its performance on a touchscreen, the location-independent nature of FlexType allows it to be used to provide accessible text input on a variety of platforms.

In Chapter 2, we found success with Tap123: our non-Braille ambiguous keyboard. Users selected from six Qwerty-based character groups by tapping with one, two, or three fingers on the left or right side of the screen. Users were able to achieve an entry

rate of 19.1 words per minute with a 2.1% error rate after eight hours of practice. The main things that we took away from this study were the abundance of disambiguation errors and the need for word suggestions. These two needs fueled the studies that we conducted in Chapters 3 and 5, as we sought to improve our interface in every way possible.

In Chapter 3, we sought to improve this performance by optimizing the character groupings to reduce disambiguation errors. We conducted offline experiments to optimize both four-group (T4) and six-group (T6) keyboards, both with and without alphabetical constraints. Our T6 groupings (both constrained and unconstrained) were able to easily out-score our previous Qwerty-based groupings on both clarity metrics. Our constrained T4 groupings did not perform quite as well, but one of the unconstrained T4 groupings was able to best the T6 Qwerty groupings on the WER clarity metric. This showed that it was feasible to reduce the number of character groups to four, which, in our opinion, made it much more practical for mobile use since groups could be selected by tapping with up to four fingers on a single hand. It also allowed us to easily remove the location dependence that was present in the original Tap123 interface.

In Chapter 4, we developed the FlexType interface, altering our group mappings to rely only on the number of fingers with which a user tapped and not on the location of the taps. We compared the best T4 constrained and unconstrained groups we found in Chapter 3 with a user study, finding no significant difference in user performance, with the exception of session 1, where users backspaced significantly more with the unconstrained groups. While the unconstrained optimization did increase the percentage of words in the first position of the n-best list returned by the decoder, this

did not significantly reduce the users' exploration of disambiguation alternatives. In the last four sessions, participants entered text at 12.0 words per minute with a 2.0% character error rate using the constrained groups, and at 13.5 words per minute with a 1.8% character error rate using the unconstrained groups. While these entry rates were slower than we observed in Chapter 2, participants were entering text with one hand instead of two, and the change from six character groups to four may have led to an increase in disambiguation errors, even with the optimization of the groups.

In Chapter 5, we explored different ways to present word suggestions to users using only audio feedback. We found that users had quite poor performance with more than two fully simultaneous voices. However, if we added a 0.25 s delay before starting each subsequent voice, users performed about as well with three voices as with two fully sequential voices. This created a compromise between the speed of fully simultaneous suggestions and the accuracy of fully sequential suggestions. While the results of these studies showed some potential, they were conducted independently of a text entry task. It is possible that the cognitive load of simultaneous speech would have a stronger impact on a user that is also trying to compose and type text.

The interview study we conducted in Chapter 6 allowed us to gather information about blind users' daily use of nonvisual text input methods, and to solicit feedback to refine FlexType further. The most common feedback we received was to improve the accuracy of speech recognition, to improve error correction techniques, and to reduce the barrier to entry for new text input methods. In general, participants did not use word suggestions, and several found them to be disruptive to their entry. Because of this, we chose not to proceed with our work from Chapter 5.

In Chapter 7 we tuned the parameters of the decoder based on the data from Chapter 4. We explored different input models that allowed modifications to users' input in the disambiguation search to attempt to correct possible errors. While we explored using a recurrent neural network language model to re-score disambiguation hypotheses during the search, this decreased the performance of the decoder model. This was perhaps due to the particular RNN that we used, which had limited training data. Our final selected model allowed for substitutions within hypotheses, with a higher penalty for substituted groups that were further from the input groups. This suggests that even when users selected an incorrect group, they tended to be closer to the correct group.

With a reduced barrier to entry in mind, we selected the constrained groups developed and investigated in Chapters 3 and 4 for further investigation in Chapter 8. We added *letter entry mode* to FlexType to allow users to type any word, even if it was not recognized by the decoder. We conducted a longitudinal study to compare blind users' performance using FlexType to their typical text input methods. Three users input text using Braille Screen Input and had an average entry rate of 26.9 words per minute. These same users averaged 11.2 words per minute with FlexType, which was significantly slower. Eight users input text using a Qwerty keyboard with VoiceOver at 7.5 words per minute, which was not significantly different from those users' FlexType entry rates of 7.0 words per minute. Feedback from some of the VoiceOver users stated that it felt easier and faster to use FlexType instead of their typical interface. Combined, participants entered text at an average of 8.2 words per minute using FlexType.

9.2 Future Work

9.2.1 Long Term Performance Evaluation

In Chapter 8's study, we compared FlexType to users' typical text input methods. Even though this was a longitudinal study, users only had four sessions of training (designed to be about one hour each) before we began the evaluation sessions. When considering the fact that users enter text with their baseline methods daily, this comparison may not show the full potential of FlexType. In future work, it would be interesting to evaluate how users improve with FlexType with continued practice by releasing the interface as a standalone text input method that users could enable for their daily uses. This would require additional consideration on how to enter characters such as capital letters, numbers, and punctuation, but it would allow us to test P9's hypothesis that for them FlexType could become faster than Braille Screen Input.

Future work could also explore a long term performance evaluation of letter entry mode. As we discussed in Section 8.3.4, we hypothesize that enough practice with FlexType's letter entry could remove the need for audio feedback. This mode could be particularly beneficial for users with both visual and hearing impairments, or in situations where audio feedback is not practical such as noisy environments.

9.2.2 Word Predictions

In our final study, we chose not to integrate word predictions into the FlexType interface. While the feedback from our interview study in Chapter 6 was mostly negative towards word predictions, some participants in Chapter 8's study suggested that it may be beneficial for users to have the option to enable predictions. Future work could continue to evaluate the audio word suggestions we explored in Chapter 5 by placing them into a full text entry task. We could compare the simultaneous presentation method of a 0.25 s delay to a control condition similar to a commercial keyboard and evaluate the trade-offs between entry and error rate, how frequently suggestions are used, and how frequently incorrect suggestions are selected. Another option, as we discussed in Section 8.4, would be to play the single most likely prediction after each tap and allow users to swipe right to select it. This would enable users to know which word will be selected when they swipe right and potentially increase their entry rate by eliminating keystrokes.

9.2.3 Alternate Sensors

In this work we focused on investigating FlexType's performance as a nonvisual text input method on a mobile device. However, FlexType has broader applications in accessible text input with a variety of sensors.

Virtual and augmented reality headsets do not have a touchscreen, and often rely on midair keyboards for text input. FlexType could be adapted for nonvisual text

input on these devices, using finger-to-thumb gestures to select each group. The headsets could detect these gestures through computer vision, wearable gloves [9], finger-mounted pressure sensors [33], or surface electromyography via electrodes on the forearm [4].

Neurodegenerative diseases such as amyotrophic lateral sclerosis (ALS) can cause motor impairments that often progress with time. When symptoms first begin, users may have difficulty moving their finger to a precise location, but may be able to select larger targets. As the diseases progress, users may lose the ability to speak altogether and rely on Alternative and Augmentative Communication (AAC) systems to communicate with other people. These AAC systems can take many forms, such as eye-gaze typing [31, 51], row-column scanning [41], or brain-computer interfaces (BCIs) [21, 26, 28].

Ambiguous keyboards have previously been used as AAC aids for single-switch users. Mackenzie and Felzer [42] created a Scanning Ambiguous Keyboard (SAK) with three character groups. In a scanning keyboard, the system highlights one key at a time in a cyclic pattern. To enter text, users activate their switch when the key they wish to select is highlighted. In the case of the SAK, there were three keys each containing a character group and a fourth containing the space character. After selecting the space key, the SAK would scan through the list of matching words in order of decreasing frequency in a text corpus.

Steady-state visually evoked potentials (SSVEPs) can be elicited by flickering a visual stimulus and measuring the neural responses at that frequency [34]. Work by Higger et al. [28] has explored using SSVEPs to determine which group of characters a user

is looking at by flickering lights at different frequencies in the vicinity of each group. However, their Shuffle Speller interface assigned letters to different groups each time it measured a user's response. While the flexible group assignments allowed the system to more easily discern between possible target characters (by placing them in different groups), it took additional time to relocate each character for each selection, potentially slowing the overall entry rate. In another avenue of future work, we could explore an implementation of FlexType that uses SSVEPs to select from static ambiguous groups instead of touchscreen interactions.

9.2.4 Large Language Models

In this work we used the VelociTap decoder [64] to perform disambiguation and turn sequences of character groups into words. For our final study in Chapter 8, disambiguation was performed on a remote server, which could lead to latency and privacy concerns. As we explained in Chapter 2, VelociTap uses n-gram language models to make predictions based on the frequencies of character and word sequences in the models' training data. Recently, large language models (LLMs) have become quite common and effective tools for natural language processing tasks such as text generation [17, 48, 56, 69]. These LLMs tune billions of parameters on vast quantities of data, which requires the computing resources of a large organization. The training of the LLaMA model with 65B parameters used over a trillion tokens of training data and took over one million GPU-hours [56], meaning that training this model on a single GPU would take over 100 years.

Without access to the same level of computing resources, researchers can use fine-tuning to adapt published LLMs to their needs [18]. One example of this is fine-tuning a model for a specific domain like biomedical text, where there is also a limited amount of training data [54]. Another example is fine-tuning an LLM to perform a specific task such as classifying text based on topic or sentiment analysis [29]. An extension of the work presented in this thesis could fine-tune a large language model to disambiguate sequences of character groups into words, either after the completion of the word as we did here, or by producing likely word completions based on the first few groups selections in a word (or both). To do this, we could use the data we collected in Chapters 4 and 8 to amass examples of user input (sequences of character groups) with the corresponding desired output from the LLM (disambiguation or word completion). We would then create an additional model layer on top of an LLM and train it using our examples. We could conduct both offline computational studies and user studies to investigate the potential performance gains provided by the fine-tuned LLM.

References

[1] Abdolrahmani, A., Kuber, R., and Hurst, A. An empirical investigation of the situationally-induced impairments experienced by blind mobile device users. In *Proceedings of the 13th International Web for All Conference* (New York, NY, USA, 2016), W4A '16, Association for Computing Machinery.

[2] Adhikary, J., Berger, J., and Vertanen, K. Accelerating text communication via abbreviated sentence input. In *Proceedings of the Joint Conference of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing* (2021), pp. 6574–6588.

[3] Ahmed, T., Hoyle, R., Connelly, K., Crandall, D., and Kapadia, A. Privacy concerns and behaviors of people with visual impairments. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems* (New York, NY, USA, 2015), CHI '15, Association for Computing Machinery, p. 3523–3532.

[4] Al-Timemy, A. H., Bugmann, G., Escudero, J., and Outram, N. Classification of finger movements for the dexterous hand prosthesis control with

surface electromyography. *IEEE Journal of Biomedical and Health Informatics 17*, 3 (2013), 608–618.

[5] Azenkot, S., and Lee, N. B. Exploring the use of speech input by blind people on mobile devices. In *Proceedings of the 15th International ACM SIGACCESS Conference on Computers and Accessibility* (New York, NY, USA, 2013), ASSETS '13, Association for Computing Machinery.

[6] Azenkot, S., Wobbrock, J. O., Prasain, S., and Ladner, R. E. Input finger detection for nonvisual touch screen text entry in perkinput. In *Proceedings of Graphics Interface 2012* (CAN, 2012), GI '12, Canadian Information Processing Society, p. 121–129.

[7] Bi, X., Smith, B. A., and Zhai, S. Quasi-qwerty soft keyboard optimization. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (New York, NY, USA, 2010), CHI '10, Association for Computing Machinery, p. 283–286.

[8] Bonner, M. N., Brudvik, J. T., Abowd, G. D., and Edwards, W. K. No-look notes: Accessible eyes-free multi-touch text entry. In *Pervasive Computing* (Berlin, Heidelberg, 2010), P. Floréen, A. Krüger, and M. Spasojevic, Eds., Springer Berlin Heidelberg, pp. 409–426.

[9] Bowman, D. A., Wingrave, C. A., Campbell, J. M., Ly, V. Q., and Rhoton, C. J. Novel uses of pinch gloves™ for virtual environment interaction techniques. *Virtual reality : the journal of the Virtual Reality Society 6*, 3 (2002), 122–129.

[10] Brungart, D., and Simpson, B. Cocktail party listening in a dynamic multitalker environment. *Perception & Psychophysics 69*, 1 (2007), 79–91.

[11] Brungart, D. S. Informational and energetic masking effects in the perception of two simultaneous talkers. *The Journal of the Acoustical Society of America 109*, 3 (2001), 1101–1109.

[12] Cerney, M. M., Mila, B. D., and Hill, L. C. Comparison of mobile text entry methods. *Proceedings of the Human Factors and Ergonomics Society Annual Meeting 48*, 5 (2004), 778–782.

[13] Cherry, E. C. Some experiments on the recognition of speech, with one and with two ears. *The Journal of the Acoustical Society of America 25*, 5 (1953), 975–979.

[14] Choudhary, T., Kulkarni, S., and Reddy, P. A braille-based mobile communication and translation glove for deaf-blind people. In *2015 International Conference on Pervasive Computing (ICPC)* (New York, NY, USA, 2015), Institue of Electrical and Electronics Engineers, pp. 1–4.

[15] Darwin, C. J., Brungart, D. S., and Simpson, B. D. Effects of fundamental frequency and vocal-tract length changes on attention to one of two simultaneous talkers. *The Journal of the Acoustical Society of America 114*, 5 (2003), 2913–2922.

[16] De Rosa, M., Fuccella, V., Costagliola, G., Adinolfi, G., Ciampi, G., Corsuto, A., and Di Sapia, D. T18: an ambiguous keyboard layout for smartwatches. In *2020 IEEE International Conference on Human-Machine Systems (ICHMS)* (2020), pp. 1–4.

[17] Devlin, J., Ming-Wei, C., Lee, K., and Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv.org* (2019).

[18] Dodge, J., Ilharco, G., Schwartz, R., Farhadi, A., Hajishirzi, H., and Smith, N. Fine-tuning pretrained language models: Weight initializations, data orders, and early stopping. arXiv 2002.06305, 2020.

[19] Dunlop, M., and Levine, J. Multidimensional pareto optimization of touchscreen keyboards for speed, familiarity and improved spell checking. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (New York, NY, USA, 2012), CHI '12, Association for Computing Machinery, p. 2669–2678.

[20] Egan, J. P., Carterette, E. C., and Thwing, E. J. Some factors affecting multi-channel listening. *The Journal of the Acoustical Society of America 26*, 5 (1954), 774–782.

[21] Farwell, L., and Donchin, E. Talking off the top of your head: toward a mental prosthesis utilizing event-related brain potentials. *Electroencephalography and clinical neurophysiology 70*, 6 (1988), 510–523.

[22] Fischer, A. R. H., Price, K. J., and Sears, A. 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 (2005), 279–304.

[23] Gaines, D. Exploring an ambiguous technique for eyes-free mobile text entry. In *Proceedings of the 20th International ACM SIGACCESS Conference on Computers and Accessibility* (New York, NY, USA, 2018), ASSETS '18, Association for Computing Machinery, p. 471–473.

[24] Gaines, D., Baker, M. M., and Vertanen, K. FlexType: Flexible text input with a small set of input gestures. In *Proceedings of the 28th International Conference on Intelligent User Interfaces* (New York, NY, USA, 2023), IUI '23, Association for Computing Machinery, p. 584–594.

[25] Gong, J., and Tarasewich, P. Alphabetically constrained keypad designs for text entry on mobile devices. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (New York, NY, USA, 2005), CHI '05, Association for Computing Machinery, p. 211–220.

[26] Guan, C., Thulasidas, M., and Wu, J. High performance p300 speller for brain-computer interface. In *IEEE International Workshop on Biomedical Circuits and Systems, 2004* (2004), IEEE, pp. S3/5/INV–S3/13.

[27] Guerreiro, J., and Gonçalves, D. Faster text-to-speeches: Enhancing blind people's information scanning with faster concurrent speech. In *Proceedings of the 17th International ACM SIGACCESS Conference on Computers & Accessibility* (New York, NY, USA, 2015), ASSETS '15, Association for Computing Machinery, p. 3–11.

[28] Higger, M., Quivira, F., Akcakaya, M., Moghadamfalahi, M., Nezamfar, H., Cetin, M., and Erdogmus, D. Recursive bayesian coding for bcis. *IEEE Transactions on Neural Systems and Rehabilitation Engineering 25*, 6 (2017), 704–714.

[29] Howard, J., and Ruder, S. Universal language model fine-tuning for text classification. arXiv 1801.06146, 2018.

[30] I. Scott Mackenzie, S. X. Z., and Soukoreff, R. W. Text entry using soft keyboards. *Behaviour & Information Technology 18*, 4 (1999), 235–244.

[31] Istance, H. O., Spinner, C., and Howarth, P. A. Providing motor impaired users with access to standard graphical user interface (gui) software via eye-based interaction. In *Proceedings of the 1st european conference on disability, virtual reality and associated technologies (ECDVRAT'96)* (1996).

[32] James, C. L., and Reischel, K. M. Text input for mobile devices: Comparing model prediction to actual performance. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (New York, NY, USA, 2001), CHI '01, Association for Computing Machinery, p. 365–371.

[33] Jiang, H., Weng, D., Zhang, Z., and Chen, F. Hifinger: One-handed text entry technique for virtual environments based on touches between fingers. *Sensors (Basel, Switzerland) 19*, 14 (2019), 3063–3086.

[34] Joon Kim, Y., Grabowecky, M., Paller, K. A., Muthu, K., and Suzuki, S. Attention induces synchronization-based response gain in steady-state visual evoked potentials. *Nature Neuroscience 10*, 1 (2007), 117–125.

[35] Kane, S. K., Bigham, J. P., and Wobbrock, J. O. Slide rule: Making mobile touch screens accessible to blind people using multi-touch interaction techniques. In *Proceedings of the 10th International ACM SIGACCESS Conference on Computers and Accessibility* (New York, NY, USA, 2008), Assets '08, Association for Computing Machinery, p. 73–80.

[36] Karimi, P., Brady, E., Martin-Hammond, A., and Bolchini, D. "i stepped into a puddle": Non-visual texting in nomadic contexts. In *Proceedings of the 20th International Web for All Conference* (New York, NY, USA, 2023), W4A '23, Association for Computing Machinery, p. 32–43.

[37] Karrenbauer, A., and Oulasvirta, A. Improvements to keyboard optimization with integer programming. In *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology* (New York, NY, USA, 2014), UIST '14, Association for Computing Machinery, p. 621–626.

[38] Krzywinski, M. Carpalx keyboard layout optimizer. http://mkweb.bcgsc.ca/carpalx/, 2005.

[39] Lee, D., Kim, J., and Oakley, I. Fingertext: Exploring and optimizing performance for wearable, mobile and one-handed typing. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems* (New York, NY, USA, 2021), CHI '21, Association for Computing Machinery.

[40] Lesher, G. W., Moulton, B. J., and Higginbotham, D. J. Optimal character arrangements for ambiguous keyboards. *IEEE Transactions on Rehabilitation Engineering 6*, 4 (1998), 415–423.

[41] Lin, Y.-L., Wu, T.-F., Chen, M.-C., Yeh, Y.-M., and Wang, H.-P. Designing a scanning on-screen keyboard for people with severe motor disabilities. In *Computers Helping People with Special Needs* (Berlin, Heidelberg, 2008), K. Miesenberger, J. Klaus, W. Zagler, and A. Karshmer, Eds., Springer Berlin Heidelberg, pp. 1184–1187.

[42] Mackenzie, I. S., and Felzer, T. Sak: Scanning ambiguous keyboard for efficient one-key text entry. *ACM Trans. Comput.-Hum. Interact. 17*, 3 (jul 2010).

[43] Mascetti, S., Bernareggi, C., and Belotti, M. TypeInBraille: Quick eyes-free typing on smartphones. In *Computers Helping People with Special*

Needs (Berlin, Heidelberg, 2012), K. Miesenberger, A. Karshmer, P. Penaz, and W. Zagler, Eds., Springer Berlin Heidelberg, pp. 615–622.

[44] Nicolau, H., Montague, K., Guerreiro, T., Rodrigues, A., and Hanson, V. L. Investigating laboratory and everyday typing performance of blind users. *ACM Trans. Access. Comput. 10*, 1 (mar 2017).

[45] Nicolau, H., Rodrigues, A., Santos, A., Guerreiro, T., Montague, K., and Guerreiro, J. a. The design space of nonvisual word completion. In *Proceedings of the 21st International ACM SIGACCESS Conference on Computers and Accessibility* (New York, NY, USA, 2019), ASSETS '19, Association for Computing Machinery, p. 249–261.

[46] Nivasch, K., and Azaria, A. A deep genetic method for keyboard layout optimization. In *2021 IEEE 33rd International Conference on Tools with Artificial Intelligence (ICTAI)* (2021), pp. 435–441.

[47] Oliveira, J., Guerreiro, T., Nicolau, H., Jorge, J., and Gonçalves, D. BrailleType: Unleashing braille over touch screen mobile phones. In *Proceedings of the 13th IFIP TC 13 International Conference on Human-Computer Interaction - Volume Part I* (Berlin, Heidelberg, 2011), INTERACT'11, Springer-Verlag, p. 100–107.

[48] OpenAI. Gpt-4 technical report. arXiv 2303.08774, 2023.

[49] Qin, R., Zhu, S., Lin, Y.-H., Ko, Y.-J., and Bi, X. Optimal-T9: An optimized T9-like keyboard for small touchscreen devices. In *Proceedings of the 2018 ACM International Conference on Interactive Surfaces and Spaces* (New York, NY, USA, 2018), ISS '18, Association for Computing Machinery, p. 137–146.

[50] Rodrigues, A., Nicolau, H., Montague, K., Guerreiro, J., and Guerreiro, T. Open challenges of blind people using smartphones. *International Journal of Human–Computer Interaction 36*, 17 (2020), 1605–1622.

[51] Sarcar, S., Panwar, P., and Chakraborty, T. Eyek: An efficient dwell-free eye gaze-based text entry system. In *Proceedings of the 11th Asia Pacific Conference on Computer Human Interaction* (New York, NY, USA, 2013), APCHI '13, Association for Computing Machinery, p. 215–220.

[52] Shinohara, K. Designing assistive technology for blind users. In *Proceedings of the 8th International ACM SIGACCESS Conference on Computers and Accessibility* (New York, NY, USA, 2006), Assets '06, Association for Computing Machinery, p. 293–294.

[53] Southern, C., Clawson, J., Frey, B., Abowd, G., and Romero, M. An evaluation of BrailleTouch: Mobile touchscreen text entry for the visually impaired. In *Proceedings of the 14th International Conference on Human-Computer Interaction with Mobile Devices and Services* (New York, NY, USA, 2012), MobileHCI '12, Association for Computing Machinery, p. 317–326.

[54] Tinn, R., Cheng, H., Gu, Y., Usuyama, N., Liu, X., Naumann, T., Gao, J., and Poon, H. Fine-tuning large neural language models for biomedical natural language processing. *Patterns 4*, 4 (2023), 100729.

[55] Tinwala, H., and MacKenzie, I. S. Eyes-free text entry with error correction on touchscreen mobile devices. In *Proceedings of the 6th Nordic Conference on Human-Computer Interaction: Extending Boundaries* (New York, NY, USA, 2010), NordiCHI '10, Association for Computing Machinery, p. 511–520.

[56] Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., Rodriguez, A., Joulin, A., Grave, E., and Lample, G. Llama: Open and efficient foundation language models. arXiv 2302.13971, 2023.

[57] Vertanen, K. Counting fingers: Eyes-free text entry without touch location. In *CHI '16: Extended Abstracts of the the ACM Conference on Human Factors in Computing Systems* (May 2016).

[58] Vertanen, K. *Probabilistic Text Entry-Case Study 3*, 1 ed. Association for Computing Machinery, New York, NY, USA, 2021, pp. 277–320.

[59] Vertanen, K., Fletcher, C., Gaines, D., Gould, J., and Kristensson, P. O. The impact of word, multiple word, and sentence input on virtual keyboard decoding performance. In *CHI '18: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (2018).

[60] Vertanen, K., and Kristensson, P. O. The imagination of crowds: Conversational AAC language modeling using crowdsourcing and large data sources. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing* (Edinburgh, Scotland, UK., July 2011), R. Barzilay and M. Johnson, Eds., Association for Computational Linguistics, pp. 700–711.

[61] Vertanen, K., and Kristensson, P. O. A versatile dataset for text entry evaluations based on genuine mobile emails. In *MobileHCI '11: Proceedings of the 13th International Conference on Human-Computer Interaction with Mobile Devices and Services* (2011).

[62] Vertanen, K., and Kristensson, P. O. A versatile dataset for text entry evaluations based on genuine mobile emails. In *Proceedings of the 13th International Conference on Human Computer Interaction with Mobile Devices and Services* (New York, NY, USA, 2011), MobileHCI '11, Association for Computing Machinery, p. 295–298.

[63] Vertanen, K., and Kristensson, P. O. Mining, analyzing, and modeling text written on mobile devices. *Natural Language Engineering 27* (2021), 1–33.

[64] Vertanen, K., Memmi, H., Emge, J., Reyal, S., and Kristensson, P. O. VelociTap: investigating fast mobile text entry using sentence-based decoding of touchscreen keyboard input. In *CHI '15: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (New York, NY, USA, 2015), Association for Computing Machinery, pp. 659–668.

[65] Vertanen, K., Memmi, H., and Kristensson, P. O. The feasibility of eyes-free touchscreen keyboard typing. In *ASSETS '13: Proceedings of the ACM SIGACCESS Conference on Computers and Accessibility* (2013).

[66] Xu, Z., Wong, P. C., Gong, J., Wu, T.-Y., Nittala, A. S., Bi, X., Steimle, J., Fu, H., Zhu, K., and Yang, X.-D. Tiptext: Eyes-free text entry on a fingertip keyboard. In *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology* (New York, NY, USA, 2019), UIST '19, Association for Computing Machinery, p. 883–899.

[67] Zhai, S., and Kristensson, P.-O. Shorthand writing on stylus keyboard. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (New York, NY, USA, 2003), CHI '03, Association for Computing Machinery, p. 97–104.

[68] Zhang, M. R., Wen, H., and Wobbrock, J. O. Type, then correct: Intelligent text correction techniques for mobile text entry using neural networks. In *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology* (New York, NY, USA, 2019), UIST '19, Association for Computing Machinery, p. 843–855.

[69] Zhang, S., Roller, S., Goyal, N., Artetxe, M., Chen, M., Chen, S., Dewan, C., Diab, M., Li, X., Lin, X. V., Mihaylov, T., Ott, M., Shleifer, S., Shuster, K., Simig, D., Koura, P. S., Sridhar, A., Wang, T., and Zettlemoyer, L. Opt: Open pre-trained transformer language models. arXiv 2205.01068, 2022.