FlexType: Flexible Text Input with a Small Set of Input Gestures

Dylan Gaines
Michigan Technological University
Houghton, Michigan, United States
dcgaines@mtu.edu

Mackenzie Baker
Michigan Technological University
Houghton, Michigan, United States
mmbaker1@mtu.edu

Keith Vertanen
Michigan Technological University
Houghton, Michigan, United States
vertanen@mtu.edu

ABSTRACT

In many situations, it may be impractical or impossible to enter text by selecting precise locations on a physical or touchscreen keyboard. We present an ambiguous keyboard with four character groups that has potential applications for eyes-free text entry, as well as text entry using a single switch or a brain-computer interface. We develop a procedure for optimizing these character groupings based on a disambiguation algorithm that leverages a long-span language model. We produce both alphabetically-constrained and unconstrained character groups in an offline optimization experiment and compare them in a longitudinal user study. Our results did not show a significant difference between the constrained and unconstrained character groups after four hours of practice. As expected, participants had significantly more errors with the unconstrained groups in the first session, suggesting a higher barrier to learning the technique. We therefore recommend the alphabetically-constrained character groups, where participants were able to achieve an average entry rate of 12.0 words per minute with a 2.03% character error rate using a single hand and with no visual feedback.

CCS CONCEPTS

• **Human-centered computing** → **Accessibility technologies**; **Empirical studies in ubiquitous and mobile computing**; **Text input**; *Auditory feedback*; *Empirical studies in HCI*.

KEYWORDS

text entry, accessibility, human-computer interaction, mobile keyboard

ACM Reference Format:
Dylan Gaines, Mackenzie Baker, and Keith Vertanen. 2023. FlexType: Flexible Text Input with a Small Set of Input Gestures. In *28th International Conference on Intelligent User Interfaces (IUI '23), March 27–31, 2023, Sydney, NSW, Australia.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3581641.3584077

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 has a visual impairment, they may not be able to see a virtual keyboard well enough (or at all) to provide precise input. This would impede their ability to enter text on both touchscreen devices and Augmented and Virtual Reality devices that lack the haptic feedback of a physical keyboard.

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 dwell-based eye-gaze typing [15] or row-column scanning [20].

All of these situations could be aided 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.

Each character group in an ambiguous keyboard can be mapped to a distinct gesture. For example, the Tap123 interface designed by Gaines [11] divided a Qwerty keyboard into six ambiguous groups. The number of fingers the user tapped with (between one and three) represented the row of the keyboard, and the left or right side of the screen the user tapped represented the side of the keyboard on which their intended character lies. In this work, by reducing the number of groups from six to four, we can remove all location dependency and allow users to tap a touchscreen with between one and four fingers on a single hand, as opposed to the bi-manual method used by Tap123. Since reducing the number of groups increases the number of characters in each group, there are more possible words to consider during the disambiguation process. While past studies have optimized ambiguous keyboards using simple word or character frequencies within a text corpus, they have not considered the impact of recognition using a long-span language model. Our paper makes three primary contributions:

(1) The design of FlexType, a four group ambiguous keyboard with no location dependency and many possible applications in accessible text input.
(2) A new ambiguous keyboard optimization procedure that incorporates a long-span language model in the process. To our knowledge, we are the first to consider a language-model-aware optimization procedure.

IUI '23, March 27–31, 2023, Sydney, NSW, Australia

ACM ISBN 979-8-4007-0106-1/23/03.
https://doi.org/10.1145/3581641.3584077

(3) A longitudinal user study comparing performance over time on two optimized character groups: one constrained to alphabetical order and one not constrained to alphabetical order.

While FlexType has many possible applications in accessible text input, in this work we focus on investigating its performance as an eyes-free text input method on a mobile device.

2 RELATED WORK

2.1 Accessibility

Some existing eyes-free text entry solutions rely on indicating Braille mappings to enter characters [2, 21, 23, 25, 28]. While Perkinput [2] users achieved 17.6 words per minute with one hand, it required both knowledge of the Braille character set and the ability to target specific areas of a touchscreen. For people with a motor impairment in addition to a visual impairment, this may not be feasible.

Other eyes-free interfaces allow for exploration of a keyboard using audio feedback and a secondary touch event to confirm selection [5, 17]. This technique is also used by modern Android and iOS mobile operating systems to enable eyes-free text entry. Another approach by Tinwala and MacKenzie [29] performs recognition on drawn characters. Performing multiple touch events per character can be quite slow, with Bonner et al. [5] finding entry rates of only 1.32 words per minute with their method.

A common text entry interface for people with severe motor impairments is row-column scanning [20, 30]. Row-column scanning interfaces cyclically highlight groups of characters (rows), allowing the highlighted group to be selected in some way (e.g. by pushing a button, twitching a muscle, or blinking an eye). The interface then highlights each character in the selected row in turn (columns), allowing the user to select their desired character.

An alternative single-switch input interface is Nomon [4, 7]. Nomon displays multiple character and word options, each with its own rotating clock. A user makes a selection by repeatedly activating their switch when the clock for their desired target is at noon. The target is selected after a variable number of switch activations that depends on the likelihood of the target and how precisely the user triggers their switch.

Instead of activating a switch, some people with severe motor impairments may instead use an eye-gaze interface [15, 24, 27]. In a typical dwell-based eye keyboard, a user looks at their intended key for a specified amount of time (typically a second or so) and this triggers the key. If a person can neither reliably trigger a switch or control their eye gaze, they may need to resort to a brain-computer interface (BCI) system to decipher brain signals into text [10, 13, 14].

2.2 Ambiguous Keyboards

Ambiguous keyboards have been utilized 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 [26] used 9 keys on a smartwatch keyboard. The T18 keyboard [8] extended this to combine ambiguous and non-ambiguous keys into an interface with 18 keys. Users typed with the T18 keyboard at 15.7 words per minute. TipText [34] allowed users to input text to a wearable device by using their thumb to type on an invisible ambiguous keyboard on the tip of their index finger. Users wrote with TipText at about 13.3 words per minute.

Ambiguous keyboards have previously been used as AAC aids for single-switch users. Mackenzie and Felzer [22] 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.

In HiFinger [16], 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.

2.3 Optimization

There has been a considerable amount of past research on optimizing keyboard layouts for a variety of metrics such as finger travel distance, keystroke efficiency, familiarity, and tap clarity. Lesher et al. [19] showed that evaluating every possible set of character groupings is computationally infeasible, but proposed an algorithm that efficiently finds locally optimized groupings for an ambiguous keyboard. We describe this algorithm in detail here since we modify it later in this work. The first step in their algorithm is to compute a confusability matrix for a 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. 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

$$M(\alpha, \beta) = C(\alpha, \beta) + C(\beta, \alpha), \tag{1}$$

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 is the sum of the mutual confusabilities between each pair of characters.

After computing the confusability matrix, Lesher et al. [19] run their n-opt algorithm. The algorithm starts with a valid arrangement of characters into groupings. On each pass, it checks every possible tuple of n characters to see if shuffling the characters results in a better overall arrangement according to whatever metric is being optimized (in this case, the confusability of the groupings). For example, a two-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. The n-opt algorithm requires factorially increasing computations for higher values of n; the highest pass completed by Lesher et al. was five-opt [19]. Since the n-opt

algorithm only finds local optima, the authors start with many initial arrangements and run the two-opt algorithm on each. They take the result with the best performance and further improve it via the five-opt algorithm.

Gong and Tarasewich [12] compared an ambiguous layout that was constrained to be in strict alphabetical order with a layout that was unconstrained and freely optimized. Both layouts were optimized to minimize the number of keystrokes required on average to enter a character. While they were able to fully enumerate the possible constrained layouts, they used a genetic algorithm to optimize the unconstrained layouts. The authors found that the more familiar constrained layout aided users' ability to learn the interface.

Other authors found similar results when comparing freely optimized non-ambiguous keyboards to ones with familiarity constraints. For example, Bi et al. [3] 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, thereby increasing the entry rate. However, the Quasi-Qwerty layout 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 layout had better movement efficiency than the standard Qwerty layout, it was not as efficient as the freely optimized layout. 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 layout, and finally the standard Qwerty layout. The authors concluded that the Quasi-Qwerty layout was effective at obtaining an increased movement efficiency while not adding too much time to the initial visual search. While through practice users would improve with the freely optimized layout, using a more familiar layout reduced the amount of practice needed.

Instead of simply enforcing a strict Qwerty restriction on their non-ambiguous keyboard layout, Dunlop and Levine [9] chose to optimize their keyboard layout on three different parameters:

- **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 layout. 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** — Calculated to reduce the number of adjacent characters in a layout 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 to a probability by dividing by the total number of badgram occurrences. The authors defined their tap clarity metric for a given keyboard layout as

$$M_{tap\_clarity} = 1 - \sum_{\forall i,j \in \alpha} \begin{cases} p_{ij} & \text{if } neighbors_{ij} \\ 0 & \text{else} \end{cases}, \qquad (2)$$

  where p_{ij} is the badgram probability for letters i and j in the symbol set α and $neighbors_{ij}$ is true if i and j are adjacent in the layout.
- **Familiarity** —The similarity of a given layout to the Qwerty layout 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 layouts that had most of their letters near their Qwerty positions.

Qin et al. [26] also used multiple parameters 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 tap sequences given an ambiguous layout. They then defined the clarity of a layout 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 the previous paper, this is not relevant to interfaces such as FlexType that remove location dependency. In their interface, 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. [18] 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. [19].

We have discussed here many different text entry methods that relate to FlexType with respect to their ambiguity, accessibility, or optimization. For easier comparison between FlexType and past work, we have classified the key features of previous text input methods in Table 1.

3 OFFLINE OPTIMIZATION EXPERIMENT

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. [19] 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. [26] 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.

Name	Ambiguity	Disambiguation Method	Basis	Platform	Accessibility	Optimization Criteria
FlexType	T4	Language model	Alphabetical	Smartphone	Eyes-free	Clarity
Tap123 [11]	T6	Language model	QWERTY	Smartphone	Eyes-free	None
Perkinput [2]	None	N/A	Braille	Smartphone, tablet	Eyes-free	None
Optimal-T9 [26]	T9	Language model	QWERTY	Smartwatch	None	Clarity, speed
T18 [8]	T18	Word frequency	QWERTY	Smartwatch	None	Clarity
TipText [34]	T6	Language model	QWERTY	Smartwatch	Eyes-free	Clarity, tap accuracy
RC Scan [30]	None	N/A	Letter freq.	Desktop	Single switch	Letter frequency
SAK [22]	T3	Word frequency	Alphabetical	Desktop	Single switch	Scans per character
HiFinger [16]	T6	Second action	Alphabetical	Virtual env.	None	None
Alpha Constrained [12]	T8	Word frequency	Alphabetical	Pocket PC	None	Keystrokes
FingerText [18]	T5, T10	Language model	QWERTY	HMD	None	Speed, accuracy, comfort, clarity
Quasi-Qwerty [3]	None	N/A	QWERTY	Smartphone	None	Finger travel distance

Table 1: The key features of FlexType and related text entry interfaces. The 'T' notation in the Ambiguity column refers to the number of character groups used by the interface.

We optimize sets that each have four groups of characters. 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. [6] and by Jiang et al. [16]. 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 [6], finger-mounted pressure sensors [16], or surface electromyography via electrodes on the forearm [1]. A four-group set could also be used in a scanning keyboard for single switch users. While the Scanning Ambiguous Keyboard developed by MacKenzie et al. [22] had three groups of letters, adding another group as well as long-span language modeling could improve disambiguation performance and reduce the need to select from a set of matching hypotheses after each word. The SAK also assumed that users were able to receive visual feedback to make their selections.

3.1 Procedure

We developed our optimization procedure by combining some of the ideas used for optimization in past work [9, 19], 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 [32] and consists of forum posts made by users on a mobile device.

3.1.1 Corpus Analysis. To perform this analysis, we adapted the ideas of Lesher et al. [19]. 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 used software based on the VelociTap decoder [33] with a 12-gram character model and a 4-gram word model to predict the most likely next 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, storing 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 the true word 'world', the matrix entries for the character pairs 'hw', 'eo', 'lr', and 'od' would be incremented.

3.1.2 Performance Metrics. During the optimization process, we need a way to compare one set of ambiguous groups to another to determine the better of the two. We investigate two different metrics to evaluate the potential performance of a given set.

The first metric is *badgram clarity*. Using the confusion matrix we generated during our corpus analysis, similar to Lesher et al. [19], 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 [9], to obtain badgram probabilities p_{ij}:

$$p_{ij} = \frac{C(i,j) + C(j,i)}{\sum_{\forall i,j|i \neq j} C(i,j) + C(j,i)}, \tag{3}$$

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

$$Clarity_{badgram} = 1 - \sum_{\forall i,j|i \neq j} \begin{cases} p_{ij} & \text{if } sameGroup(i,j) \\ 0 & \text{else} \end{cases}, \tag{4}$$

where $sameGroup(i,j)$ is true if i and j are in the same ambiguous group.

The second metric we investigate is *word error rate (WER) clarity*. To calculate WER clarity, we utilize the mobile test set from the corpus released by Vertanen and Kristensson [32]. We first define the groups that we are evaluating as an ambiguous keyboard in the VelociTap decoder [33], specifying which characters are in each group. We then iterate through each word in the first 250 phrases

in the test set. We simulate the taps that would be needed to type each word given the current set of groups. We then use the decoder to find the probability of each word that fits the ambiguous group sequence given the words in the phrase to the left of the current word. We look at the most probable word returned by the decoder, and determine if it predicted the true word correctly. We aggregate all of the predictions to define WER clarity as:

$$Clarity_{WER} = 1 - \frac{\sum_{\forall word} correct(word)}{n_{words}}, \quad (5)$$

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.

Badgram clarity is a simpler metric but 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 WER clarity requires us to iterate through a test phrase set to make predictions, it becomes computationally expensive to calculate for a large number of character groups.

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

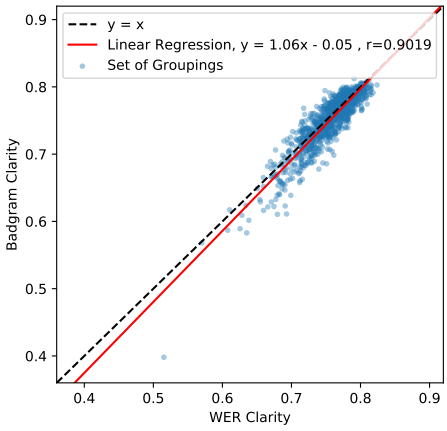

Figure 1: Comparison of the badgram and WER clarity metrics on 1000 random sets of groups. The dashed line represents $Clarity_{badgram} = Clarity_{WER}$. The solid red line represents the best fit linear regression.

3.1.3 Unconstrained Optimization. We began by optimizing sets freely, without any familiarity constraints. The goal was to determine the upper bound of performance that can be achieved, even if it takes additional training time for users. Since, as Lesher et al. [19] showed, an exhaustive search is computationally infeasible, we used a similar n-opt approach. In their work, when they examined the possible swaps between a set 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 two-opt pass, we checked the set where we simply swapped α and β as Lesher et al. did. However, we also checked the sets 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 generated all possible permutations (e.g. 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 and b were each in a separate group, $k = 2$). An example of all partitions generated on the permutation (ab) is $(|ab,\ a|b,\ ab|)$, where the vertical bar represents the boundary between groups. Note that there are $k-1$ dividers 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. For example, 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.

3.1.4 Badgram Clarity. We first optimized using badgram clarity. Similar to Lesher et al. [19], we ran our version of the two-opt algorithm on 50,000 random initial sets. We then ran our version of the five-opt algorithm on the best result from two-opt.

3.1.5 WER Clarity. As we discussed previously, the WER clarity metric is computationally expensive and requires iteration through the test phrases to check each set of groups. This, combined with the factorial growth of the n-opt algorithm prevented us from running a large number of initial sets and from running anything larger than two-opt. Using WER clarity, we ran 50 random initial sets through our two-opt algorithm.

3.1.6 Constrained Optimization. While the Tap123 interface [11] was based on a Qwerty keyboard layout, 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, we will focus on alphabet-based groupings. While Gong and Tarasewich [12] implemented a strict alphabetical constraint, we investigated allowing a small number of characters to shift outside of alphabetical order.

Constraining the sets 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 to that which we used on our unconstrained sets, but this time we did not allow

Constraint	Opt. metric	Badgram clarity	WER clarity
None	Badgram	0.8191	0.8209
None	WER	0.7990	0.8542
Alphabetical	Badgram	0.8030	0.7872

Table 2: Both clarity scores for the top unconstrained and constrained sets. The unconstrained sets were optimized with respect to two different metrics and both results are reported.

any groups to be empty, as moving a letter from a group where it is alone will never increase performance. This left us with 26 possible divider locations (our 27 character alphabet size minus 1, since the divider cannot be located on either end). For a set with four groups, there are 2,600 possible partitions (3 dividers over 26 positions = $\binom{26}{3}$).

For each partition, we selected every combination of s characters, where s was the maximum number of characters allowed to move from their group. We tested values of s from 0 to 4, inclusive. 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 tuple being considered, here characters could move to any group. Due to the factorial growth of the number of possible combinations, we were only able to complete this optimization using the badgram clarity metric.

3.2 Results

3.2.1 Unconstrained. We computed both clarity scores for our best unconstrained sets. The results are shown in the upper portion of Table 2. When optimizing for badgram clarity, the best score we were able to achieve was 0.8191. There were 208 of the 50,000 initial sets of groups that when run through our two-opt algorithm produced final sets 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 sets were all identical. This set 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 set through the five-opt algorithm, there were no swaps made in the entire pass. This set also achieved a WER clarity score of 0.8209.

Optimizing with WER clarity, we had hoped to see multiple of the 50 random initial sets converge to the same optima as we had with the badgram clarity metric, but this was not the case; the best clarity was only achieved by a single set. We were able to achieve a peak WER clarity of 0.8542 with the following groups:

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

This set had a badgram clarity score of 0.7990, which was surprisingly far from its WER clarity score given the correlation shown in Figure 1.

It was interesting to note that in both of these sets the vowels were separated as much as possible. With only four groups, it was impossible to separate all of the vowels, but e and u belonged to the same group in both sets, as well as i and y.

3.2.2 Constrained. Optimizing our constrained sets of groups for badgram clarity without allowing any characters to stray from alphabetical order yielded a badgram clarity score of 0.8030. The set that produced this was:

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

As seen in Table 2, this set of groups also had a WER clarity rating of 0.7872. As expected, both of these clarity ratings were lower than the top ratings achieved by unconstrained sets. As with the unconstrained sets, the vowels were separated as much as possible, with a and e in the first group, and u and y in the fourth group.

When we allowed characters to stray from alphabetical order, we saw only marginal gains in the clarity metrics (1.02% for the first character out of order, and about 0.25% for each character after that), which we determined would not be worth the additional cognitive overhead for users.

4 USER STUDY

Based on our results in our offline experiments, we opted to use the set with no swaps as our constrained groups in our user study. For our unconstrained groups, we chose the set optimized using the 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. We tested the constrained and unconstrained groups in a longitudinal user study. While past work has shown that users perform better initially on familiar layouts, we wanted to evaluate how long this benefit would persist. We also wanted to evaluate the extent to which the differences in the WER clarity metric would impact real user performance.

4.1 FlexType System Description

We used a OnePlus 5T smartphone for the user study. To enter text, 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 set of groups that was 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 simulate eyes-free text entry. During the input of a word, the following gestures were available:

- Tap with 1–4 fingers — Select a character from the corresponding group. After each tap, the device would read all characters in the selected group via text-to-speech.
- Swipe left with one finger — Backspace a single character.
- Swipe left with two fingers — Backspace all characters in the current word.
- Swipe right with one finger — Indicate that the current word is finished and trigger disambiguation.

Disambiguation was performed by software based on the VelociTap decoder [33]. We used a 12-gram character language model

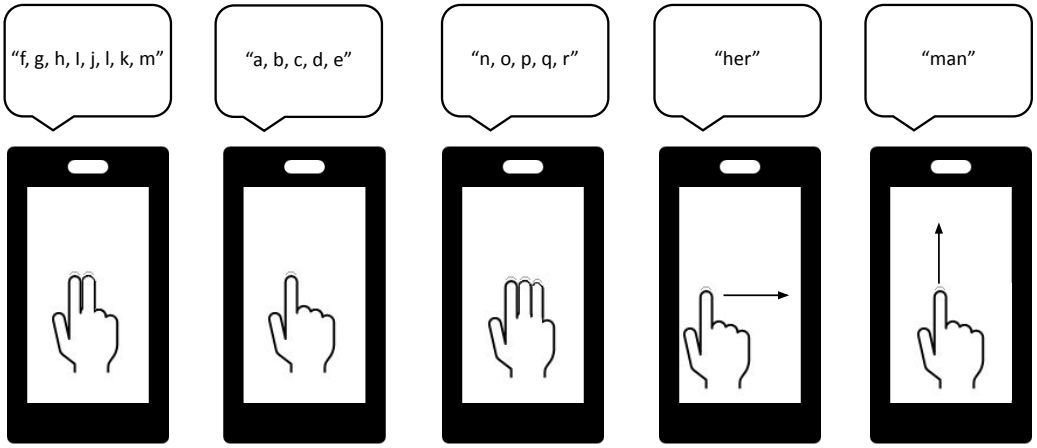

Figure 2: An example input sequence for the word 'man'. From left to right, the user enters each character by tapping with two fingers, one finger, and then three fingers. The letters corresponding to a group are read out after each tap. The user then swipes to the right and the word 'her' is recognized as most likely and read out. The user then swipes up to change 'her' to the second most likely word, 'man'.

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 placed into the text entry area, 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 2.

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, eight in each condition.

Each participant completed a total of eight sessions. Each session lasted approximately one hour and participants received US$10 per session as compensation. No 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. Once the user selected the 'Start' button, the FlexType interface opened, blocking the entire display. The prompts that the participants received varied based on the session they were completing. We created a progression designed to introduce them to the input technique as follows:

- Session 1 - Participants received single-letter prompts. The goal of this session was to teach the participants the groupings. 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.
- Sessions 4–8 - Phrases were only restricted to be no longer than six words to aide participants in remembering them correctly.

All phrases were drawn from the Enron mobile data set [31].

After finishing each transcription, participants swiped down with two fingers to complete the input. The participant was shown a summary screen with the reference 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. The

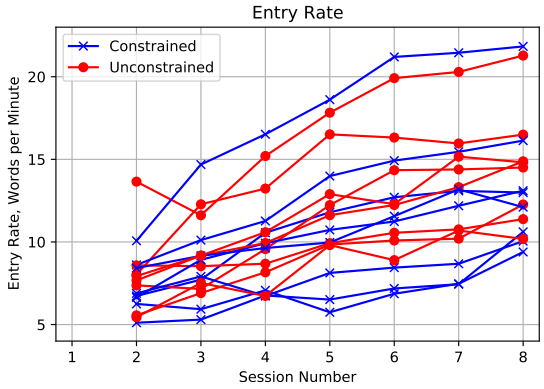

Figure 3: Entry rates of each participant throughout the study. Session 1 entry rates are not plotted since participants only entered single letters in this session.

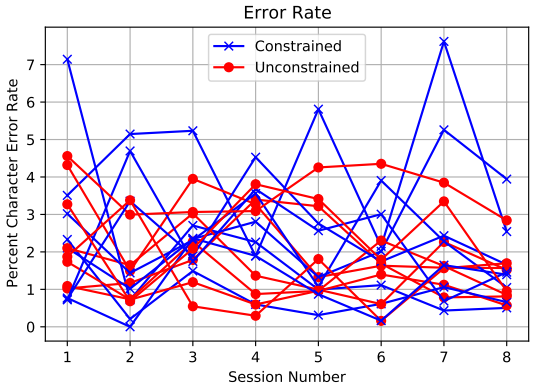

Figure 4: Character error rates of each participant throughout the study.

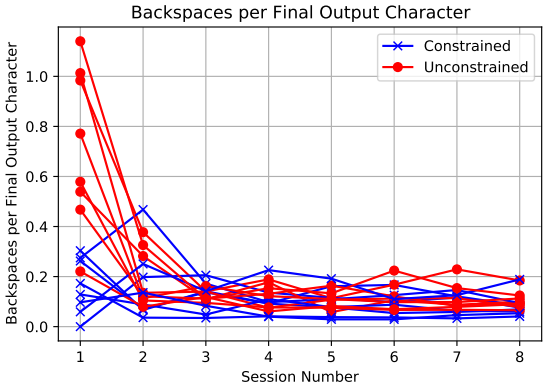

Figure 5: Backspaces per final output character for each participant during the user study.

incomplete participants' data was excluded from analysis. Participants in the Constrained condition were 19–22 years old (mean 20.75). One identified as male and seven identified as females. Participants in the Unconstrained condition were 20–45 years old (mean 23.88). Six identified as males 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. As our dependent variables, we measured different metrics of user performance and behavior. Since the beginning sessions were designed to train the users on the interface, the main results summarized in Table 3 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.

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 3, 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 3.

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. As shown in Figure 4, participants' error rates varied throughout their sessions with most participants having a CER of less than 5% in most sessions. Across the final four 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 3.

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 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 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 ($t(14) = -4.66, p < 0.001$), but there were no significant differences in any of the remaining sessions.

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

Condition	Entry rate (WPM)	Error rate (% CER)	Backspaces per char	Swipes per word
Constrained	$12.0 \pm 4.4[7.4, 20.8]$	$2.03 \pm 1.31[0.66, 4.53]$	$0.090 \pm 0.043[0.037, 0.154]$	$0.318 \pm 0.111[0.180, 0.468]$
Unconstrained	$13.5 \pm 3.3[9.9, 19.8]$	$1.81 \pm 0.91[1.01, 3.82]$	$0.111 \pm 0.037[0.069, 0.174]$	$0.288 \pm 0.080[0.186, 0.447]$
Statistical test	$t(14) = -0.75, p = 0.46$	$t(14) = 0.40, p = 0.69$	$t(14) = -1.06, p = 0.31$	$t(14) = 0.61, p = 0.55$

Table 3: The main results from sessions 5 through 8 of the user study. Results are reported in the format $mean \pm sd[min, max]$. The statistical tests reported are independent means t-tests.

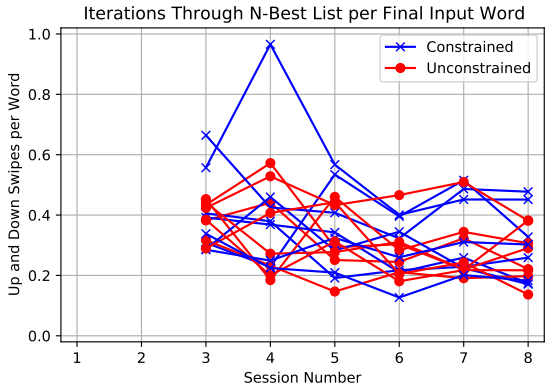

Figure 6: Total number of explorations through the n-best list per final input word for each participant. Sessions 1 and 2 did not use the n-best list and are not plotted.

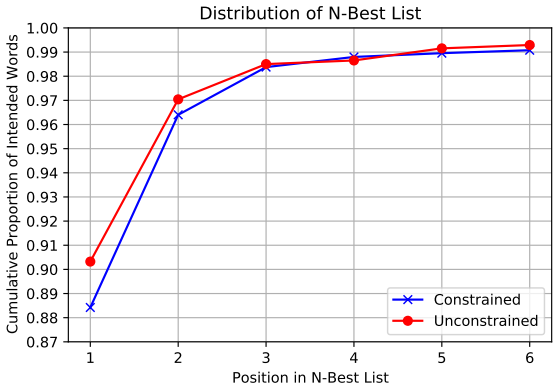

Figure 7: The cumulative proportion of intended words located at or before each position in the n-best list.

number of words that were input. We measured this from session 3 on since the n-best list was not used in the first two sessions. As shown in Figure 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 layout 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 (Table 3). 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 ($t(14) = 0.03, p = 0.97$).

Finally, we analyzed each word entered with the proper tap sequence and context to determine the distribution of the target word in the n-best list. The proportion of words, averaged among all participants in each condition, found at or before each position (e.g. position 2 includes words found in either position 1 or position 2) can be seen in Figure 7. As we expected from our optimization experiment, this proportion was higher for the Unconstrained 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 distribution is reported in Table 4. Using this distribution, we calculated the expected number of swipes per correct word for each condition. 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 participants in the Constrained condition did swipe less than we would have expected given the words that they entered. 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 was not in it. 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.

5 DISCUSSION

The goal of this work was to optimize both constrained and unconstrained ambiguous groups and then to compare user performance between the two. 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 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

Condition	Position 1	Position 2	Position 3	Position 4	Position 5	Position 6	Not in n-best
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: The distribution of intended word positions in the n-best list when they were entered with the correct tap sequence and context.

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, 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 was restricted to a maximum of six 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, this occurred in about 0.81% of words across both conditions where the user entered the proper tap sequence with the correct prior text. To remedy this issue, we could add a mode where users designate the exact desired character from their selected group in some way (e.g., by long pressing with the correct number of fingers and releasing when their desired letter from the corresponding group is read). While this will slow entry, it would 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. Gestures that do not require a touchscreen could include finger-to-thumb touches, detected by either pressure sensors [16] or gloves with conductive fabric [6].

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 sets of groupings, 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 [29], Perkinput: 17.6 WPM with one hand [2], and TipText: 13.3 WPM [34]). Future work could conduct an experiment that directly compares FlexType to a commonly available eyes-free text entry method (e.g., the braille input method now available on iOS).

6 CONCLUSION

Through a series of optimization experiments, we designed two ambiguous keyboards consisting of four groups of characters. Our character grouping optimization procedure took into account, for the first time, the impact of a recognition algorithm capable of utilizing prior words to predict the most likely word based on ambiguous input. In a multi-session user study, we found that our unconstrained groupings, while they had a slightly better clarity metric, did not perform meaningfully better than alphabetically-constrained groupings. We conclude that since the unconstrained groups did not produce a noticeable benefit in our long-term evaluation, it is not worth the higher barrier to entry that they create. For example, users of the unconstrained grouping backspaced over four times more often in their first hour of use. With both sets of groups, participants had performance similar to other eyes-free text entry methods. 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.

ACKNOWLEDGMENTS

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

REFERENCES

[1] Ali H. Al-Timemy, Guido Bugmann, Javier Escudero, and Nicholas Outram. 2013. 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. https://doi.org/10.1109/JBHI.2013.2249590

[2] Shiri Azenkot, Jacob O. Wobbrock, Sanjana Prasain, and Richard E. Ladner. 2012. Input Finger Detection for Nonvisual Touch Screen Text Entry in Perkinput. In *Proceedings of Graphics Interface 2012* (Toronto, Ontario, Canada) *(GI '12)*. Canadian Information Processing Society, CAN, 121–129.

[3] Xiaojun Bi, Barton A. Smith, and Shumin Zhai. 2010. Quasi-Qwerty Soft Keyboard Optimization. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Atlanta, Georgia, USA) *(CHI '10)*. Association for Computing Machinery, New York, NY, USA, 283–286. https://doi.org/10.1145/1753326.1753367

[4] Nicholas Ryan Bonaker, Emli-Mari Nel, Keith Vertanen, and Tamara Broderick. 2022. A Performance Evaluation of Nomon: A Flexible Interface for Noisy Single-Switch Users. In *CHI Conference on Human Factors in Computing Systems* (New Orleans, LA, USA) *(CHI '22)*. Association for Computing Machinery, New York, NY, USA, Article 495, 17 pages. https://doi.org/10.1145/3491102.3517738

[5] Matthew N. Bonner, Jeremy T. Brudvik, Gregory D. Abowd, and W. Keith Edwards. 2010. No-Look Notes: Accessible Eyes-Free Multi-touch Text Entry. In *Pervasive Computing*, Patrik Floréen, Antonio Krüger, and Mirjana Spasojevic (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 409–426. https://doi.org/10.1007/978-3-642-12654-3_24

[6] D. A Bowman, C. A Wingrave, J. M Campbell, V. Q Ly, and C. J Rhoton. 2002. Novel Uses of Pinch Gloves™ for Virtual Environment Interaction Techniques. *Virtual reality : the journal of the Virtual Reality Society* 6, 3 (2002), 122–129. https://doi.org/10.1007/s100550200013

[7] Tamara Broderick and David J. C. MacKay. 2009. Fast and Flexible Selection with a Single Switch. *PLoS ONE* 4, 10 (2009). https://doi.org/10.1371/journal.pone.0007481

[8] Mattia De Rosa, Vittorio Fuccella, Gennaro Costagliola, Giuseppe Adinolfi, Giovanni Ciampi, Antonio Corsuto, and Donato Di Sapia. 2020. T18: an ambiguous keyboard layout for smartwatches. In *2020 IEEE International Conference on Human-Machine Systems (ICHMS)*. 1–4. https://doi.org/10.1109/ICHMS49158.2020.9209483

[9] Mark Dunlop and John Levine. 2012. 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* (Austin, Texas, USA) *(CHI '12)*. Association for Computing Machinery, New York, NY, USA, 2669–2678. https://doi.org/10.1145/2207676.2208659

[10] L.A. Farwell and E. Donchin. 1988. 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. https://doi.org/10.1016/0013-4694(88)90149-6

[11] Dylan Gaines. 2018. Exploring an Ambiguous Technique for Eyes-Free Mobile Text Entry. In *Proceedings of the 20th International ACM SIGACCESS Conference on Computers and Accessibility* (Galway, Ireland) *(ASSETS '18)*. Association for Computing Machinery, New York, NY, USA, 471–473. https://doi.org/10.1145/3234695.3240991

[12] Jun Gong and Peter Tarasewich. 2005. Alphabetically Constrained Keypad Designs for Text Entry on Mobile Devices. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Portland, Oregon, USA) *(CHI '05)*. Association for Computing Machinery, New York, NY, USA, 211–220. https://doi.org/10.1145/1054972.1055002

[13] Cuntai Guan, M. Thulasidas, and Jiankang Wu. 2004. High performance P300 speller for brain-computer interface. In *IEEE International Workshop on Biomedical Circuits and Systems, 2004.* IEEE, S3/5/INV–S3/13. https://doi.org/10.1109/BIOCAS.2004.1454155

[14] Matt Higger, Fernando Quivira, Murat Akcakaya, Mohammad Moghadamfalahi, Hooman Nezamfar, Mujdat Cetin, and Deniz Erdogmus. 2017. Recursive Bayesian Coding for BCIs. *IEEE Transactions on Neural Systems and Rehabilitation Engineering* 25, 6 (2017), 704–714. https://doi.org/10.1109/TNSRE.2016.2590959

[15] Howell Owen Istance, Christian Spinner, and Peter Alan Howarth. 1996. 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)*.

[16] Haiyan Jiang, Dongdong Weng, Zhenliang Zhang, and Feng Chen. 2019. HiFinger: One-Handed Text Entry Technique for Virtual Environments Based on Touches between Fingers. *Sensors (Basel, Switzerland)* 19, 14 (2019), 3063–3086. https://doi.org/10.3390/s19143063

[17] Shaun K. Kane, Jeffrey P. Bigham, and Jacob O. Wobbrock. 2008. 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* (Halifax, Nova Scotia, Canada) *(Assets '08)*. Association for Computing Machinery, New York, NY, USA, 73–80. https://doi.org/10.1145/1414471.1414487

[18] DoYoung Lee, Jiwan Kim, and Ian Oakley. 2021. 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* (Yokohama, Japan) *(CHI '21)*. Association for Computing Machinery, New York, NY, USA, Article 283, 15 pages. https://doi.org/10.1145/3411764.3445106

[19] Gregory W. Lesher, Bryan J. Moulton, and D. Jeffery Higginbotham. 1998. Optimal character arrangements for ambiguous keyboards. *IEEE Transactions on Rehabilitation Engineering* 6, 4 (1998), 415–423. https://doi.org/10.1109/86.736156

[20] Yun-Lung Lin, Ting-Fang Wu, Ming-Chung Chen, Yao-Ming Yeh, and Hwa-Pey Wang. 2008. Designing a Scanning On-Screen Keyboard for People with Severe Motor Disabilities. In *Computers Helping People with Special Needs*, Klaus Miesenberger, Joachim Klaus, Wolfgang Zagler, and Arthur Karshmer (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 1184–1187. https://doi.org/10.1007/978-3-540-70540-6_178

[21] Mateus M Luna, Hugo AD Nascimento, Aaron Quigley, and Fabrizzio Soares. 2022. Text entry for the Blind on Smartwatches: A study of Braille code input methods for a novel device. *Universal Access in the Information Society* (2022), 1–19. https://doi.org/10.1007/s10209-022-00870-2

[22] I. Scott Mackenzie and Torsten Felzer. 2010. SAK: Scanning Ambiguous Keyboard for Efficient One-Key Text Entry. *ACM Trans. Comput.-Hum. Interact.* 17, 3, Article 11 (jul 2010), 39 pages. https://doi.org/10.1145/1806923.1806925

[23] Sergio Mascetti, Cristian Bernareggi, and Matteo Belotti. 2012. TypeInBraille: Quick Eyes-Free Typing on Smartphones. In *Computers Helping People with Special Needs*, Klaus Miesenberger, Arthur Karshmer, Petr Penaz, and Wolfgang Zagler (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 615–622. https://doi.org/10.1007/978-3-642-31534-3_90

[24] Katsumi Minakata, John Paulin Hansen, I. Scott MacKenzie, Per Bækgaard, and Vijay Rajanna. 2019. Pointing by Gaze, Head, and Foot in a Head-Mounted Display. In *Proceedings of the 11th ACM Symposium on Eye Tracking Research & Applications* (Denver, Colorado) *(ETRA '19)*. Association for Computing Machinery, New York, NY, USA, Article 69, 9 pages. https://doi.org/10.1145/3317956.3318150

[25] Joao Oliveira, Tiago Guerreiro, Hugo Nicolau, Joaquim Jorge, and Daniel Gonçalves. 2011. 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* (Lisbon, Portugal) *(INTERACT'11)*. Springer-Verlag, Berlin, Heidelberg, 100–107. https://doi.org/10.1007/978-3-642-23774-4_10

[26] Ryan Qin, Suwen Zhu, Yu-Hao Lin, Yu-Jung Ko, and Xiaojun Bi. 2018. Optimal-T9: An Optimized T9-like Keyboard for Small Touchscreen Devices. In *Proceedings of the 2018 ACM International Conference on Interactive Surfaces and Spaces* (Tokyo, Japan) *(ISS '18)*. Association for Computing Machinery, New York, NY, USA, 137–146. https://doi.org/10.1145/3279778.3279786

[27] Sayan Sarcar, Prateek Panwar, and Tuhin Chakraborty. 2013. EyeK: An Efficient Dwell-Free Eye Gaze-Based Text Entry System. In *Proceedings of the 11th Asia Pacific Conference on Computer Human Interaction* (Bangalore, India) *(APCHI '13)*. Association for Computing Machinery, New York, NY, USA, 215–220. https://doi.org/10.1145/2525194.2525288

[28] Caleb Southern, James Clawson, Brian Frey, Gregory Abowd, and Mario Romero. 2012. 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* (San Francisco, California, USA) *(MobileHCI '12)*. Association for Computing Machinery, New York, NY, USA, 317–326. https://doi.org/10.1145/2371574.2371623

[29] Hussain Tinwala and I. Scott MacKenzie. 2010. Eyes-Free Text Entry with Error Correction on Touchscreen Mobile Devices. In *Proceedings of the 6th Nordic Conference on Human-Computer Interaction: Extending Boundaries* (Reykjavik, Iceland) *(NordiCHI '10)*. Association for Computing Machinery, New York, NY, USA, 511–520. https://doi.org/10.1145/1868914.1868972

[30] Horabail S. Venkatagiri. 1999. Efficient keyboard layouts for sequential access in augmentative and alternative communication: AAC. *Augmentative and Alternative Communication* 15, 2 (06 1999), 126. https://doi.org/10.1080/07434619912331278625 Copyright - Copyright Decker Periodicals, Inc. Jun 1999; Last updated - 2022-11-16; CODEN - AAACEC.

[31] Keith Vertanen and Per Ola Kristensson. 2011. 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* (Stockholm, Sweden) *(MobileHCI '11)*. Association for Computing Machinery, New York, NY, USA, 295–298. https://doi.org/10.1145/2037373.2037418

[32] Keith Vertanen and Per Ola Kristensson. 2021. Mining, Analyzing, and Modeling Text Written on Mobile Devices. *Natural Language Engineering* 27 (2021), 1–33. https://doi.org/10.1017/S1351324919000548

[33] Keith Vertanen, Haythem Memmi, Justin Emge, Shyam Reyal, and Per Ola Kristensson. 2015. 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* (Seoul, Korea). Association for Computing Machinery, New York, NY, USA, 659–668. https://doi.org/10.1145/2702123.2702135

[34] Zheer Xu, Pui Chung Wong, Jun Gong, Te-Yen Wu, Aditya Shekhar Nittala, Xiaojun Bi, Jürgen Steimle, Hongbo Fu, Kening Zhu, and Xing-Dong Yang. 2019. TipText: Eyes-Free Text Entry on a Fingertip Keyboard. In *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology* (New Orleans, LA, USA) *(UIST '19)*. Association for Computing Machinery, New York, NY, USA, 883–899. https://doi.org/10.1145/3332165.3347865