Voice and Touch Based Error-tolerant Multimodal Text Editing and Correction for Smartphones

Maozheng Zhao
Department of Computer Science,
Stony Brook University
Stony Brook, NY, USA
mazhao@cs.stonybrook.edu

Wenzhe Cui
Department of Computer Science,
Stony Brook University
Stony Brook, NY, USA
wecui@cs.stonybrook.edu

I.V. Ramakrishnan
Department of Computer Science,
Stony Brook University
Stony Brook, NY, USA
ram@cs.stonybrook.edu

Shumin Zhai
Google LLC
Mountain View, California, USA
zhai@acm.org

Xiaojun Bi
Department of Computer Science,
Stony Brook University
Stony Brook, NY, USA
xiaojun@cs.stonybrook.edu

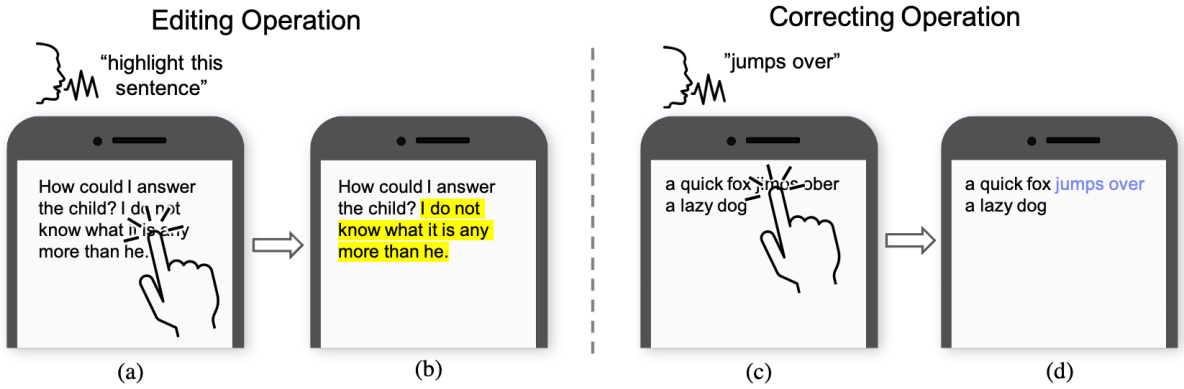

Figure 1: Demonstration of VT. (a): To edit a sentence, the user taps the sentence and speaks the editing command. (b) is the result of the editing operation. (c): To correct errors in a sentence, the user taps the position of the errors and speaks the new content for correction. (d) is the outcome of correction. The phrase "jimos ober" in the original sentence is corrected to "jumps over".

ABSTRACT

Editing operations such as cut, copy, paste, and correcting errors in typed text are often tedious and challenging to perform on smartphones. In this paper, we present VT, a voice and touch-based multi-modal text editing and correction method for smartphones. To edit text with VT, the user glides over a text fragment with a finger and dictates a command, such as "bold" to change the format of the fragment, or the user can tap inside a text area and speak a command such as "highlight this paragraph" to edit the text. For text correcting, the user taps approximately at the area of erroneous text fragment and dictates the new content for substitution or insertion. VT combines touch and voice inputs with language context such as language model and phrase similarity to infer a user's editing intention, which can handle ambiguities and noisy input signals. It is a great advantage over the existing error correction methods (e.g., iOS's Voice Control) which require precise cursor control or text selection. Our evaluation shows that VT significantly improves the efficiency of text editing and text correcting on smartphones over the touch-only method and the iOS's Voice Control method. Our user studies showed that VT reduced the text editing time by 30.80%, and text correcting time by 29.97% over the touch-only method. VT reduced the text editing time by 30.81%, and text correcting time by 47.96% over the iOS's Voice Control method.

UIST '21, October 10–14, 2021, Virtual Event, USA

ACM ISBN 978-1-4503-8635-7/21/10.
https://doi.org/10.1145/3472749.3474742

CCS CONCEPTS

• **Human-centered computing** → **Human computer interaction (HCI)**; *Interaction techniques.*

KEYWORDS

Multimodal interaction; text editing; text correction; touch input; smartphones.

ACM Reference Format:
Maozheng Zhao, Wenzhe Cui, I.V. Ramakrishnan, Shumin Zhai, and Xiaojun Bi. 2021. Voice and Touch Based Error-tolerant Multimodal Text Editing and Correction for Smartphones. In *The 34th Annual ACM Symposium on User Interface Software and Technology (UIST '21), October 10–14, 2021, Virtual Event, USA.* ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3472749.3474742

1 INTRODUCTION

Changing text input such as correcting errors and editing text is a core activity we perform daily on smartphones. Although such an activity is essential for messaging, emailing, searching, and social networking applications, it however is difficult to perform. The bottleneck lies in the need for precise and repetitive *manual* control. For example, the *de facto* cursor-based text correction technique requires accurately positioning the cursor at the error text, repeatedly pressing backspace to delete errors, and re-positioning the cursor back at its original location. Besides finger touch, voice input is another input modality we can leverage for editing text. However, voice input is prone to speech recognition errors, especially in noisy environments. It also is cumbersome to use voice for correcting errors in spoken text, mainly because voice input is unsuitable for specifying the location of the error [26, 42].

Both touch and voice input modalities have their respective strengths and weaknesses. The current text editing techniques fail to synergistically combine the strengths of both modalities to overcome the limitations of each modality. For example, although iOS's Voice Control [21] allows a user to use voice and touch to correct an erroneous word in a text, the process is laborious: the user needs to *precisely* select the word with touch, and speak out the *exact* content for correction. There is little room for human imprecision, which makes it difficult for the error-prone voice and touch input. Potentially, the performance and experience of text editing can be improved by coordinated use of speech and touch together with an underlying intelligent inference model that can predict the user's interaction intent. For example, if a user uses finger touch to point at the approximate location of an erroneous word and simultaneously speaks out the replacement word; the inference model should be able to determine the exact location of the erroneous word and replace it with the spoken word, thereby significantly reducing the user's overall interaction effort.

In this paper we research and develop voice and touch input based multimodal text editing technique, called VT. To edit the text, the user glides the finger over the text on the touchscreen, and speaks out the editing command (e.g., "cut", "copy", or "bold"), or the user can just tap the target text then speaks a command such as "bold the paragraph" to edit the text. VT combines the input signals from both voice and touch input to infer a user's editing intention, and then form and execute the corresponding editing operation. To correct errors such as inserting missing words or correcting words, the user points to the text to be corrected and speaks out the new text content. We developed a set of methods and algorithms in VT that can infer a user's intention of correction by combing the voice input, touch input, and text context, and then directly execute the most likely correction operation. Our evaluation showed that VT significantly improved the efficiency of text editing and text correcting over the touch-only method and the iOS's Voice Control method: VT reduced the text editing time by 30.80%, and text correcting time by 29.97% over the touch-only method, and reduced the text editing time by 30.81%, and text correcting time by 47.96% over iOS's Voice Control. Overall, VT mitigates the weaknesses of touch and voice input AND leverage their strengths, hence improving the efficiency of text editing on mobile devices.

2 RELATED WORK

As background of the current work, we review previous techniques for text editing, text correction, and multi-modal interactions.

2.1 Techniques for Editing and Correcting Text on Mobile Devices.

Text editing and correction is an essential part of the heavy process of text entry on mobile devices [28, 43]. The cursor-based method was widely studied in previous research. iPhone Keyboard [3] supports magnifying lens and hard-press cursor touchpad. Hackerskeyboard [52] allows users to control the cursor with arrow keys. Previous work also adapted gestural methods for cursor positioning. For example, previous research explored cursor moving by horizontal gestures [15], "scroll ring" with four-direction gesture with swipe [55], swiping left or right from "space" [22], including both taps and gestures to be drawn on the top of the soft keyboard to move the cursor, select text, and use the clipboard [16].

Along with the cursor-based method, researchers have explored a number of facilitative methods for text selection. On Android devices, users could select a word with multiple different operations, including sliding finger, double-tap or long-press. Gestural methods such as clock-wise gesture [22] and two-finger gesture [15] have also been implemented in text selection. Gaze'N'Touch [44] explored gaze-based interaction on text selection.

For editing operations on selected text, current keyboards such as iPhone keyboard [3] and google keyboard generally work with a pop-up widget(menu) providing possible editing actions. There are also gesture-based command [2, 8, 16, 30] have been explored on the selected text.

Challenges of cursor-based editing and correction on mobile devices often come from small screen sizes and the fat finger problem [5, 20, 51]. Intelligent interaction techniques such as auto-correction were introduced to address these challenges. In modern input methods on smartphones, auto-correction is wildly implemented. It automatically corrects the word currently being entered [6, 17, 50]. However, the limitation of auto-correction techniques lies in the current input word which indicates that it is not suitable for editing the text that has already been entered. Arif et al. added intelligent sliding functions to the backspace in the Smart-Restorable Backspace [4]. This technique can predict correction position and restore the previously deleted text. It helps to reduce the operations on deleting and positioning operations in word correction. WiseType [1] introduced novel visualization to highlight errors to assist error correction. Besides, grammar checking methods on entered text such as Gboard [33] and Grammarly [23] support correcting words and revising text by providing possible text on the suggestion bar. However, these approaches offer operations without considering the user's correction intention.

Thus the results can be irrelevant. VT adopts a user-guided approach to perform more goal-oriented operations. Users can point at a location for error correction or dictate more context words around the error words for better correction.

The "Type, then Correct" technique [54] and "JustCorrect" [9] are recent techniques on reducing cursor operations by injecting intelligence into the post hoc text correction process. Those methods are only for correcting a single word at a time, while VT is able to correct a whole phrase at once. And those methods are mainly for the scenario where there is only one input sentence, or correcting the last input sentence by typing on a keyboard. VT can be applied to anywhere the user points to in arbitrary text without any typing.

There are also voice-based editing method, such as Dragon NatuallySpeaking and Voice Typing in Google Docs [19, 37]. Targeted at accessibility applications, these methods transcribe the user's dictated words into input texts or editing commands and allow users to dictate instead of typing on the text areas. However, these approaches do not support the multimodal interaction to combine the voice input with touch gesture operations. VT also has a larger scope than previous research systems such as WiseType [1], Gestures and Widgets [15], TouchTap [16], Gaze'N'Touch [44] and Gedit [55]. VT focuses on both text formatting and error correction, while those systems focus on only one aspect of text editing: Gestures and Widgets [15], TouchTap [16] and Gedit [55] were about text formatting only; Gaze'N'Touch [44] was about improving text selection; WiseType [1] introduced novel visualization to highlight errors to assist error correction. Additionally WiseType [1], Gestures and Widgets [15], TouchTap [16], and Gedit [55] involve only touch input, while VT integrates two modalities (touch and voice).

2.2 Multimodal Interaction Technologies on Smartphones

Prior research has shown benefits with multimodal interaction, such as being natural and and more error tolerant [29, 39, 40], flexible [41], i.e., letting users pick any input mode as needed, and accommodating people with different input capabilities [13]. Previous research has also shown improved performance using pen marks and handwriting to correct speech recognition errors[49]. The presented research is particularly inspired by previous work on leveraging multiple input modalities to improve text entry performance. Modern soft keyboards (e.g., Gboard [33]) support entering text via touch and voice input. However, these two modalities are often used in isolation.

Researchers have also explored fusing information from multiple modalities to reduce text entry ambiguity, such as combining speech and gesture typing [38, 47], using finger touch to specify the word boundaries to improve speech recognition accuracy [46], or using unistrokes together with key landings [24] to improve input efficiency. In desktop computing, combining eye gaze with keyboard typing has been shown to be an effective approach to text editing [48].

Previous research also explored the performance of multimodal interaction both on single-app tasks [14, 25] and cross-app tasks [53]. The combination of voice and touch enhanced the experience on the mobile devices. Besides, multimodal method were also implemented to enhance the performance on disambiguation interfaces [32, 35, 45].

Although iOS's Voice Control [21] and Android Voice Access [18] enable users to edit and correct text with voice and touch input, these two methods are not error-tolerant as they require precise text selection, cursor manipulation, and precise text content for error correction. For example, in both iOS's Voice Control [21] and Android Voice Access [18], substituting a text segment with touch and voice requires the user to first precisely select the text segment with touch, and speak the exact new content to replace the selected text. In contrast VT is error-tolerant: VT can infer a user's error correction intention by combining language model, Word2Vec, and Levenshtein distance between input text and existing content to resolve ambiguity in input. A user only needs to approximately indicate the location of error, and speak the content which may or may not include words that already exist in the content. Additionally VT supports flexibly blending voice and touch input for text formatting: the touch input (e.g., gliding over the text) could occur before, during, or after the voice input. In contrast, iOS's Voice Control requires a user to select the text first and then issue the command. Touch and voice input must follow a strict order in iOS's Voice Control.

3 USE SCENARIO

The following use scenario illustrates the goal of this project.

Bob often writes and edit text on smartphones. However he is unsatisfied with the interaction experience. Touch input is inefficient for editing text as it requires precise control over the cursor positions. Speech input is difficult to specify the editing location. Neither of these two modalities meets his interaction needs. Existing voice and touch based multi-modal systems such as iOS's Voice Control is still not efficient enough because it still requires precise text selection before edit or correct the text. He had heard through the grapevine that VT combines voice and touch input modalities which could avoid the tedious and precise text selection process and is intelligent enough to infer the text span he intends to correct or edit. Bob got it installed on his smartphone. The scenario below illustrates how Bob utilizes the functionalities offered by VT.

It is a sunny Sunday afternoon and Bob is walking in the neighborhood. During his walk, Bob remembers that he should not forget to reply to the email his friend Jane had sent him yesterday. Bob opens the email apps and dictates "I will talk to you on Monday at 9 am." Bob, on a second thought, wishes to reschedule the meeting to Tuesday. Bob taps the word "Monday" and speaks "Tuesday". VT infers that Bob intends to change "Monday" to "Tuesday" given the language context, the caret location, and the spoken utterance. It then changes "Monday" to "Tuesday" in the email. Bob also wants to make "Tuesday at 9 am" bold as this meeting time is different from their usual meeting time. Bob then taps "Tueday" and speaks "bold four words". VT then change the sentence to "I will talk to you on **Tuesday at 9 am.**" Note that in this illustration, Bob combined voice and touch input to correct an error and edit text in email, without any precise text selection. VT has considerably simplified Bob's interaction with smartphone. He feels it has made him far more productive than before.

Next, we describe how VT supports multimodal text editing and correcting techniques.

4 MULTIMODAL TEXT EDITING AND CORRECTION TECHNIQUES

As shown in Figure 1, VT allows a user to combine voice and touch input to edit text such as highlighting, cutting, copying, and pasting text, or correcting errors such as correcting "jimos" to "jumps". It supports editing actions at word, sentence, and paragraph levels, and correction action at phrase level.

VT supported common touch input actions provided by the Android EditText view [11], such as tapping to reposition the caret position, or gliding the finger over a text segment to select it. The outcome of an touch input action is the new position of the caret, or the start and end positions of a selected text span.

VT supports voice input that starts before, during, and after those touch input actions. The user can signal the start of voice input by tapping or long clicking the text or tapping the microphone button.

Figure 2 shows the workflow of VT. A multimodal input event is represented as $X =< t, s >$, where t represents the outcome of a touch input, which could be the new caret position specified by tapping, or start and end positions of a text span specified by finger gliding, and s is the voice input.

After receiving $X =< t, s >$, VT first determines whether it is an editing or correcting operation as follows. It obtains top N (e.g., $N = 20$) voice recognition results $T_1, T_2, ..., T_N$ by feeding the voice input s into a speech recognizer (e.g., Android built-in voice recognizer). If the first word in any of T_i is one of the six reserved editing commands (copy, cut, paste, highlight, underline, and bold), VT considers it as an editing command. Otherwise, the voice inputs will be viewed as content for error correction. This limitation prevents VT to correct content starting with the six reserved word, which is common for current voice-based editing systems. For example, iOS's Voice Control reserves "copy that, cut that, bold that, etc." for command input. The editing and correction operations are supported separately, described in Section 4.1 Editing Text and Section 4.2 Correcting Text, respectively.

4.1 Editing Text

VT supports six classes of common editing operations: copy, cut, paste, highlight, underline, and bold.

4.1.1 Representation of an Editing Operation. VT first represents a text editing operation as a 2-tuple: $c =< r, w >$, where r is the operation name specified as a character string ("highlight", "bold", "underline", "cut", "copy", or "paste"), w is the location parameter specifying the text segment which the named operation will be applied to.

Here are two examples showing how an editing operation is defined by a 2-tuple: $c =< r, w >$. For example, to highlight the words "tomorrow at noon" in the sentence "The event will take place tomorrow at noon", the operation name r is "highlight", the location information w is the text segment "tomorrow at noon". Another example of operation is to cut the entire sentence "The event will take place tomorrow at noon.". For such an operation, the operation name r is "cut", the location w is the sentence.

Under this representation, the key of supporting multimodal text editing is to infer the intended text editing operation $c^* =< r, w >$ from the multimodal input $X =< t, s >$.

4.1.2 Creating and Executing Editing Operation. VT integrates the touch input t and voice input s in X to create and execute the intended text editing operation $c^* =< r, w >$ in the following steps, as specified in procedure *GetEditingOperation* (Algorithm 1).

One input to the algorithm is the touch input t, which could be the new caret position specified by tapping, or start and end positions of a text span specified by finger gliding.

The other input to the algorithm is T which is the voice recognition result that includes the command name *command*. T is determined as follows. VT examines the first word of T_i, which is one of the top N voice recognition results from the voice input s. If an editing command name *command* is found, we set $T = T_i$. If multiple voice recognition results include a command name of the first word, the one with the highest recognition score will be chosen as T.

The algorithm first set $r = command$, which is the operation name parameter of c^*. VT then integrates the touch input t and the voice recognition result T to form the location parameter w for c^*. VT supports two modes of forming w, namely Single Command and Compound Command mode:

- Single Command Mode: The user dictates a single editing command (e.g. bold), and selects the text with finger touch. The editing command will then be applied to the selected text. For example (Figure 3 left), to bold the phrase "tomorrow at noon", the user selects the phrase by pressing and gliding the finger from "tomorrow" to "noon", and dictates the command "bold" during the touch input. VT supports all the text selection action supported by Android EditText view, including double tapping or long pressing to select a word, or pressing and gliding to select a text segment.
- Compound Command Mode: A user combines voice and touch input to specify the text on which the spoken command will be applied. For example (Figure 3 right), a user taps to place the caret within a paragraph and says "bold this paragraph", to bold the entire paragraph where the caret resides, or places the caret to a sentence and says "highlight this sentence", to highlight the entire sentence.

Whether an editing operation is in single or compound command mode depends on whether the voice recognition result T includes the words specifying the scope of the operation including numbers and scoping word which is one of the words in the set <"word(s)", "sentence(s)", "paragraph(s)">. If no such a word is found, the editing is in the single command mode, otherwise in the compound command mode.

In the single command mode, the selected texts by touch input t is the location parameter w. In the compound command mode, w is determined by combining the location information specified by touch and scope information specified by voice input. For example, if the voice input includes "two paragraphs" and the finger touch lands on a specific paragraph, the w will include the paragraph that the finger points to and the following paragraph. After c^* is created from $X =< t, s >$, VT will execute it. This algorithm to

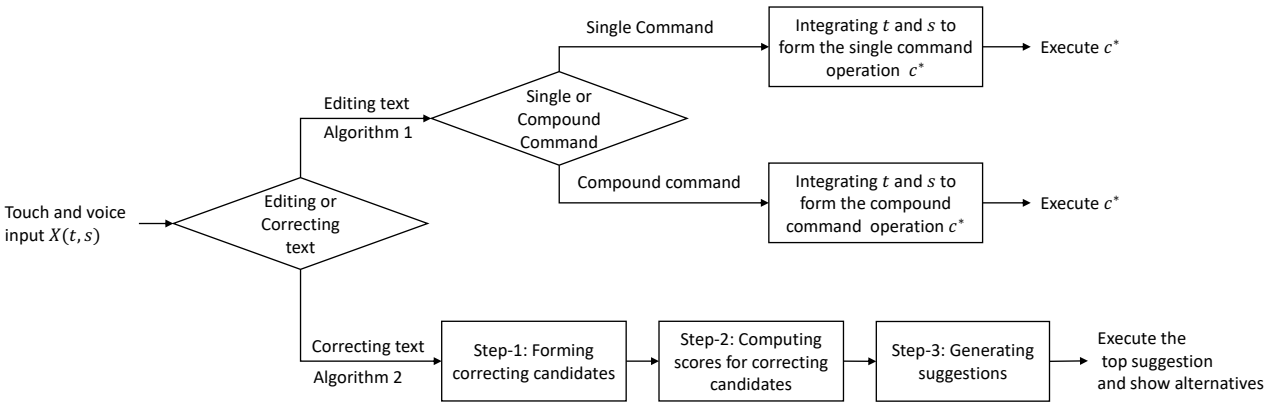

Figure 2: The workflow of VT. The multimodal input from a user is represented as $X =< t, s >$ where t is the touch input, and s is the voice input. The algorithms and components in this figure are explained later in this section.

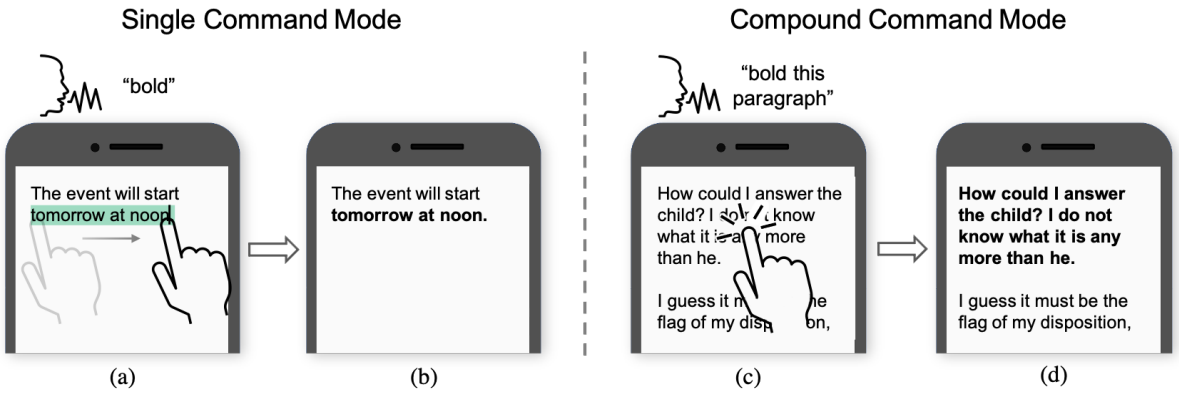

Figure 3: Examples for Single Command (left) and Compound Command (right) modes. (a): in the Single Command example, the user is selecting the text "tomorrow at noon" by gliding the finger and dictating "bold" at the same time. (b): the selected text becomes bold. (c) in the Compound Command example, a user taps the first paragraph and dictates "bold this paragraph". (d): the paragraph becomes bold.

get the editing operation $c^* =< r, w >$ is formally summarized in Algorithm 1.

Our techniques support a user to speak the editing command before, during or after the finger touch interaction, to accommodate different collaboration patterns between voice and touch input. The speech recognition is turned on as soon as a touch event occurs. Should a user want to speak a command before landing the finger on the text, she can click the voice input button on the screen. The speech recognition will wait for 5 seconds for the user to start the dictation, the duration of the dictation is up to 2 minutes.

4.2 Correcting Text

If a multimodal input event X is determined as a correction operation (i.e., none of the voice recognition result includes an editing command as the first word), VT will follow the procedure described in this section to correct text.

VT supports correcting existing text with touch and voice input, such as inserting missing words or replacing wrong or inappropriate words with new content. The user first specifies the location where correction will occur with the input finger by either tapping the error location, or selecting the erroneous text, and then dictates the new text content to be inserted, or to substitute the erroneous text. As shown in the use scenario (Section 3), Bob taps the word "Monday" and says "Tuesday" to correct "Monday" to "Tuesday".

The user can also speak some context around the text to be corrected. For example, to correct the sentence "it waspada very nice" to "it was very nice", the user can touch "waspada" and say "was". But since "was" is a short word, without any context the speech recognition model may recognize it as "were". Instead of saying "was", the user can say "it was" or "it was very" or "was very

Algorithm 1 VT Editing Algorithm

procedure GET EDITING OPERATION $c^* =< r, w >$
input:
  $t \leftarrow$ touch input
  $T \leftarrow$ text recognized from voice input s
process:
  w : location parameters of the text to edit
  r : operation name
  a : scope of operation specified by speech
  $r \leftarrow$ search the operation name from T
  $a \leftarrow$ search the operation scope from T
  if operation scope a is found **then**
    $w \leftarrow$ get the location parameters by combining t and a
  else
    $w \leftarrow$ get the location parameters from t
  end if
output:
  $c^* =< r, w >$
end procedure

nice", etc. Adding context can help the speech recognition model to better recognize the speaking content, and VT would utilize the context to better locate the text to be corrected. Without adding context, VT would still infer user's correction intention.

VT enables text correction in three steps. In step-1, it takes the multimodal input $X =< t, s >$ and the sentence to be corrected L as input to generates 3 types of text correction candidates: insertion-only candidates I, substitution-only candidates S and insertion-and-substitution candidates IS. In step-2, it assigns sentence score *SenScore* or substitution score *SubScore* to each candidate based on the speech recognition confidence score, an n-gram language model, and the similarity between the new and existing text. The *SenScore* or *SubScore* indicates how likely a candidate is the intended correction operation the user will perform. In step-3, it generates top 3 suggestions based on *SenScore* and *SubScore* of correction candidates. The top suggestion is executed by default, and the second and third candidates are provided as alternatives to user.

In the rest of this section we give more algorithmic details of each of the three steps.

4.2.1 Step-1: Forming Correction Candidates. The objective of this step is to form correction candidates based on the multimodal input $X =< t, s >$ and the sentence to be corrected L.

Given multimodal input $X =< t, s >$ and the language context L, VT first obtains top N voice recognition results $T_1, T_2, ..., T_N$ from a voice recognizer, where N is the total number of voice recognition results. We chose $N = 20$ in the current implementation.

VT then uses each of the recognition results T_i to generate three types of correction candidates: insertion-only, substitution-only, and insertion-and-substitution correction candidates. The three types of correction candidates are described as follows.

Insertion-Only candidates. An insertion-only operation is an operation that inserts the voice recognition result T_i into a sentence. The location of the insertion depends on the the touch location t. To accommodate the imprecise touch operation, T_i could be inserted to the white space the t points to, or the space before and after the touched word. The jth insertion-only candidate of the ith speech recognition alternative T_i is defined as I_{ij}.

For example, assuming a user selects the word "yoybgade" in the sentence "when do yoybgade to be there", and says "do you have", the ith recognition result T_i is "do you have", although this example cannot be fixed by insertion-only candidates, VT would still generate two insertion-only candidates for T_i by inserting T_i in two possible locations indicated by the underlines: "when do_yoybgade_to be there" (also illustrated in Table 1).

Substitution-only Candidates. A substitution-only correction is an operation that substitutes existing words in a sentence with the voice recognition results T_i. Assuming T_i includes n words: $T_i =< w_1, w_2, ..., w_n >$, the touch input t selects the phrase PH in the sentence, PH includes m words. VT would use T_i to replace n consecutive words in the sentence, under the constraint that in the replaced words at least one word is adjacent to or overlaps with words in PH. Such a constraint ensures that the substitution happens at or adjacent to the location specified by finger. If the finger just taps on a word, the PH only includes the single word tapped by the finger. The jth substitution-only candidate of the ith speech recognition alternative T_i is defined as S_{ij}.

For example, assuming that a user selects the word "yoybgade" in the sentence "when do yoybgade to be there", the ith speech recognition result T_i is "do you have", VT would create substitution-only correction candidates for T_i as shown in Table 1.

Insertion-and-Substitution Candidates. An insertion-and-substitution correction is an operation that performs both insertion and substitution operations at the location specified by the touch input t. It works similar to the substitution-only operation. The only difference is that the number of words that are substituted in the original sentence is less than the number of words in voice recognition result T_i. The jth insertion-and-substitution candidate of the ith speech recognition alternative T_i is defined as IS_{ij}.

In the same example where the user selects the word "yoybgade" in the sentence "when do yoybgade to be there", the ith speech recognition result T_i was "do you have", VT would generate insertion-and-substitutin candidates for T_i as shown in Table 1. In Table 1, the candidates IS_{i1} to IS_{i4} are generated by replacing 2 words in the original sentence with "do you have", the candidates IS_{i5} to IS_{i7} are generated by replacing 1 word in the original sentence with "do you have".

4.2.2 Step-2: Computing Scores for Correcting Candidates. The objective of this step is to compute scores of correction candidates, which are I_{ij}, S_{ij} and IS_{ij} generated from Step 1 (examples are illustrated in Table 1). The score of a correction candidate represents how likely the candidate is the intended correction operation.

For each insertion-only candidate I_{ij}, VT computes the sentence score $SenScore_{ij}$ as follows:

$$SenScore(ij) = SC_i * LS_{ij}, \quad (1)$$

which is the product of speech recognition score SC_i for the ith speech recognition alternative T_i and language score LS_{ij}.

Table 1: Examples of correction candidates. This example assumes that the original sentence is "when do yoybgade to be there", and the user taps the word "yoybgade" and says "do you have". It also assumes that the speech recognition result T_i is "do you have". This table shows all the candidates generated for T_i. I_{ij} is the jth insertion-only candidate for T_i. S_{ij} is the jth substitution-only candidate T_i. IS_{ij} is the jth insertion-and-substitution candidate for T_i. The scores $SubScore_{ij}$, $SenScore_{ij}$, SC_i, PS_{ij} and LS_{ij} are defined in Section 4.2.2. IS_{i2} candidate is the intended correction result, its scores are bold.

Candidate Type	Correction Candidates	$SubScore_{ij}$	$SenScore_{ij}$	SC_i	LS_{ij}	PS_{ij}
Insertion-only	I_{i1} when do **do you have** yoybgade to be there	N/A	0.192	0.969	0.198	N/A
	I_{i2} when do yoybgade **do you have** to be there	N/A	0.353	0.969	0.364	N/A
Substitution-only	S_{i1} **do you have** to be there	0.139	0.969	0.969	1.000	0.144
	S_{i2} when **do you have** be there	0.229	0.687	0.969	0.709	0.333
	S_{i3} when do **do you have** there	0.121	0.652	0.969	0.673	0.186
	S_{i4} when do yoybgade **do you have**	0.106	0.490	0.969	0.505	0.217
Insertion-and-substitution	IS_{i1} **do you have** yoybgade to be there	0.000	0.529	0.969	0.546	0.000
	$\boldsymbol{IS_{i2}}$: when **do you have** to be there	**0.300**	**0.847**	**0.969**	**0.354**	**0.874**
	IS_{i3} when do **do you have** be there	0.010	0.472	0.969	0.487	0.021
	IS_{i4} when do yoybgade **do you have** there	0.121	0.652	0.969	0.385	0.213
	IS_{i5} when **do you have** yoybgade to be there	0.136	0.407	0.969	0.420	0.333
	IS_{i6} when do **do you have** to be there	0.026	0.632	0.969	0.652	0.042
	IS_{i7} when do yoybgade **do you have** be there	0.020	0.192	0.969	0.199	0.105

VT computes the $SubScore_{ij}$ for each substitution-only candidate S_{ij} or insertion-and-substitution candidate IS_{ij} as follows:

$$\begin{aligned} SubScore(ij) &= SenScore(ij) * PS_{ij} \\ &= SC_i * LS_{ij} * PS_{ij}, \end{aligned} \tag{2}$$

which is the product of speech recognition score SC_i, language score LS_{ij}, and phrase similarity score PS_{ij}.

The speech recognition score SC_i, language score LS_{ij}, and phrase similarity score PS_{ij} in Equations 1 and 2 are computed as follows.

Speech Recognition Score. The term SC_i is the speech recognition confidence of the spoken text T_i. In our current implementation, it is between 0 and 1 and generated by the Android built-in speech recognizer [12].

Language score. The language score LS_{ij} reflects how likely a candidate C_{ij} is a valid sentence. This score was computed similar to the "Sentence Channel" score described in the previous work [9]. More specifically, we trained a 3-gram language model using the KenLM Language Model over the Corpus of Contemporary American English (COCA) [10] (2012 to 2017), which contains over 500 million words. The fitted language model file was compiled into a binary file to accelerate processing.

This language model will take a candidate sentence C_{ij} as input, and outputs its estimated log probability $P(C_{ij})$. By normalizing $P(C_{ij})$ in the range of 0 to 1, we get the language score LS_{ij}:

$$LS_{ij} = \frac{P(C_{ij}) - min(P(C_{ij}))}{max(P(C_{ij})) - min(P(C_{ij}))} \tag{3}$$

where $min(C_{ij})$ and $max(C_{ij})$ are the minimum and maximum language scores among all the correction candidates.

Phrase similarity. The phrase similarity score PS_{ij} is defined for substitution-only, and insertion-and-substitution corrections candidates. It reflects how similar the substituted n-word phrase in the original sentence (denoted by P_j, which refers to the substituted n-word phrase of the j-th possible substitution candidate) is to the new m-word phrase, which is the voice recognition result T_i. The higher PS_{ij}, the more similar P_j is to T_i.

The phrase similarity score is computed as follows. For a word w_i in P_j ($i = 1, 2, ..., n$), we first find a matching word w'_i in T_i that has the highest similarity score. The similarity score of between two words w_i and w'_i, denoted by $Score(w_i, w'_i)$, is computed as:

$$Score(w_i, w'_i) = \frac{ES(w_i, w'_i) + WS(w_i, w'_i)}{2}. \tag{4}$$

The term $ES(w_i, w'_i)$ is obtained by dividing the Levenshtein [31] edit distance between w_i and w'_i with $max(L(w_i), L(w'_i))$, where $L(w_i)$ and $L(w'_i)$ are the length of w_i and w'_i in characters. This term $ES(w_i, w'_i)$ has a value between 0 and 1, reflecting how similar w_i is to w'_i in spelling. The term $WS(w_i, w'_i)$ is the cosine similarity between the word embeddings of w_i and w'_i, reflecting the semantic similarity between w_i and w'_i. Our word embedding model was learned over the "Text8" dataset [34] using the Word2Vec skip-gram approach [36].

The phrase similarity PS_{ij} is computed as the weighted average of word similarity score over all the words in the substituted phrase P_j:

$$PS = \sum_{i=1}^{n} \alpha_i * Score(w_i, w'_i), \tag{5}$$

where w_i is a word in P_j, w'_i is the word in T_i that has the highest similarity score with w_i, namely w'_i is the matching word for w_i in T_i, and α_i is a weight which reflects whether the position of w_i in the substituted phrase P_j is close to the position of w'_i in the new text content T_i.

The α_i is calculated as follows. We first use a range $[PStart(w_i), PEnd(w_i)]$ to represent the relative position of w_i in P_j. Both $PStart(w_i)$ and $PEnd(w_i)$ are numbers between 0 and 1. Assuming P_j has n words, the index of its words are from 0 to $n-1$, and

k is the index of the word w_i in P_j, we define $PStart(w_i) = k/n$ and $PEnd(w_i) = (k+1)/n$. Following the same method, we represent the position of w'_i in T_i which has m words as $[TStart(w'_i), TEnd(w'_i)]$. Assuming the length of the overlapped range between $[PStart(w_i), PEnd(w_i)]$ and $[TStart(w'_i), TEnd(w'_i)]$ is τ, we defined α_i as $\alpha_i = \tau \cdot Max(n, m)$, where $Max(n, m)$ is the maximum value of n and m.

Under this definition of α_i, if P_j and T_i has the same number of words (i.e., $n = m$), and the position of w_i in P_j is the same with the position of w'_i in T_i, we have $\alpha_i = 1$. If $\tau = 0$, which means the range $[PStart(w_i), PEnd(w_i)]$ and $[TStart(w'_i), TEnd(w'_i)]$ have no overlap, we have $\alpha_i = 0$. Such an α_i value ($\alpha_i = 0$) reflects the condition where the position of w_i is far different from the position of w'_i.

With the above equations (Equations 1-5), we can compute a $SenScore_{ij}$ or $SubScore_{ij}$ for each correction candidate. Table 1 show scores for some examples.

4.2.3 Step-3: Generating suggestions. The objective of this step is to order correction candidates by their scores and output top three candidates. VT orders the correction candidates as follows. It first merges the list S which contains all substitution-only candidates with the list IS which contains all insertion-and-substitution candidates to a merged list denoted by M, and sort the merged list M by *SubScore* in descending order. Second, VT sorts the list I which contains all insertion-only candidates by *SenScore* in descending order. Third, it compares the top elements in both the merged list M and sorted list I by candidates' *SenScore*, and pick the one with higher *SenScore* as the top 1 suggestion. It then removes the picked correction candidate from the corresponding sorted list and performs the comparison again to obtain 2nd, and 3rd suggestions. We used the *SenScore* to compare candidates between I and M because *SenScore* is common score between all three types candidates while insertion-only candidates do not have *Subscore*, so using *SenScore* can compare candidates in M and I with the same metric.

The outcome of this step is the top 3 suggestions for correcting operation. The top candidate is the default outcome and the 2nd and 3rd candidates are suggested as alternatives which can be selected by tapping it with finger touch.

The algorithm for VT text correcting is summarized in Algorithm 2.

Algorithm 2 VT correcting Algorithm

```
procedure GET CORRECTING SUGGESTIONS
input:
    t ← touch input
    s ← voice input
    L ← sentence to be corrected
process:
    I ← generating insertion-only candidates by t, s and L (e.g., Table 1)
    S ← generating substitution-only candidates by t, s and L (e.g., Table 1)
    IS ← generating insertion-and-substitution candidates by t, s and L (e.g., Table 1)
    compute sentence scores SenScore for each candidate in I, S and IS (Equation (1))
    compute substitution scores SubScore for each candidate in S and IS (Equation (2))
    SortedM ← merge S and IS , and sort the merged candidates by SubScore in descending order
    SortedI ← sort I by SenScore in descending order
    for i ← 0 to 2 do
        if SortedM[0]'s SenScore > SortedI[0]'s SenScore then
            Suggestions[i] ← SortedM[0]
            Remove SortedM[0] from SortedM
        else
            Suggestions[i] ← SortedI[0]
            Remove SortedI[0] from SortedI
        end if
    end for
output:
    Suggestions[0], Suggestions[1], Suggestions[2]
end procedure
```

5 EXPERIMENT 1: COMPARING VT WITH TOUCH-ONLY METHOD

In this experiment, we compared VT with the state-of-the-art touch-only method for text editing and text correction tasks. We implemented the multimodal text editing and correcting techniques on an Android smartphone (Google Pixel with Android 9) and conducted a user study to evaluate its performance. We used the Android built-in SpeechRecognizer class [12] for voice recognition.

5.1 Participants

We recruited 16 participants (five females) from 22 to 32 years old (Mean = 26.8, Std = 3.1). The self reported median familiarity (1: not familiar, 5: very familiar) with the existing touch-only text editing technique was 5. The participants were instructed to use their preferred hand posture throughout the study.

5.2 Apparatus

A Google Pixel device (Android version: 9, Processor: Qualcomm Snapdragon 821, GPU: Qualcomm Adreno 530, RAM: 4GB LPDDR4, Internal storage: 32GB) with a 5.0" display (AMOLED with 1080 × 1920 pixel resolution) was used for the experiment.

5.3 Design

The study was a within-subjects design. The independent variable was the text editing and correcting method, which has two levels: touch-only condition and VT condition.

- Touch-Only condition. The user used the existing touch-based method in Android OS to complete the task. More specifically, the user could manipulate the caret, and select text using the default touch gestures supported by Android Text View, such as tapping to reposition caret, double tapping to select a word, and pressing and gliding to select multiple words. After a text segment was selected, a floating menu with common text editing operations was displayed near the text as shown in Figure 4 and the user could execute an editing command via the menu. The floating menu design followed the design in Android text editing application such as Google Keep. The top-level menu includes common

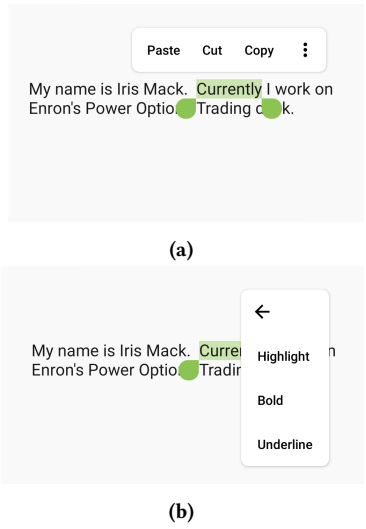

Figure 4: The floating menu for text editing task in the touch-only condition. (a) The menu is displayed after some text are selected. The "paste", "cut" and "copy" are displayed as default, extra menu items will be displayed after clicking the three dots at the end of the menu. (b) Extra menu items are shown after clicking the three dots in (a). Clicking the back arrow in the menu will go back to the menu in (a).

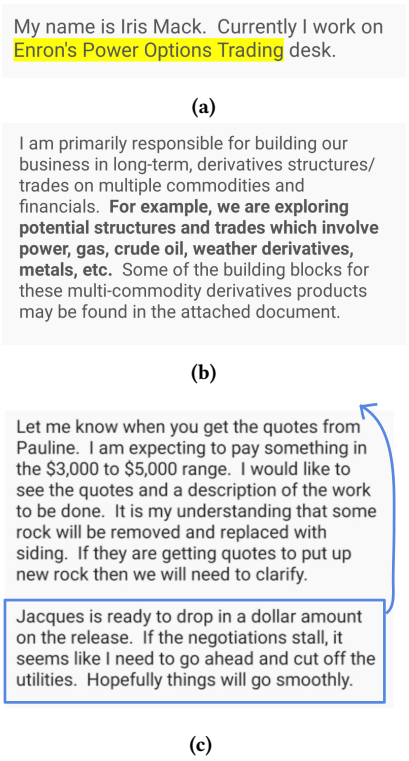

Figure 5: Examples of the editing tasks on different levels of text. Figure 5a is a task to highlight 4 words (word-level task). Figure 5b is a task to make a sentence bold (sentence-level task). Figure 5c is a task to cut and paste a paragraph, the paragraph in the box needs to be cut and pasted at the location pointed by the arrow.

operations such as "cut", "copy", and "paste"; tapping the three dot icon will reveal more operations including "highlight", "bold", and "underline". In this condition a user used the Google's Gboard to correct errors and type corrected content.

- VT condition. The participant used VT to edit text and correct errors. The touch-only method was kept as a fallback method. The user may choose to use touch-only method to finish the task if she failed on using VT. We kept touch-only method as a fallback method because VT is proposed to augment rather than replace the existing touch-only method.

The study included two tasks: Text editing task and text correction task, which are described in the next section. In the experiment, the order of the two tasks and the two conditions were counterbalanced across 16 users.

5.4 Tasks

5.4.1 Text Editing Task. There were 5 classes of editing tasks in total: cut & paste, copy & paste, highlight, bold, underline. The editing tasks were applied to 3 levels of texts: words, sentences, and paragraphs. In the experiment, there was one trial for a editing class × level combination, so there were $1 \times 3 \times 5 = 15$ trials in total. The orders of the 15 trials were randomized for each condition and each user. We created editing tasks on text chosen from the Enron Email Dataset [27], which contained a total of about 0.5M emails from about 150 users. Some editing tasks are shown in Figure 5 as examples.

In total, the experiment included 16 participants × 2 methods × 15 trials = 480 trials.

5.4.2 Text Correction task. Participants corrected text errors in this task. The sentences with errors were selected from Palin et al.'s mobile typing dataset [43]. This data set had entered text and their correct versions by 37,370 users on mobile phones. We focused on omission and substitution errors since the editing operation of VT was designed to handle these two types of errors. There were 28 testing sentences for this task, 5 have omission errors, 23 have substitution errors.

The difficulty of a correcting task was defined by the character-level edit distance between the target sentence and the sentence with errors. For the edit distance ranged from 1 to 6, we created at least three correcting trials for each edit distance. We also created 3 trials with edit distance larger than 6. Table 2 shows some example sentences used in the experiment. We excluded sentences with errors on numbers, names and acronyms due to the difficulty of recognizing these words with voice input. Each participant corrected the same set of sentences for each condition. The order of the sentences were randomized for different conditions and users.

In total, the experiment included 16 participants × 2 methods × 28 trails = 896 trials.

Table 2: Examples of correcting tasks in the experiment. The first sentence contains an omission error. The rest sentences contain substitution errors. The different words between the text to edit and the target text are underlined.

Text to edit	Target text
1. If not can call you	If not can I call you
2. What so you thinl	What do you think
3. Itu waspada very nice	It was very nice
4. The matter address tomorrow with Stan	We will address tomorrow with Stan

5.5 Procedure

In each trial, a presentation page was first displayed to explain the editing/correcting task. For example Figure 6(a) and Figure 6(c) show the presentation pages for an editing and correcting trial respectively. Clicking the "start" button started the trial. Once the task is accomplished, a "Success!" would show on the top right of the screen as shown in Figure 6(d), then clicking the "Next" button would start the next trial.

During an editing trial, a user could click the "Back" button located at the bottom-left of the screen to return to the presentation page to check the editing instruction, and then press the start button again to restart the trial. The "Undo" button on the bottom right would undo the last editing operation. In the VT condition, there would be a "microphone" button to start the speech recognition, in case a user wanted to start voice input before touch input.

Before the experiment, participants completed a warm up session to get familiar with VT condition and touch-only condition. They used VT and touch-only conditions to complete 10 warm-up correction trials separately, and used touch-only condition, VT single command mode, and VT compound command mode to complete 8 editing trials separately. In the experiment, participants were instructed to complete each trial (from the moment "Start" button being clicked to the moment "Success!" was shown) as fast as possible.

5.6 Results

5.6.1 Error Rate. Because participants were required to successfully complete a trial to move to next trial, there was no erroneous (or incomplete) trial left. The error rate was 0 for both VT and touch-only method in both editing and correction tasks.

5.6.2 Completion Time for Editing tasks. The task completion time was the main metric for evaluating the performance of each method. We referred to task completion time as "editing time" for an editing trial, which was defined as the duration from the moment the "start" button was clicked on the task presentation page to the moment that "Success!" was shown on the editing page. This metric measures users' operation time to accomplish the editing task.

The average editing time for all trials using the designated methods are shown on the left part of Figure 7. The mean ± 95% CI of the editing time was 10.28 ± 0.71 seconds for the touch-only method and 7.20 ± 0.44 seconds for the VT method. A paired-samples t-test indicates that the difference was statistically significant ($t_{15} = 5.36, p < 0.001$). VT reduced the average editing time by 30.80%.

To investigate the performance of VT on different levels of text, the average editing time for different text levels using the designated methods are shown in Figure 8. We can see that the editing time of touch-only method increases with levels while the editing time of VT method stays relative stable among levels. This is because there are more texts to be selected for the higher levels, touch-only method needs to select text by gliding and taps on the screen, while VT method can automatically select texts by compound voice commands. In the word level, the mean ± 95% CI of text editing time with touch-only method and VT method was 8.10 ± 1.10 and 7.02 ± 0.85 respectively. In sentence level, the mean ± 95% CI of text editing time with touch-only method and VT method was 10.80 ± 0.87 and 7.21 ± 0.74. In paragraph level, the mean ± 95% CI of text editing time with touch-only method and VT method was 11.94 ± 1.48 and 7.38 ± 0.67. In each level, the VT method performed faster than the touch-only method. Pairwise comparisons with Bonferroni correction showed that differences were statistically significant in sentence level and paragraph level ($p < 0.001$) and not significant at word level ($p = 0.2076$).

To investigate how VT and touch-only method complement each other in the VT condition, the percentage of different methods used per level are counted, as shown in Figure 10. We can see that for all trials (the first bar) users chose to use VT method for more than 90% of the operations. For VT method, the compound commands are more frequently used than single commands. For words level editing, the touch-only method is used more frequently than sentences level or paragraphs level, this is because VT saves more time for higher levels.

VT's single commands support flexibly blending voice and touch input for text editing. The touch input (e.g., gliding over the text) could occur before, during, or after the voice input. For all the single commands used in this user study, 4.7% of touch inputs occur prior to the voice input, 91.6% of touch inputs occur during the voice input, and 3.7% of touch inputs occur after the voice input. Users prefer issuing the voice command during the touch input.

The editing time for trials using the touch-only method and VT method in different editing tasks are shown in Figure 9. In each task, the editing time of the VT method was lower than the touch-only method.

5.6.3 Method Usage Pattern in Text Editing Task. Since touch-only method was kept as a fallback method in the VT condition, we examined the percentage of using this method. The percentage of using VT-single-command, VT-compound-command, and touch-only method to edit text in the VT condition is shown in Figure 11. We can see that for each operation VT was chosen for more than 80% of the operations. The "Paste" did not have VT compound

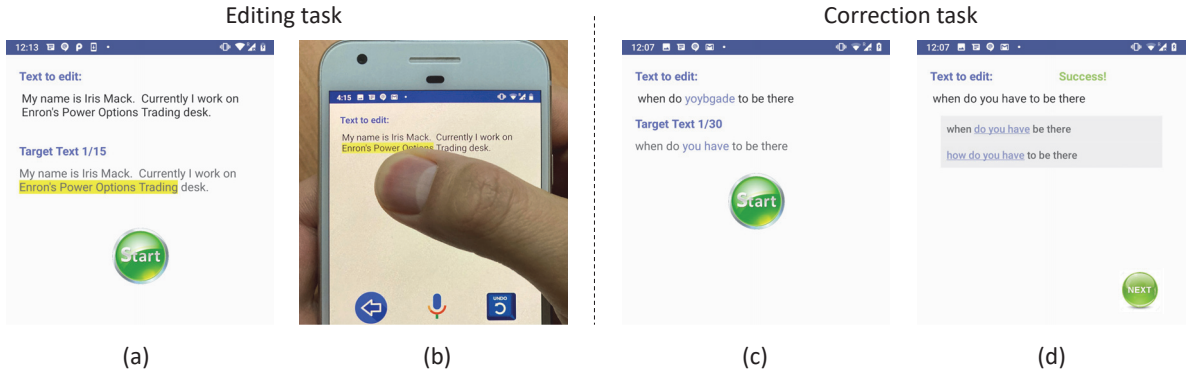

Figure 6: The procedures of the experiment, (a) and (b) are for the text editing task, (c) and (d) are for text correction task. (a): the presentation page for a text editing task. (b): a participant is highlighting the text by voice command and gliding. (c): the task presentation page of a text correction task. (d): outcome of using VT method to correct the error. Two alternatives are shown under the editing text. The user can choose a suggestion by tapping it.

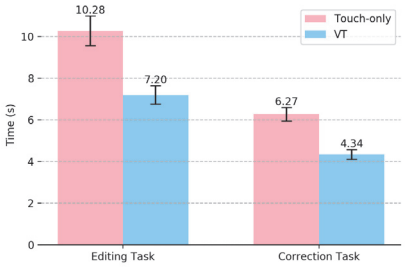

Figure 7: The mean (95% CI) of completion time by task type × method.

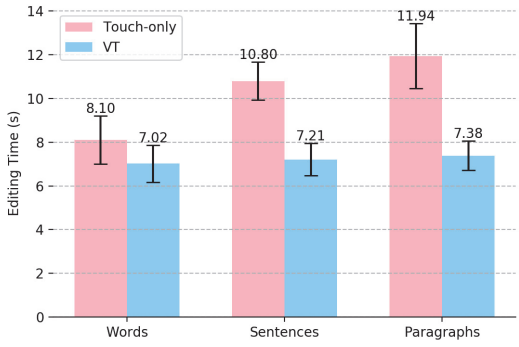

Figure 8: The mean (95% CI) of completion time by text level × method in text editing task.

commands by design, still its VT single command was chosen for nearly all the operations.

5.6.4 Completion Time for Correction tasks. For correction tasks, the "correcting time" for each trial was defined as the duration from the moment the "start" button in the task presentation page was clicked to the moment that "Success!" was shown on the editing page. This metric measures users' operation time to correct the errors.

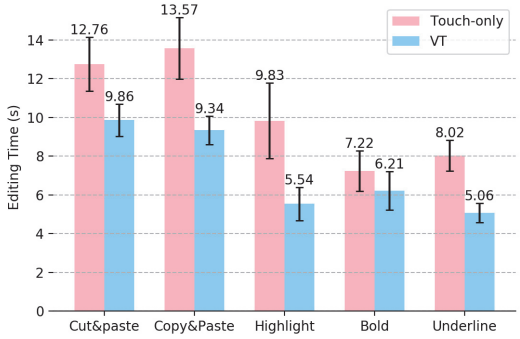

Figure 9: The mean (95% CI) of completion time by editing task type × method.

The average correcting time for all trials using the designated method is shown on the right part of Figure 7. The mean ± 95% CI of the correcting time was 6.27 ± 0.33 seconds for the touch-only method and 4.34±0.23 seconds for the VT method. A paired-samples t-test indicates that the difference was statistically significant ($t_{15} = 7.47, p < 0.001$). VT methods reduced the average correcting time by 29.97%.

To understand the effectiveness of the methods, we grouped all the correcting trials by edit distance between the target sentence and the incorrect sentence. The average text correcting time for different edit distances with the two methods are shown in Figure 12. When the edit distance is 1, the correcting time with VT method did not show better performance than touch-only method. The VT method were faster than the touch-only baseline for the rest edit distances.

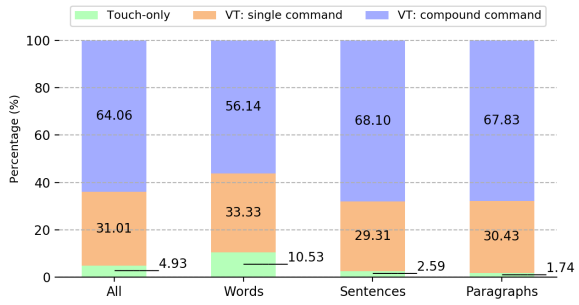

Figure 10: In the VT condition, the percentage of using VT-single-command, VT-compound-command, and touch-only method to edit text by words, sentences and paragraphs level tasks in the text editing task. Note that in the VT condition, touch-only method was a fallback method.

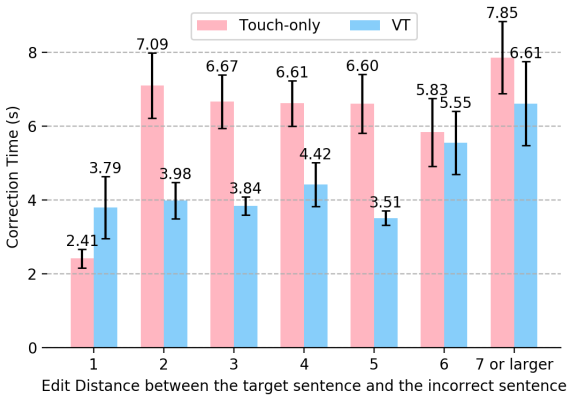

Figure 12: The mean (95% CI) of completion time by edit distance × method.

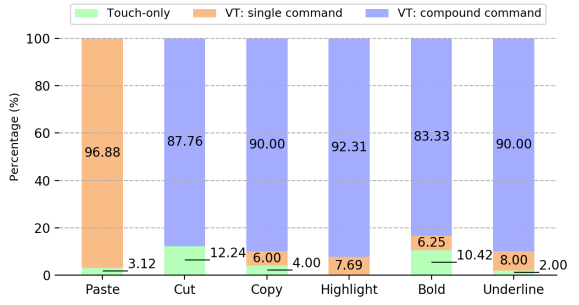

Figure 11: In the VT condition, the percentage of using VT-single-command, VT-compound-command, and touch-only method to edit text by task type, in the text editing task.

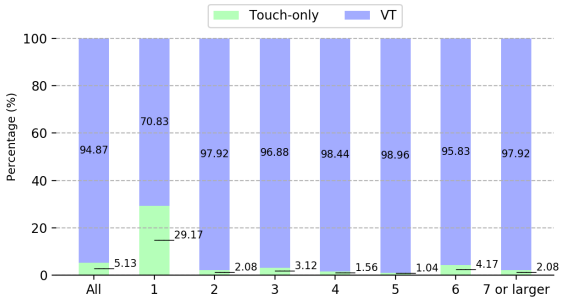

Figure 13: In the VT condition, the percentage of using VT and touch-only method to correct errors by edit distance in the text correction task.

5.6.5 Method Usage Pattern in Text Correction Task. Figure 13 shows the percentage of different method used per edit distance for VT condition. We can see that VT was chosen by users for more than 90% of all trials (the first bar). For edit distance at 1, more users chose to use the touch-only method than other edit distances. It is understandable as Figure 12 shows that touch-only method was faster than VT when edit distance is 1, which means only one character is wrong in the sentence.

5.6.6 Subjective feedback. At the end of the experiment , we asked participants their preferred method (VT, touch-only method or No preference) for the two kinds of tasks. For the editing tasks, 16 out of 16 participants preferred VT method. For the correcting tasks, 15 out of 16 participants preferred VT, 1 participant prefer touch-only method.

We also asked the participants to provide a numerical rating (1: least demanding, 10: most demanding) on mental and physical demand for each method and each kind of tasks. Mental demand describes how much mental effort is required. Physical demand describes how much physical effort is required. The medians of subjective ratings are shown in Figure 14. For the editing tasks, the subjective ratings were in favor of VT for both mental and physical demands. Wilcoxon Signed-Ranks Tests indicated that the subjective mental ($p < 0.001$) and physical ($p < 0.001$) demands of the VT method were significantly lower than those of the touch-only method. In correction tasks, the subjective ratings were also in favor of VT for both demands. Wilcoxon Signed-Ranks Tests indicated that the subjective mental ($p < 0.001$) and physical ($p < 0.001$) demands of the VT method were significantly lower than those of the touch-only method.

6 EXPERIMENT 2: COMPARING VT WITH IOS'S VOICE CONTROL

In this experiment, we compared VT with iOS's Voice Control for voice and touch based multimodal text editing and correction.

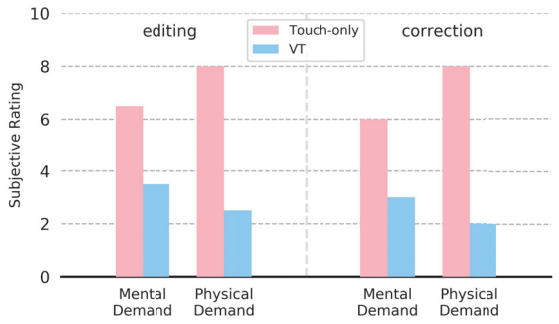

Figure 14: Median of subjective ratings. Lower rating means lower mental or physical demand.

6.1 Participants

We recruited 14 participants (two females) from 23 to 32 years old (Mean = 26.4, Std = 2.5). 10 of them were Android phones users and 4 of them were iPhone users. Although none of them had experience of using iOS's Voice Control for text editing and correction, each of them went through a 25-minute long training about iOS's Voice Control prior to the experiment, as explained in Section 6.5 Procedure.

6.2 Apparatus

We used an iPhone SE device (2nd generation, iOS version: 14.6, Processor: A13 Bionic chip, RAM: 3GB, Internal storage: 64GB) with a 4.7" display (LCD with 1334 × 750 pixel resolution), and a Google Pixel phone which was the same as the phone used in Section 5.

6.3 Design

The study was a within-subjects design. The independent variable was the text editing and correcting method, which has two levels: VT and iOS's Voice Control conditions.

- VT condition. The participants used VT to correct and edit text on an Android phone.
- iOS's Voice Control condition. Users used iOS's Voice Control to edit text and correct errors on an iPhone.

Because the objective of this experiment is to compare the voice and touch based multimodal methods, participants were not allowed to use keyboards to enter text and the floating menus for editing, in either VT or iOS's Voice Control.

The study included two tasks: Text editing task and text correction task. In the experiment, the order of the two tasks and the two conditions were counterbalanced across 14 users.

6.4 Tasks

6.4.1 Text Editing task. The text editing tasks in this study were the same as Section 5, except that 3 tasks using the "highlight" command were not included. Because iOS's Voice Control does not have a voice command to highlight text.

6.4.2 Text Correction task. The text correction tasks in this study are the same as Section 5.

6.5 Procedure

The procedures for VT condition are the same as Section 5. The experiment APP for the VT condition was in Android system. For iOS's Voice Control condition, we made an experiment APP in iOS system to replicate the same experiment procedure as the VT condition. The procedures are the same as the VT condition, except that there is no microphone button in the iOS APP because iOS's Voice Control keeps recognising speech all the time.

Before the experiment, participants are thoroughly instructed about how to correct and edit text under each condition. For the iOS's Voice Control condition, participants were demonstrated about the touch gestures in iOS for text selection, such as double tap to select a word, triple tap to select a paragraph, etc. And they were demonstrated about the voice commands in iOS's Voice Control, such as "replace {phrase} with {phrase}", "insert {phrase} before/after {phrase}", etc. Then participants completed a warm up session to get familiar with VT and iOS's Voice Control. They used each method to complete 10 warm-up correction trials and 8 warm-up editing trials.

6.6 Results

6.6.1 Error Rate. Similar to Section 5, the error rate was 0 for both VT and iOS's Voice Control in both editing and correction tasks, because participants must successfully complete a trial to advance to the next one.

6.6.2 Completion Time for Editing tasks. The average completion times for editing trials using iOS's Voice Control and VT are shown in Figure 15. The mean ± 95% CI of the editing time was 13.13 ± 1.17 seconds for the iOS's Voice Control and 9.09 ± 0.83 seconds for the VT method. A paired-samples t-test indicated that the difference was statistically significant ($t_{13} = 7.05, p < 0.001$). VT reduced the average editing time by 30.81%.

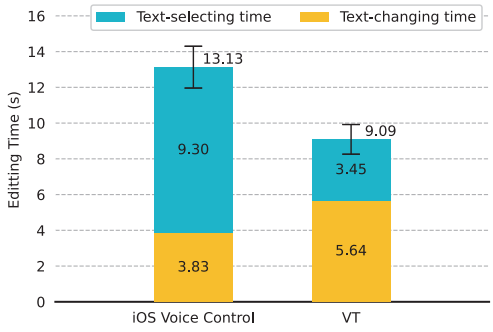

Figure 15: Mean (95% CI) of completion times for editing tasks for iOS's Voice Control and VT. The mean text-selecting time and mean text-changing time are shown in different colors.

To investigate where the improvement of VT came from, we divided the task completion time into two parts, text-selecting time and text-changing time. The text-selecting time is the time to select text or move the caret. The text-changing time is the time to change

the text after the text was selected or the caret was moved. The text-selecting time and text-changing time for iOS's Voice Control and VT are shown in Figure 15.

VT significantly reduced text-selecting time over iOS's Voice Control for editing tasks. The mean ± 95% CI of the text-selecting time was 9.30 ± 1.00 seconds for the iOS's Voice Control method and 3.45 ± 0.50 seconds for the VT method. A paired-samples t-test indicated that the difference was statistically significant ($t_{13} = 8.86, p < 0.001$). VT methods reduced the average text-selecting time by 62.90%.

The mean ± 95% CI of the text-changing time was 3.83 ± 0.29 seconds for the iOS's Voice Control method and 5.64 ± 0.46 seconds for the VT method. A paired-samples t-test indicated that the difference was statistically significant ($t_{13} = 3.88, p < 0.002$).

6.6.3 Completion Time for Correction tasks. The average completion times for correction trials using iOS's Voice Control and VT are shown in Figure 16. The mean ± 95% CI of the correcting time was 9.80 ± 1.52 seconds for the iOS's Voice Control and 5.10 ± 0.44 seconds for the VT method. A paired-samples t-test indicated that the difference was statistically significant ($t_{13} = 4.62, p < 0.001$). VT methods reduced the average correcting time by 47.96%.

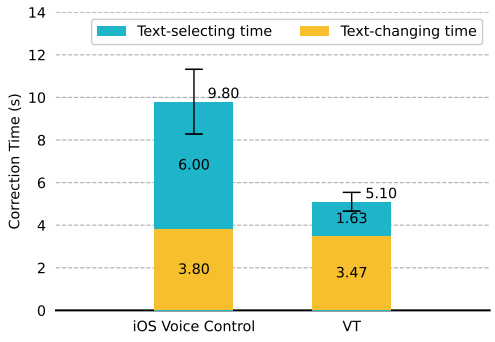

Figure 16: Mean (95% CI) of completion times for correction tasks for iOS's Voice Control and VT. The mean text-selecting time and mean text-changing time are shown in different colors.

The text-selecting time and text-changing time for iOS's Voice Control and VT for the text correcting tasks are shown in Figure 16. VT significantly reduced the text-selecting time over iOS's Voice Control for correcting tasks. The mean ± 95% CI of the text-selecting time was 6.00 ± 1.32 seconds for the iOS's Voice Control method and 1.63 ± 0.27 seconds for the VT method. A paired-samples t-test indicated that the difference was statistically significant ($t_{13} = 5.75, p < 0.001$). VT methods reduced the average text-selecting time by 72.83%. The difference between the two methods' text-changing times were not significant for correcting tasks. The mean ± 95% CI of the text-changing time was 3.80 ± 0.40 seconds for the iOS's Voice Control method and 3.47 ± 0.25 seconds for the VT method. A paired-samples t-test indicated that the difference was not statistically significant ($t_{13} = 1.07, p = 0.30$).

6.6.4 Subjective feedback. At the end of the experiment, we asked participants to rate each method on a scale of 1 to 5 (1: least preferred, 5: most preferred) for each task. The medians of subjective ratings are shown in Figure 17. For editing tasks, the median rating for VT and iOS's Voice Control were 5 and 3. A Wilcoxon Signed-Ranks Test indicated that the subjective ratings of VT was significantly higher than that of iOS's Voice Control (Z = 2.4809, p = 0.01314). For correction tasks, the median rating for VT and iOS's Voice Control were 5 and 3. A Wilcoxon Signed-Ranks Test indicated that the subjective ratings of VT was significantly higher than that of iOS's Voice Control (Z = 3.0594, p = 0.0022).

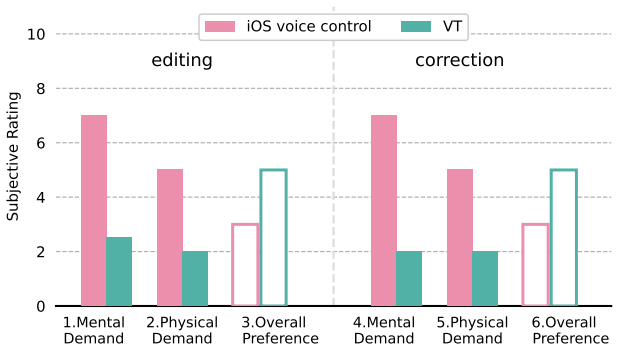

Figure 17: Median of subjective ratings for text editing and correction using VT and iOS's Voice Control. For measure 1, 2, 4 and 5, a lower rating means lower mental and physical demand. For measure 3 and 6 (1: least, 5: most preferred), a higher score means the method is more preferred. VT received favorable ratings in all categories.

We also asked the participants to provide a numerical rating (1: least demanding, 10: most demanding) on mental and physical demand for each method and each task. The medians of subjective ratings are shown in Figure 17. For each task, the subjective ratings were in favor of VT for both mental and physical demands.

7 GENERAL DISCUSSION

VT as a research project developed and studied novel text editing methods that synergistically combine touch and voice. Touch, being intuitive and direct at expressing spatial information, has been central to modern mobile computing but suffers from weaknesses such as imprecision. Speech is fundamentally fast (typical broadcasting speed is around 200 words per minute) but suffers from weaknesses such as expressing precise spatial information. Progress in deep learning has made speech recognition increasingly accurate and practical [7]. Our insight guiding the design of VT was to leverage the respective strength of touch and voice and avoid their respective weakness. In particular VT uses touch gestures to approximately indicate the editing or correction scope and uses voice to articulate the corresponding command or replacement text. Furthermore, by using the speech recognition APIs pre-installed on Android Pixel Phones and developing a set of relatively simple algorithms against the text editing task space, VT were able to infer the user's intents

from natural speech and touch input that is imprecise in timing (in relation to the voice utterance) and space (the text selection boundary can be fuzzy).

In our studies involving a set of common editing tasks and an error correction task, VT demonstrated marked improvements over the state-of-the-art touch-only baseline and the iOS's Voice Control baseline. In the study comparing VT with the touch-only method, VT reduced the time for editing tasks by 30.80% and the time for error correcting tasks by 29.96%. The users' subjective preference was also overwhelmingly favorable toward VT than the conventional touch-only method. For editing tasks, the improved efficiency was mainly attributed to automatic text selection by VT. For error correcting tasks, the improved efficiency was mainly attributed to reduction of typing actions and moving the cursor by touch.

In the study comparing VT with the iOS's Voice Control method, VT reduced the time for editing tasks by 30.81% and the time for error correcting tasks by 47.96%. For both editing and correcting tasks, the improvements of VT came from reducing the text-selecting time. For the editing tasks, the compound command of VT does not need precise selection of the text before issuing the editing command. For the correction tasks, VT could inference the location to replace or insert the new phrase while iOS's Voice Control requires precise selection of the erroneous phrase. Although iOS's Voice Control has voice commands to correct text without touch input, such as "replace {phrase} with {phrase}" and "insert {phrase} before/after {phrase}", those voice commands are not usable for mistyped phrases that do not have recognizable pronunciations. There are 53.6% correction trials in the study have this kind of mistyped phrases. When using iOS's Voice Control, those trials require precisely selecting the erroneous phrases before correcting them by speech.

While the improvements were quite significant and clear cut, there are many limitations of the study and further improvements to be made for VT. VT could use a customized speech recognition model. The editing tasks tested were representative of, but nonetheless not literately, naturally occurring tasks in real world writing activities. Depending on the app context, there may be a need to extend the command set. How well the current set of simple algorithms can scale to the extended set remains to be explored. More complex algorithms could be proven necessary.

8 CONCLUSION

We researched and developed VT, a voice and touch based multimodal text editing and correcting method for smartphones. It allows a user to combine voice and touch input to edit text and correct errors. For text editing, the user can tap a text area then speak a compound command such as "bold the paragraph" to edit text, or the user can glide the finger over a text segment and speak out an editing command (e.g., "cut", "copy", or "bold") to edit text. For text correction, the user can point to the approximate location of errors and speak the new word or phrase to correct them. VT can work without precise text selection because it can infer a user's editing or correcting intention by combining the voice input, touch input, and text context. Our user study showed that VT greatly improves the text editing and correcting performance over the existing touch-only method and iOS's Voice Control method. Compared to touch-only method, VT reduced the task completion time of text editing tasks by 30.80%, and the time for text correcting by 29.97%. Compared to iOS's Voice Control method, VT reduced the task completion time of text editing tasks by 30.81%, and the time for text correcting by 47.96%. VT is also strongly preferred by participants. Overall, VT mitigates the weaknesses of touch and voice input by leveraging the strengths of the other, improving the efficiency of text editing and correcting on mobile devices.

ACKNOWLEDGMENTS

We thank anonymous reviewers for their insightful comments, and our user study participants. This work was supported by NIH award R01EY030085 and NSF awards 1805076, 1936027, 2113485, 1815514. This work was done as part of the Ph.D. dissertation of Maozheng Zhao, a Stony Brook Ph.D. student supervised by Dr. Xiaojun Bi.

REFERENCES

[1] Ohoud Alharbi, Ahmed Sabbir Arif, Wolfgang Stuerzlinger, Mark D. Dunlop, and Andreas Komninos. 2019. WiseType: A Tablet Keyboard with Color-Coded Visualization and Various Editing Options for Error Correction. In *Proceedings of the 45th Graphics Interface Conference on Proceedings of Graphics Interface 2019* (Kingston, Canada) *(GI'19)*. Canadian Human-Computer Communications Society, Waterloo, CAN, Article 4, 10 pages. https://doi.org/10.20380/GI2019.04

[2] Jessalyn Alvina, Carla F. Griggio, Xiaojun Bi, and Wendy E. Mackay. 2017. CommandBoard: Creating a General-Purpose Command Gesture Input Space for Soft Keyboard. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology* (Québec City, QC, Canada) *(UIST '17)*. ACM, New York, NY, USA, 17–28. https://doi.org/10.1145/3126594.3126639

[3] Apple. 2018. About the keyboards settings on your iPhone, iPad, and iPod touch. https://support.apple.com/en-us/HT202178. [Online; accessed 22-August-2019].

[4] Ahmed Sabbir Arif, Sunjun Kim, Wolfgang Stuerzlinger, Geehyuk Lee, and Ali Mazalek. 2016. Evaluation of a Smart-Restorable Backspace Technique to Facilitate Text Entry Error Correction. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems* (San Jose, California, USA) *(CHI '16)*. Association for Computing Machinery, New York, NY, USA, 5151–5162. https://doi.org/10.1145/2858036.2858407

[5] Xiaojun Bi, Yang Li, and Shumin Zhai. 2013. FFitts Law: Modeling Finger Touch with Fitts' Law. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Paris, France) *(CHI '13)*. Association for Computing Machinery, New York, NY, USA, 1363–1372. https://doi.org/10.1145/2470654.2466180

[6] Xiaojun Bi, Tom Ouyang, and Shumin Zhai. 2014. Both Complete and Correct?: Multi-objective Optimization of Touchscreen Keyboard. In *Proceedings of the 32Nd Annual ACM Conference on Human Factors in Computing Systems* (Toronto, Ontario, Canada) *(CHI '14)*. ACM, New York, NY, USA, 2297–2306. https://doi.org/10.1145/2556288.2557414

[7] Chung-Cheng Chiu, Tara N Sainath, Yonghui Wu, Rohit Prabhavalkar, Patrick Nguyen, Zhifeng Chen, Anjuli Kannan, Ron J Weiss, Kanishka Rao, Ekaterina Gonina, et al. 2018. State-of-the-art speech recognition with sequence-to-sequence models. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 4774–4778.

[8] Wenzhe Cui, Jingjie Zheng, Blaine Lewis, Daniel Vogel, and Xiaojun Bi. 2019. HotStrokes: Word-Gesture Shortcuts on a Trackpad. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems* (Glasgow, Scotland Uk) *(CHI '19)*. ACM, New York, NY, USA, Article 165, 13 pages. https://doi.org/10.1145/3290605.3300395

[9] Wenzhe Cui, Suwen Zhu, Mingrui Ray Zhang, H. Andrew Schwartz, Jacob O. Wobbrock, and Xiaojun Bi. 2020. JustCorrect: Intelligent Post Hoc Text Correction Techniques on Smartphones. In *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology* (Virtual Event, USA) *(UIST '20)*. Association for Computing Machinery, New York, NY, USA, 487–499. https://doi.org/10.1145/3379337.3415857

[10] Mark Davies. 2018. The corpus of contemporary American English: 1990-present.

[11] Android developers. 2021. Android EditText. https://developer.android.com/reference/android/widget/EditText. [Online; Accessed: 2021-04-06].

[12] Android developers. 2021. Android.Speech. https://developer.android.com/reference/android/speech/package-summary. [Online; Accessed: 2021-04-06].

[13] A. D. N. Edwards. 2002. *Multimodal Interaction and People with Disabilities*. Springer Netherlands, Dordrecht. https://doi.org/10.1007/978-94-017-2367-1_5

[14] Michael Fischer, Giovanni Campagna, Silei Xu, and Monica S. Lam. 2018. Brassau: Automatic Generation of Graphical User Interfaces for Virtual Assistants. In *Proceedings of the 20th International Conference on Human-Computer Interaction with Mobile Devices and Services* (Barcelona, Spain) *(MobileHCI '18)*. Association

for Computing Machinery, New York, NY, USA, Article 33, 12 pages. https://doi.org/10.1145/3229434.3229481

[15] Vittorio Fuccella, Poika Isokoski, and Benoit Martin. 2013. Gestures and Widgets: Performance in Text Editing on Multi-touch Capable Mobile Devices. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Paris, France) *(CHI '13)*. ACM, New York, NY, USA, 2785–2794. https://doi.org/10.1145/2470654.2481385

[16] Vittorio Fuccella and Benoît Martin. 2017. TouchTap: A Gestural Technique to Edit Text on Multi-Touch Capable Mobile Devices. In *Proceedings of the 12th Biannual Conference on Italian SIGCHI Chapter* (Cagliari, Italy) *(CHItaly '17)*. Association for Computing Machinery, New York, NY, USA, Article 21, 6 pages. https://doi.org/10.1145/3125571.3125579

[17] Joshua Goodman, Gina Venolia, Keith Steury, and Chauncey Parker. 2002. Language Modeling for Soft Keyboards. In *Proceedings of the 7th International Conference on Intelligent User Interfaces* (San Francisco, California, USA) *(IUI '02)*. ACM, New York, NY, USA, 194–195. https://doi.org/10.1145/502716.502753

[18] Google. 2021. Get started with Voice Access. https://support.google.com/accessibility/android/answer/6151848?hl=en. [Online; Accessed: 2021-07-18].

[19] google.com. 2021. Type with your voice. https://support.google.com/docs/answer/4492226?hl=en#zippy=%2Cselect-text. [Online; accessed 6-April-2021].

[20] Christian Holz and Patrick Baudisch. 2011. Understanding Touch. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Vancouver, BC, Canada) *(CHI '11)*. Association for Computing Machinery, New York, NY, USA, 2501–2510. https://doi.org/10.1145/1978942.1979308

[21] iMore.com. 2021. Everything you can do with Voice Control on iPhone and iPad. https://www.imore.com/everything-you-can-do-voice-control-iphone-and-ipad. [Online; Accessed: 2021-07-18].

[22] ExIdeas Inc. 2018. MessagEase - The Smartest Touch Screen keyboard. https://www.exideas.com/ME/index.php. [Online; accessed 22-August-2019].

[23] Grammarly Inc. 2020. Grammarly Keyboard. https://en.wikipedia.org/wiki/Grammarly [Online; accessed May-2020].

[24] Poika Isokoski, Benoît Martin, Paul Gandouly, and Thomas Stephanov. 2010. Motor Efficiency of Text Entry in a Combination of a Soft Keyboard and Unistrokes. In *Proceedings of the 6th Nordic Conference on Human-Computer Interaction: Extending Boundaries* (Reykjavik, Iceland) *(NordiCHI '10)*. ACM, New York, NY, USA, 683–686. https://doi.org/10.1145/1868914.1869004

[25] Michael Johnston, John Chen, Patrick Ehlen, Hyuckchul Jung, Jay Lieske, Aarthi Reddy, Ethan Selfridge, Svetlana Stoyanchev, Brant Vasilieff, and Jay Wilpon. 2014. MVA: The Multimodal Virtual Assistant. In *Proceedings of the 15th Annual Meeting of the Special Interest Group on Discourse and Dialogue (SIGDIAL)*. Association for Computational Linguistics, Philadelphia, PA, U.S.A., 257–259. https://doi.org/10.3115/v1/W14-4335

[26] Clare-Marie Karat, Christine Halverson, Daniel Horn, and John Karat. 1999. Patterns of Entry and Correction in Large Vocabulary Continuous Speech Recognition Systems. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Pittsburgh, Pennsylvania, USA) *(CHI '99)*. Association for Computing Machinery, New York, NY, USA, 568–575. https://doi.org/10.1145/302979.303160

[27] Bryan Klimt and Yiming Yang. 2004. The enron corpus: A new dataset for email classification research. In *European Conference on Machine Learning*. Springer, 217–226.

[28] Andreas Komninos, Mark Dunlop, Kyriakos Katsaris, and John Garofalakis. 2018. A Glimpse of Mobile Text Entry Errors and Corrective Behaviour in the Wild. In *Proceedings of the 20th International Conference on Human-Computer Interaction with Mobile Devices and Services Adjunct* (Barcelona, Spain) *(MobileHCI '18)*. ACM, New York, NY, USA, 221–228. https://doi.org/10.1145/3236112.3236143

[29] N. Krahnstoever, S. Kettebekov, M. Yeasin, and R. Sharma. 2002. A Real-Time Framework for Natural Multimodal Interaction with Large Screen Displays. In *Proceedings of the 4th IEEE International Conference on Multimodal Interfaces (ICMI '02)*. IEEE Computer Society, USA, 349. https://doi.org/10.1109/ICMI.2002.1167020

[30] Per Ola Kristensson and Shumin Zhai. 2007. Command Strokes with and Without Preview: Using Pen Gestures on Keyboard for Command Selection. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (San Jose, California, USA) *(CHI '07)*. ACM, New York, NY, USA, 1137–1146. https://doi.org/10.1145/1240624.1240797

[31] Vladimir Iosifovich Levenshtein. 1966. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady* 10, 8 (feb 1966), 707–710. Doklady Akademii Nauk SSSR, V163 No4 845-848 1965.

[32] Toby Jia-Jun Li, Jingya Chen, Haijun Xia, Tom M. Mitchell, and Brad A. Myers. 2020. Multi-Modal Repairs of Conversational Breakdowns in Task-Oriented Dialogs. In *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology* (Virtual Event, USA) *(UIST '20)*. Association for Computing Machinery, New York, NY, USA, 1094–1107. https://doi.org/10.1145/3379337.3415820

[33] Google LLC. 2020. Gboard. https://en.wikipedia.org/wiki/Gboard [Online; accessed May-2020].

[34] Matt Mahoney. 2011. About Text8 file. http://mattmahoney.net/dc/textdata.html. [Online; accessed May-2020].

[35] Jennifer Mankoff, Gregory D Abowd, and Scott E Hudson. 2000. OOPS: a toolkit supporting mediation techniques for resolving ambiguity in recognition-based interfaces. *Computers & Graphics* 24, 6 (2000), 819–834. https://doi.org/10.1016/S0097-8493(00)00085-6 Calligraphic Interfaces: towards a new generation of interactive systems.

[36] Tomas Mikolov, Kai Chen, Greg S. Corrado, and Jeffrey Dean. 2013. Efficient Estimation of Word Representations in Vector Space. http://arxiv.org/abs/1301.3781

[37] nuance.com. 2021. Dragon Speech Recognition - Get More Done by Voice: Dragon. https://www.nuance.com/dragon.html. [Online; accessed 6-April-2021].

[38] Per Ola Kristensson and Keith Vertanen. 2011. Asynchronous Multimodal Text Entry Using Speech and Gesture Keyboards.. In *Proceedings of the International Conference on Spoken Language Processing* (Florence, Italy). 581–584.

[39] Sharon Oviatt and Philip Cohen. 2000. Perceptual User Interfaces: Multimodal Interfaces That Process What Comes Naturally. *Commun. ACM* 43, 3 (March 2000), 45–53. https://doi.org/10.1145/330534.330538

[40] Sharon Oviatt, Phil Cohen, Lizhong Wu, John Vergo, Lisbeth Duncan, Bernhard Suhm, Josh Bers, Thomas Holzman, Terry Winograd, James Landay, Jim Larson, and David Ferro. 2000. Designing the User Interface for Multimodal Speech and Pen-Based Gesture Applications: State-of-the-Art Systems and Future Research Directions. *Hum.-Comput. Interact.* 15, 4 (Dec. 2000), 263–322. https://doi.org/10.1207/S15327051HCI1504_1

[41] Sharon Oviatt and Philip R. Cohen. 2015. *The Paradigm Shift to Multimodality in Contemporary Computer Interfaces*. Morgan & Claypool Publishers.

[42] S. Oviatt and R. VanGent. 1996. Error resolution during multimodal human-computer interaction. In *Proceeding of Fourth International Conference on Spoken Language Processing. ICSLP '96*, Vol. 1. 204–207 vol.1. https://doi.org/10.1109/ICSLP.1996.607077

[43] Kseniia Palin, Anna Feit, Sunjun Kim, Per Ola Kristensson, and Antti Oulasvirta. 2019. How do People Type on Mobile Devices? Observations from a Study with 37,000 Volunteers.. In *Proceedings of 21st International Conference on Human-Computer Interaction with Mobile Devices and Services (MobileHCI'19)*. ACM.

[44] Radiah Rivu, Yasmeen Abdrabou, Ken Pfeuffer, Mariam Hassib, and Florian Alt. 2020. Gaze'N'Touch: Enhancing Text Selection on Mobile Devices Using Gaze. In *Extended Abstracts of the 2020 CHI Conference on Human Factors in Computing Systems* (Honolulu, HI, USA) *(CHI EA '20)*. Association for Computing Machinery, New York, NY, USA, 1–8. https://doi.org/10.1145/3334480.3382802

[45] Ritam Jyoti Sarmah, Yunpeng Ding, Di Wang, Cheuk Yin Phipson Lee, Toby Jia-Jun Li, and Xiang 'Anthony' Chen. 2020. Geno: A Developer Tool for Authoring Multimodal Interaction on Existing Web Applications. In *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology* (Virtual Event, USA) *(UIST '20)*. Association for Computing Machinery, New York, NY, USA, 1169–1181. https://doi.org/10.1145/3379337.3415848

[46] Khe Chai Sim. 2010. Haptic Voice Recognition: Augmenting speech modality with touch events for efficient speech recognition. In *2010 IEEE Spoken Language Technology Workshop*. 73–78. https://doi.org/10.1109/SLT.2010.5700825

[47] Khe Chai Sim. 2012. Speak-as-you-swipe (SAYS): A Multimodal Interface Combining Speech and Gesture Keyboard Synchronously for Continuous Mobile Text Entry. In *Proceedings of the 14th ACM International Conference on Multimodal Interaction* (Santa Monica, California, USA) *(ICMI '12)*. ACM, New York, NY, USA, 555–560. https://doi.org/10.1145/2388676.2388793

[48] Shyamli Sindhwani, Christof Lutteroth, and Gerald Weber. 2019. ReType: Quick Text Editing with Keyboard and Gaze. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems* (Glasgow, Scotland Uk) *(CHI '19)*. ACM, New York, NY, USA, Article 203, 13 pages. https://doi.org/10.1145/3290605.3300433

[49] Bernhard Suhm, Brad Myers, and Alex Waibel. 2001. Multimodal error correction for speech user interfaces. *ACM transactions on computer-human interaction (TOCHI)* 8, 1 (2001), 60–98.

[50] 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 *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems* (Seoul, Republic of Korea) *(CHI '15)*. ACM, New York, NY, USA, 659–668. https://doi.org/10.1145/2702123.2702135

[51] Daniel Vogel and Patrick Baudisch. 2007. Shift: A Technique for Operating Pen-Based Interfaces Using Touch. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (San Jose, California, USA) *(CHI '07)*. Association for Computing Machinery, New York, NY, USA, 657–666. https://doi.org/10.1145/1240624.1240727

[52] Klaus Weidner. 2018. Hackers Keyboard. http://code.google.com/p/hackerskeyboard/ [Online; accessed 22-August-2019].

[53] Jackie (Junrui) Yang, Monica S. Lam, and James A. Landay. 2020. DoThisHere: Multimodal Interaction to Improve Cross-Application Tasks on Mobile Devices. In *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology* (Virtual Event, USA) *(UIST '20)*. Association for Computing Machinery, New York, NY, USA, 35–44. https://doi.org/10.1145/3379337.3415841

[54] Mingrui Ray Zhang, He Wen, and Jacob O. Wobbrock. 2019. 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 Orleans, LA, USA) *(UIST '19)*. Association for Computing Machinery, New York, NY, USA, 843–855. https://doi.org/10.1145/3332165.3347924

[55] Mingrui Ray Zhang and O. Jacob Wobbrock. 2020. Gedit: Keyboard gestures for mobile text editing. In *Proceedings of Graphics Interface (GI '20)* (Toronto, Ontario) *(GI '20)*. Canadian Information Processing Society, Toronto, Ontario, 97–104.