PhraseFlow: Designs and Empirical Studies of Phrase-Level Input

Mingrui "Ray" Zhang
The Information School
University of Washington
Seattle, WA
mingrui@uw.edu

Shumin Zhai
Google
Mountain View, CA
zhai@acm.org

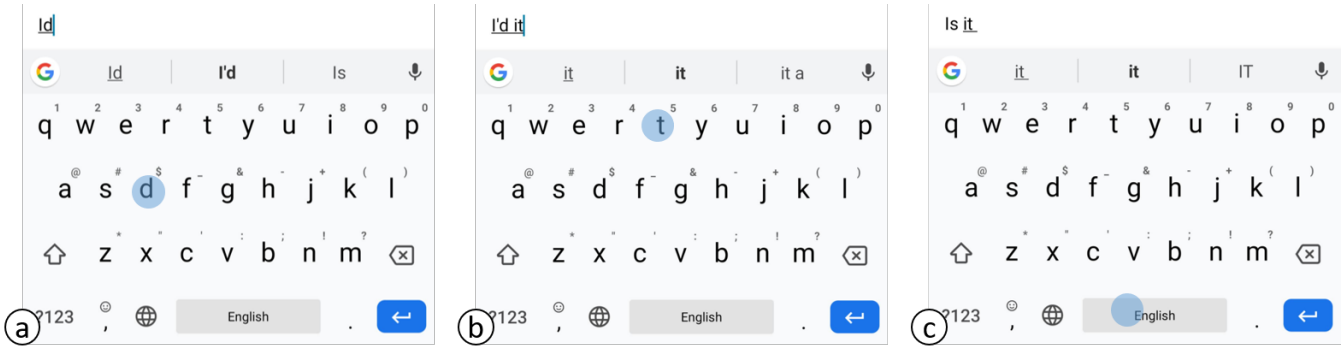

Figure 1: The final version of PhraseFlow. (a) When the user typed "id" (but meant "is") , (b) it was first corrected to "I'd" after the first space press. However, the correction was not committed. (c) After the user typed text "it", the word was finally corrected and committed as "is" on the second space press

ABSTRACT

Decoding on phrase-level may afford more correction accuracy than on word-level according to previous research. However, how phrase-level input affects the user typing behavior, and how to design the interaction to make it practical remain under explored. We present PhraseFlow, a phrase-level input keyboard that is able to correct previous text based on the subsequently input sequences. Computational studies show that phrase-level input reduces the error rate of autocorrection by over 16%. We found that phrase-level input introduced extra cognitive load to the user that hindered their performance. Through an iterative design-implement-research process, we optimized the design of PhraseFlow that alleviated the cognitive load. An in-lab study shows that users could adopt PhraseFlow quickly, resulting in 19% fewer error without losing speed. In real-life settings, we conducted a six-day deployment study with 42 participants, showing that 78.6% of the users would like to have the phrase-level input feature in future keyboards.

CHI '21, May 8–13, 2021, Yokohama, Japan

ACM ISBN 978-1-4503-8096-6/21/05…$15.00
https://doi.org/10.1145/3411764.3445166

CCS CONCEPTS

• **Human-centered computing** → **Text input**.

KEYWORDS

Text entry, autocorrection, phrase-level input, keyboard

ACM Reference Format:
Mingrui "Ray" Zhang and Shumin Zhai. 2021. PhraseFlow: Designs and Empirical Studies of Phrase-Level Input. In *CHI Conference on Human Factors in Computing Systems (CHI '21), May 8–13, 2021, Yokohama, Japan.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3411764.3445166

1 INTRODUCTION

Autocorrection has become an essential part of touchscreen smartphone keyboards. Due to the small screen size relative to the finger width, fast typing on a smartphone without autocorrection can produce up to 38% word errors [5, 14]. To remedy the problem, given a sequence of touch points, a keyboard decoder can use spatial and language models to find the best candidate and performs correction on the typed text. Simulation studies show such auto-corrections can dramatically reduce the error rate in touch keyboards [14]. Indeed, commercial mobile keyboards such as Gboard [17], SwiftKey [26] and the iOS keyboard all provide word-level decoding, which corrects the latest typed literal string to an in-vocabulary word: for example, correcting *loce* to *love*. Banovic et al. [6] has shown that with a good autocorrection decoder, the user typed 31% faster than without autocorrection.

However, word-level decoding has two major drawbacks. First, at times it can be difficult for the decoder to determine if a word

Table 1: Correction examples with word-level Gboard and phrase-level PhraseFlow. Phrase-level decoding can correct previous text using the future input context to avoid false corrections (row 1 & 2) or no corrections (row 3 & 4). It is also able to correct space-related errors (row 5 & 6)

Raw Input	Word-level Decoding	Phrase-level Decoding
stidf penalty	stuff penalty	stiff penalty
what id your	what i'd your	what is your
Feams canyon	Feams canyon	Great Canyon
Kps angeles	Kps Angeles	Los Angeles
Xommu ication	Xommu ication	Communication
facin g north	facing g north	facing north

makes sense without incorporating the future input context. For example, if a user types *he loces*, the keyboard may correct *loces* to *loves*; however, if the user continues typing *in Paris*, the expected correction should be *lives*. Not incorporating the future context can either lead to wrong corrections or fail to correct the text. Second, space-related errors often cannot be handled well without future context. Word-level decoding uses the space key tap as an immediate and deterministic commit signal, thus does not afford the benefit of correcting for superfluous touch on it or alternative possible user intentions such as aiming for the C V B N keys above the space key. As a consequence, space-related errors such as *th e*, *iter ational* can not be properly handled. Furthermore, a word-level decoder often fails to correct contiguous text without spaces such as *theboyiscominghomenow*, as it mainly consider word candidates.

One possible solution to the above problems is to decode touch points on phrase-level, instead of only decoding and correcting the touch points of the last word. Phrase-level decoding may continue to decode the touch points even if the space key is pressed, and outputs phrase candidates. Velocitap [37] was one of the first attempts towards this idea: it presented a sentence-based decoder that was able to correct multiple words at a time. Follow-up projects by Vertanen and colleagues [34, 35] further investigated smartwatch devices decoding accuracy and typing performance on word level, multi-words level and sentence level. We show examples of word and phrase-level correction results in Table 1, based on actual results from Gboard and a version of our phrase-level keyboard, PhraseFlow, presented later in this paper.

However, making phrase-level input practical faces many challenges. First, corrections beyond the last typed word require the user to pay attention to the early part of the phrase being typed; Second, delayed correction of the previous text requires the user to trust that the decoder would eventually and successfully correct the errors. If the phrase auto-correction failures, the delayed manual repair cost could be higher. Building upon the previous work, we present PhraseFlow, a keyboard prototype that focused on designing and studying the interfaces to support the phrase-level decoding. We limited the scope only to touch typing, in contrast to gesture typing [41]. PhraseFlow aims to address three essential types of questions in the phrase-level input interaction:

(1) How to change and design the interface and interactions that match phrase-level decoding?
(2) How does phrase-level input affect the user's typing behavior and cognitive load?
(3) What are the user reactions and experiences when using PhraseFlow as their daily keyboard?

We modified the Finite State Transducer (FST) based decoder [29] of Gboard to support phrase level decoding. We then performed simulation tests on the touch data collected from a composition task. The results show that word-level decoding had 7.76% word error rate (WER) while phrase-level decoding had 6.47% WER, a 16.6% relative error reduction on this data set. Space related errors were also corrected by PhraseFlow.

To explore the design space of PhraseFlow, we iterated on multiple options of: 1. visual correction effects; 2. decoding commit gesture and behavior; and 3. suggestion displays. We first built a version of PhraseFlow with similar designs to the previous phrase-level input work [34, 35, 37]. The study results showed that phrase-level input with such designs introduced extra cognitive loads to the user, and alternative designs were needed to mitigate the effect. By incorporating empirical study results from the iteration, our final version keyboard managed to reduce the cognitive load and reached a comparable level of performance of the commercial keyboard. To test the user acceptance of the keyboard, we conducted a six-day deployment study with 42 participants. During the study, participants used PhraseFlow as their primary keyboard. The survey results showed that overall 78.6% of the participants would like to have phrase-level typing in their future keyboards, in comparison to 7.1% of the participants disliked the feature. Overall, the study results suggest phrase level input is a promising feature for future mobile keyboards.

Drawing from the many lessons in implementing PhraseFlow, we offer design guidelines for future keyboards with phrase-level input, and identify the challenges and opportunities to further improve phase level input.

2 CHALLENGES OF DESIGNING PHRASEFLOW

The input chunk for typewriter-like physical keyboards is on *character level*: each key press modifies one character at a time. With smart functions such as auto-correction and word-prediction, touchscreen keyboards have enlarged the input chunk into *word level*: a string of characters inaccurately entered can be corrected into a likely intended word upon the press of the space key, which relaxes the need to type each character accurately. The basic research question of PhraseFlow is **how to further enlarge the input chunk to *phrase level***, as multiple words are corrected through one operation. Studies in human factors and psychology tended to find word as the basic chunk of typing [19, 32]. This means that people mainly focus on only the current word when typing. Enlarging the input chunk into *phrase level* requires the user to pay extra attention on the previous text, which might hinder the typing performance. PhraseFlow therefore needs to overcome three new design challenges:

C1. How to signal the change when corrections happen. As the phrase-level input might change multiple words at the same

time (and change the same word multiple times), we need to design effective feedback that is salient enough to inform the user about the correction, yet unobtrusive to avoid distracting the user from typing.

C2. How to design multi-word candidates and text output to reduce user's cognitive load. For word-level keyboards, the user only attends the latest word. Once the last word is entered, they will shift their attention to the next one. For phrase-level keyboards, users need to attend to multiple words during typing. To reduce the cognitive load, we need to explore ways of presenting the text and suggestions effectively.

C3. How to minimize correction failures. Manually recovering from a correction failure in phrase-level costs more than word-level corrections, because the failure can happen words away. While incorporating longer context in the decoding process might improve the accuracy, recovering a correction failure further away can also be more costly. We thus need to design better interactions to minimize correction failures.

To our knowledge there is no single research method that can lead to all the insights needed to make significant keyboard performance progress. We therefore applied a variety of HCI research methods to address the challenges before us, including prototyping, simulation (offline computational tests), and lab-based composition or transcription typing, with both performance and subjective experience measurements. As a research vehicle We built PhraseFlow based on the Gboard [17] code base, bearing all its strengths and limitations. On the positive side, we leveraged many years of engineering work of Gboard on product polishing, computational performance and UI iteration, so a meaningful difference caused by the phrase-level input could be found against a strong baseline. On the other hand, Gboard as a commercial keyboard has a very compact language model with short span ngrams. Note that previous work on the trade-off between the language model size and its correction power, albeit on a limited data set, did not show dramatic increase in accuracy from very large n-gram models [37].

3 RELATED WORK

3.1 Keyboard Decoding Models

The decoder of a smartphone keyboard contains two essential models: the spatial model and the language model [9, 14, 16, 21]. The spatial model relates intended keys to the probability distributions of touch coordinates and other features [5, 13, 40, 44]. The distribution is then combined with a language model, such as an n-gram back-off model [21], to correctly decode noisy touch events into the intended text [14, 16]. Borrowing the idea from speech recognition, the classic approach to combining the spatial model and language model estimations is through the Bayes' rule, as in Goodman et al. [16]. Practical keyboards may also model spelling errors by adding letter insertion and deletion probability estimates in its decoding algorithms [29].

Various works have been proposed to improve the text entry decoding process. For example, the Finger Fitts Law [8] proposed a dual-distribution to model finger's touch point distribution accurately; WalkType [15] incorporated the accelerometer data to improve the touch accuracy during walking conditions; Yin et al. proposed a hierarchical spatial backoff model to make the touchscreen keyboards adaptive to individuals and postures [40]. Weir et al. [38] utilized the touch pressure to "lock" the characters during decoding. Zhu et al. [44] showed that participants could type reasonably fast on an invisible keyboard with adjusted spatial models.

3.2 Phrase-level Text Entry

We are not the first to explore phrase-level text entry techniques. Production level keyboards (e.g., Google Gboard) have long had a "space omission" feature which allows its user to enter multiple words a time without a space separator, although only reliably for the most common short phrases. For example, "thankyouverymuch" is decoded into "Thank you very much". Vertanen et al. [37] developed VelociTap, a phrase-level decoder for mobile text entry. VelociTap combines a 4-gram word model and a 12-gram character model to decode the touch inputs into correct sentences. In one simulated replay study of typing common phrases on a watch-sized keyboard, assuming perfect word deliminator input [34], Vertanen and colleagues demonstrated that phrase-level decoding could reduce the character error rate (CER) from 2.3% to 1.8%. Together with its follow-up projects [34, 35], various factors such as visual feedback on touched keys, keyboard size, word-delimiter actions (e.g. a right swipe), and decoding scopes were also investigated for phrase-level input.

The previous work on phrase-level input focused on the algorithms and the performance differences between phrase-level and word-level decoders. An important difference between PhraseFlow and previous work is that previous research all treated the space press as a deterministic word delimiter during decoding, while PhraseFlow treats it as a decodable press, so as to minimize space-related errors.

Commercial keyboards such as Gboard and iOS keyboard in recent years have also released a feature called *post-correction* [29], which is a subset to phrase-level correction. Post-correction will revise the one word preceding the current typing word if the correction confidence is high. However, post-correction only correct at most one previous word, limiting the power of using the subsequent context. It also does not correct the space-related errors mentioned in the introduction. For PhraseFlow, we explored different word limits that the keyboard can correct, derived a more generalized design of the phrase-level correction interaction, and filled the gap in the lack of empirical results on phrase-level correction techniques in the literature.

Typing research tended to find word as the basic processing chunk [19, 32]. On the other hand, experience and practice tended to increase the chunk size of information processing [27] or shift motor control behavior to higher levels of control hierarchy [30]. With proper design, fluent typists might adapt to the phrase-level input after practising.

3.3 Interaction and Interfaces for Touch Screen Keyboards

The interface of a touch screen keyboard can affect the user's typing behavior significantly. Arnold et al. [4] investigated the prediction interface by comparing word and phrase suggestions, finding that phrase suggestions affected the input contents more than the word

suggestions. Quinn and Zhai [31] conducted a cost-benefit study on suggestion interactions, finding that always showing suggestions required extra attention and could potentially hinder the typing performance. Similar results was also found by Zhang et al. [43] in their study of comparing text entry performance under different speed-accuracy conditions. WiseType [1] compared the visual effects of auto-correction and error-indication, finding that color-coded text background could improve the typing speed and accuracy. We incorporated many of the previous findings as guidance to design PhraseFlow interactions.

4 PHRASE-LEVEL DECODER

The current decoder of Gboard [17] is a finite-state transducer (FST) [29] containing a spatial keyboard model and a n-gram language model consisting a 164K English word vocabulary and 1.3M ngrams (n up to 5). The original decoder would *commit* the last word and reset its status when the space key was pressed, and then restart the FST state with the touch points of the next word. For example, if the user typed *inter* and pressed the space key, the decoder would reset and output *inter* as the best candidate; when the user continued typing *ational*, the decoder would only decode the touchpoints of *ational*, failing to correct the whole typing to *international*.

To turn the decoder into phrase level, we need to make the touch on space key decodable. We thus disabled the reset action of the decoder when a space was entered, so that it could continue the decoding process and treat the space touch as a normal touch point on the letter keys. In this way, the decoder was able to output phrase suggestions based on a touch sequence across the space key. For example, *inter ational* in which n is mistyped as space, would be treated as a whole sequence, including the space in the middle, and be decoded to *international*. The decoder could also handle longer phrases, such as correcting "*I love in new yirk*" into "*I live in New York*", as it now treats a multi-word touch sequence as decodable, rather than splitting the sequence into five touch sequences separated by the space key and resetting the state after each sequence.

Similar to VelociTap [37], to enable decoding touch sequence without word-delimiters, we also decreased the penalty of omitting a space between words, so that the decoder was able to handle contiguous text without space in between, such as *whatstheweathertoday*.

5 PHRASEFLOW V1.0

Figure 2 shows the interface of PhraseFlow v1.0. The workflow is as follows:

(1) The user types the raw text, which might contains typos and spaces.
(2) PhraseFlow decodes the touch input and displays the candidates in the suggestion bar. The text being decoded is underlined in the text window, indicating the range that might be updated in the future. We call this part of the text "*the active text*".
(3) PhraseFlow will apply the candidate to the underlined text when the user performs a commit action. The decoder will reset its state and remove the underline.
(4) Before committing, the user can modify the underlined text to update the decoded candidates.

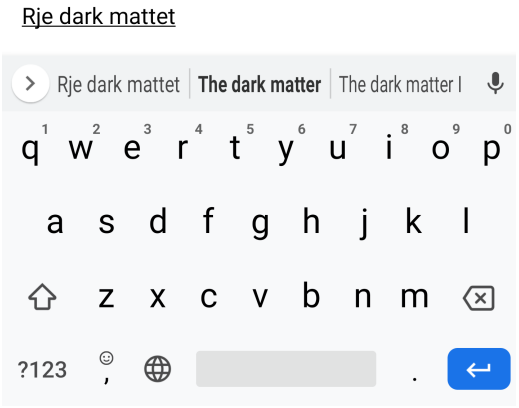

Figure 2: The interface of PhraseFlow v1.0. The keyboard layout was the same as Gboard. The typed text here is *Rje dark mettet*, and the autocorrection candidate *The dark matter* is in bold. Three candidates are shown in the list: the literal string, the autocorrection candidate, and the second best candidate

There were three kinds of commit actions: by selecting the candidate in the suggestion bar, by pressing a punctuation key, or the keyboard would commit every *n*th space the user typed. The later two actions would apply the default autocorrection candidate to the text. For the example in the figure, if the n for every *n*th space was set to 3, then the keyboard would commit *The dark matter* when the user pressed a space, as there were already two spaces typed in the active text.

5.1 Interface and Interaction Design

For the first version of PhraseFlow, we explored several design options on the commit method, visual effect of correction, suggestion bar display, and active text marker. We chose those options as they were reported to affect the typing performance in the previous work [1, 34, 35, 37].

Commit Method Since the space press was no longer a commit action with PhraseFlow, we needed to design a new interaction to let the user commit the correction candidate. Previous work [37] considered using swipe as the commit method. However, swipe also required the user to perform a very different gesture during tap-typing, and the gesture also confused the user with gesture typing on mobile keyboards.

We made PhraseFlow commit the correction to the active text on every *n*th space the user typed, *i.e.*, when the user typed the *n*th space in the active text (we call it *nth space commit method*). The rationale was that the users were already used to the space commit method with current keyboards, thus extra interaction for committing would increase the cognitive and manual control cost. Space press was a necessary step to compose the text, thus it was natural as a committing option. If n was set to 1, PhraseFlow would behave exactly the same as the current word-level decoding

keyboards, i.e. committing the text on each space press. A larger n would potentially offer greater post correction power, but also demands more user attention on the longer span active text.

Besides the space press, current keyboards also support pressing on punctuation keys, or selecting the candidate in the suggestion bar to trigger the committing, and these two actions were kept in PhraseFlow. Whenever the correction is committed, the decoder will reset its decoding status and restart the decoding for new inputs.

Visual Effect of Correction As pointed out in the design challenge section C1, it is important to design a good signal when correction happens. For example, if *is coning home* is corrected to *is coming home*, the user should be able to notice the change. We experimented with three feedback effects to indicate the correction after the user performs a committing action: 1) *Background flash*. When a string of text was corrected to another one, its background color would flash for 400ms. This is used by many current commercial keyboards. 2) *Color flash*. The color of the changed text would flash when it was corrected. 3) *Color change*. The color of the text would change to blue when it was corrected, and would change back when a new input action (such as cursor-moving, typing) happened. The three effects are shown in Figure 3(a).

Active Text Marker As is often suggested [28, 39], a system's internal state should be appropriately represented to the user. To make the user aware the range of the text that might be changed. We studied three active text markers of the active text shown in Figure 3(b): *underline*, *gray color*, and *no-marker*. The purpose was to have a visual effect that was not distracting but could also be informative of the active text range, echoing to design challenge C2. The no-marker design is used in iOS .

Suggestion Bar Display Since the decoded candidate may contains multiple words, the default display option of the suggestion bar, *i.e.*, always displaying three candidates, might make the user feel overwhelmed. To reduce the cognitive load of the user (challenge C2) and also make the candidate text always visible, we adopted a dynamic displaying design illustrated in Figure 3(c): the suggestion bar will first display all three candidates; as the text grows, it will only display two candidates and eventually decrease to one candidate if the active text is too long before the committing. In this way, we can show the complete candidate information without squeezing or hiding the text.

5.2 Study 1: Evaluating Design Options

After implemented the above design options, we conducted a pilot study with 30 participants (24 male, 6 female, 21 used Android, 9 used iOS) to test different options including the n values of the commit method (with n = 3,4,5), the visual correction effects, and the active text markers. The participants were instructed to compose messages freely using the keyboard and rate their preferences on each of the design.

The results of the study showed that for the nth commit method, participants generally preferred shorter n such as 3 and 4. Increasing n to 5 would make the participants feel too uncertain about whether the keyboard would correct the typing or not. For correction effects, *background flash* was the most preferred effect, which was not distracting but also salient enough. For active text marker, participants generally disliked the *no-marker* effect, complaining that it felt "fishy" of what the keyboard was doing. *Underline* was the most preferred marker, which was also currently used in Gboard. The study led us to choose the n=4 commit method, since the decoder could incorporate longer context. It also led us choose the *background flash* and *underline* for the correction effect and the active text marker respectively.

5.3 Study 2: Performance Simulation of PhraseFlow V1.0

This study used simulations, or "computational experiments" [14] to measure the accuracy of PhraseFlow v1.0 on autocorrection. Unlike Fowler et al. [14] which used model generated data in their simulation, we used a "remulation" approach [7] in this study: We recorded touch input data set collected in a text composition task. We then ran the data set through both the PhraseFlow v1.0 decoder and its Gboard word-level baseline decoder in a keyboard simulator. Emulating user typing behavior on a mobile phone, the simulator took touch coordinate sequences as input, then simulated the noisy touch input on a keyboard layout as input to the decoder. The simulator then compared the decoder output with the expected text, and calculated Word Error Rate (WER) of the output results.

To collect the evaluation data set, we conducted a composition study with 12 participants (7 male, 5 female) to gather their touch points on a keyboard without auto-correction functions. Modelled after the study of composition types by Vertanen and Kristensson [36], we designed six composition prompts listed in Table 2. The participants were instructed to type a long message based on the prompt fast and not to care about making errors. For each prompt, the typing lasted for three minutes. The study was conducted on a Pixel 3 smartphone, and autocorrection was disabled for the test. After the composition of each prompt, we asked the participants to read their raw text and type the corresponding correct message they intended to compose on a laptop. To ensure that participants typed the correct text on laptop, the experimenter and the participants reviewed the text together and corrected the errors if there were any. We logged their raw touch points, the raw text, and the corresponding correct text for simulation. In total, we collected 72 phrases composed of 4955 words. The average word length for prompt P1 to P6 was 63 words, 75 words, 66 words, 69 words, 64 words and 77 words respectively. Participants were compensated with $25 for the 45-minute study.

We measured the error rate of the original raw data in regards to the provided correct text, using character error rate (CER) and word error rate (WER). WER is the word-level edit distance [22]. The average CER was 6.18% (SD=2.9%) and the average WER was 26.4% (SD=11.1%). To conduct the offline computational evaluation, we kept the space key touch points but removed all punctuation-related touch points in the log data, fed the logged touch points into the simulator as the raw input (by replaying the touch points), and compared the simulation output with the correct text provided by the participants. We compared the current word-level decoder of Gboard, and the PhraseFlow decoder with the every-fourth-space commit method. The WER for the Gboard baseline was 7.76%, and 7.13% (n=4) for PhraseFlow. This 8.1% relative error reduction was a modest but clear improvement in error correction even on a mobile

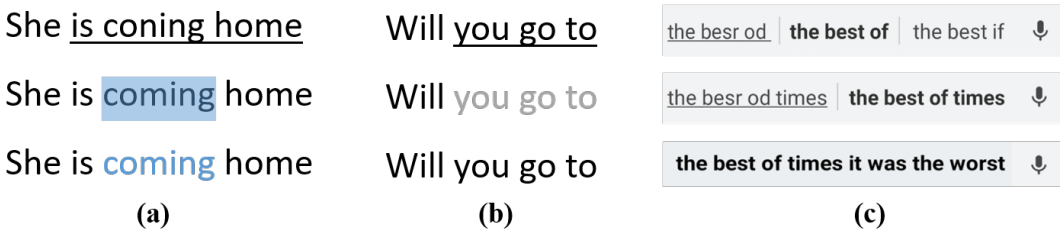

Figure 3: (a) Three visual effects of correction: when committing *is coning home*, the second row shows the *Background Flash* effect; the third row shows the *Color Flash* and the *Color Change* effect. (b) Three markers of the active text. (c) The suggestion bar display. As the input lengthens, the number of candidates decreases from three (top) to two (middle), and eventually one (bottom)

grade compact language model (set at 164K vocabulary and 1.3M ngrams). The WER was 7.14%, 7.03% and 7.08% when n was set to 2, 3 and 5. Given the similar performance, we fixed on n = 4 because of the result of study 1.

It is difficult to precisely compare this result with those by Vertanen and colleagues [34, 37]. Their results varied with a large set of decoding (from 0.4 M to 194 M word ngrams), UI (including no space separation between words), task (such as Enron mobile phrase set transcription), and form factor (phone vs. watch) variations. They generally show that more errors were corrected in phrase-level decoding than in word-level decoding. For example, in one simulation that replayed phrase input data collected on a watch-sized keyboard but assuming perfect space separation between words, they showed the character error rate (CER) reduced from 2.3% to 1.8% when the input and decoding size increased one word to five or six words. Assuming character errors were evenly distributed in words and the average word length were 4.7 letters as in common English, the corresponding word error rates (WER) were reduced from 10.35% to 8.18%[1].

5.4 Study 3: Pilot Study of PhraseFlow V1.0

We conducted a pilot study to test our design of PhraseFlow v1.0 in comparison to a word level baseline. We developed a text editor application shown in Figure 4 as the experimental apparatus. The application uses the transcription sequence model of Zhang and Wobbrock [42] for evaluation. Transcription sequence contains the sequence of the transcribed text each time its value changes, and by comparing the adjacent two sequences, it is able to analyze the dynamic text change during the typing procedure. Instead of composition tasks, we chose text transcription tasks for evaluation studies, as the text contents were controlled. Six participants (4 male, 2 female) were recruited via convenience sampling. The participants were told to enter the text shown on the screen as accurately and fast as possible using two keyboards: PhraseFlow v1.0 and unmodified Gboard. They were also allowed to use their comfortable posture for the task (all used two-thumb posture). The order of the keyboard was balanced. We randomly sampled 30 phrases from the Mackenzie phraseset [24] for each keyboard session. Afterwards, we conducted a short interview to gather their feedback. The participants were compensated with 15 USD for the 30-minute study.

[1]WER was calcualted using the formula $1 - CER^{4.7}$

Table 2: Six prompts for the composition tasks

P1	Suppose you are going to have dinner with your friend. Write a text message to schedule the time and places
P2	Write down an event happened recently
P3	Write about the local weather and your feelings about it
P4	Write about a recent trip experience
P5	Write down a recommendation about this study to your friend
P6	Free composition. Write down whatever in your mind

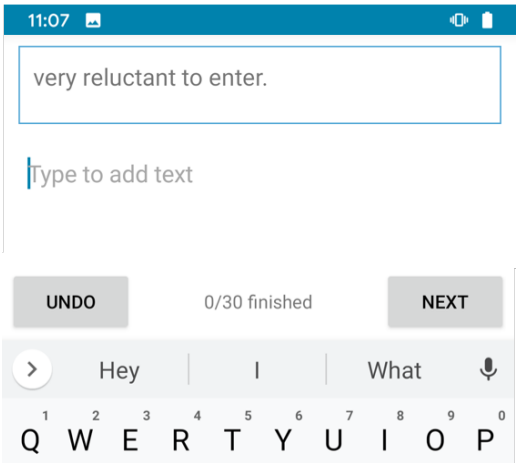

Figure 4: The text editor application used in the study. Hitting "undo" will restart the current trial

In total, 360 phrases were collected. We calculate words per minute (WPM) by subtracting the first character timestamp from the last character timestamp, as noted in [23]. The average typing speed was 52.3 WPM in the Gboard condition, and 43.5 WPM in the PhraseFlow condition. We also calculated the character error rate (CER) and word error rate (WER). The average CER was 0.6%

Table 3: The average speed and accuracy of each participant

	Speed (wpm)		Error (CER) %		Error (WER) %	
	Gboard	PhraseFlow	Gboard	PhraseFlow	Gboard	PhraseFlow
p1	35.6	23.0	0.8	1.0	3.0	4.7
p2	44.4	35.6	0	0.7	0	2.4
p3	62.1	49.1	0.1	0.4	3.1	0.5
p4	54.0	49.6	0.4	0.9	1.1	3.8
p5	64.3	60.5	0.4	0.6	1.9	1.9
p6	53.5	43.2	1.7	0.7	4.8	1.5

in the Gboard condition, and 0.7% in the PhraseFlow condition. The average WER was 2.3% in the Gboard condition and 2.4% in the PhraseFlow condition. Table 3 shows the performance of each participant.

To gain a deeper understanding of their typing behaviors, we analyzed the inter-key interval (IKI) [12] which was the time difference between two keypress events. We divided the IKI into two categories: the in-word IKI, *i.e.,* the time interval between two key presses within a word; and the between-word IKI, *i.e.*, the time interval between the presses of space and the next word. The average in-word IKI was 0.192s for Gboard and 0.210s for PhraseFlow. The average between-word IKI was 0.265s for Gboard and 0.311s for PhraseFlow. The larger difference on the between-word IKIs indicated that participants spent longer time to start typing a new word after pressing a space.

5.5 Discussion of the Pilot Study

What were the factors that caused the participants to type 10 WPM slower using PhraseFlow? The feedback of the participants pointed to two main reasons: 1) The raw text showed in the text output window was distracting. Although there was underline indicating that the text might be corrected later, many participants mentioned that looking at the raw text made them feel hesitant. P6 commented that "*(PhraseFlow) is stressful because you look at the raw text and you want to fix but it finally gets fixed.*" The hesitation might have caused slower IKIs for PhraseFlow. 2) The content of the suggestion bar was changing too much. During composing, not only each candidate length grew, the number of candidates also changed to fit in the bar space. Longer candidates required the user to spend more time reviewing them, as P1 commented, "*there is just too much going on in the suggestion bar. I have to read longer text and sometimes the number of options changes. It is distracting.*"

The analysis of IKIs also sheds light on participants' cognitive load during typing. Cognitive load may reflect the exterior information need during a task [33]. Studies in writings have found that shorter pauses contributed to higher writing fluency and lower cognitive load [2], and people generally had longer pauses at word boundaries [25]. The trend was similar from this study results: the in-word IKIs were on average shorter than the between-word IKIs, either in Gboard or PhraseFlow condition. However, the larger between-word IKIs of PhraseFlow (0.311 second) than Gboard (0.265 second) indicated that after pressing spaces, the participants might have carried greater cognitive load as they paused longer before starting typing the new word.

Overall, this pilot study shows that supporting phrase level input isn't an easy HCI problem. The greater attention demand in PhraseFlow v1.0 might have hindered the users' ability to take advantage of its features.

6 PHRASEFLOW V2.0

The pilot study of PhraseFlow v1.0 clearly showed that alternative designs on commit method and text display were needed. This led to several changes in the design of PhrsaeFlow v2.0. Specifically, we designed a buffer commit method to increase the correction accuracy (challenge C3) and real time feedback to make the active text less distracting (challenge C2). We describe each improvement in detail as follows.

6.1 Buffer commit method

PhraseFlow v1.0 would commit all the active text and apply the phrase level candidates at once when the *n*th space was pressed, which caused the user to spend time reviewing the correction after the commit. The user also had to spend more cognitive resources to manage the results of space presses, since each space press behaves differently (some would commit corrections while the rest would not). In version 2, we designed a first-in-first-out buffer commit method as shown in Figure 5: when the *n*th space was pressed, only the first word in the active text would be committed with its correction candidate; the remaining text would stay active. For example, if n was set to 3, when the user typed *thid is the* and space, only *thid* was committed and corrected to *this*; the active text then became *is the* . In this way, there would always be a buffer in the active text, which enabled the decoder to correct text in a continuous manner. The space press behaves more consistently therefore causing no *surprise* to the user. Buffer style can also handle space related errors better than committing multiple words at once, as all space presses will be decoded within the buffer. A punctuation key press would commit and clear all remaining text in the buffer.

6.2 Real Time Correction Feedback

In the pilot study of PhraseFlow v1.0 we learnt that showing the raw text distracted the users and caused the uncertainty whether the text would be fixed later. We thus applied the partial decoding results to the active text: when the user typed a space, the active

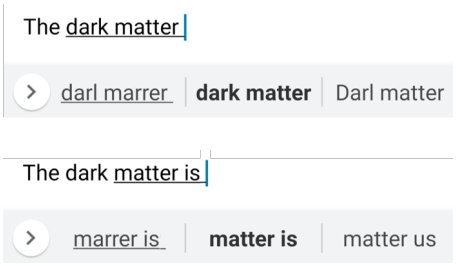

Figure 5: Buffer commit method in PhraseFlow v2.0. If the keyboard commits on every 3rd spaces, the upper figure shows the active text is *dark matter*, and the lower figure shows when the space is pressed after *is*, *dark* is committed (*"dark matter is"* already had 2 spaces)

text before the space would be updated to the suggested corrections, providing the real time correction feedback to the user. As shown in Figure 5, the raw text typed was *darl marrer*, but the active text was displayed as the correction. Since the correction was already shown as the active text, we only show the word-level candidates of the latest word in PhraseFlow v2.0 to make the suggestion bar more familiar to the users and less distracting.

6.3 Study 4: In-lab Study of PhraseFlow V2.0

We conducted an in-lab study to test out the performance of the modified designs. We recruited 12 people (6 male, 6 female). All of the participants were familiar with mobile text entry and could speak English fluently. They were also instructed to type in their preferred hand postures (11 used two-thumb posture, 1 used one thumb posture). We used a Pixel 3 XL for this study, and ran the same text editor application to conduct transcription tasks. Each participant was compensated with $25 for the one-hour study.

We compared two keyboards: Gboard and PhraseFlow v2.0. The buffer length n was set to 4 for the nth space buffer commit method. There were three identical parts of the study, with each part containing two transcription sessions with each keyboard. The order was balanced, and for each session, 20 different phrases from the Mackenzie phrase set were randomly selected. Before the formal sessions, participants practiced five phrases with each keyboard as a warm up. Participants were told to type as fast and accurately as possible, and that they could take a break between sessions. After the typing task, the participants filled out an SUS usability survey and a NASA-TLX survey [18] (for measuring the perceived workload). We also briefly interviewed the participants on their thoughts of using the two keyboards.

6.4 Results

In total, we tested on 20x2x3x12=1440 phrases. The results are shown in Figure 6. We analyzed all metrics using Wilcoxon signed-rank test rather than the potentially more sensitive parametric variance analysis. We observed that none of our performance metrics followed a normal distribution.

Speed There was no significant difference of keyboard on text entry speed ($p > .1$). The average speeds for Gboard and PhraseFlow were 63.8 wpm and 63.4 wpm.

Error Rate For Character Error Rate (CER), there was no significant difference between Gboard and PhraseFlow ($p > .1$). The average CER for Gboard was 0.017 while for PhraseFlow was 0.015. However, the Word Error Rate (WER) for PhraseFlow was significantly lower than Gboard ($p < .05$). The average WER was 0.052 for Gboard and was 0.042 for PhraseFlow, a 19.2% reduction.

Inter-key Interval (IKI) There was no significant difference on between-word IKI ($p > .1$) of the two keyboards, but PhraseFlow had significant higer in-word IKI than Gboard ($p < .05$). The average in-word IKI was 0.163 for Gboard and 0.166s for PhraseFlow; the average between-word IKI was 0.224s for Gboard and 0.223s for PhraseFlow. However, the difference of in-word IKI was smaller than the pilot study (0.003s vs 0.018s). Given the fact that the users had no experience in using phrase-level decoding keyboards in their daily life, participants performed pretty well on PhraseFlow. This also validated that our design of real time correction feedback reduced users' attention overload during typing.

Subjective Scores The SUS and TLX scores are shown in Figure 7, and the difference between the two keyboards was not significant for SUS ($p > .1$) or TLX($p > .1$). The median SUS score was 83.8 for Gboard and 80.0 for PhraseFlow, and the median SUS score was 3.4 for Gboard and 3.4 for PhraseFlow. We looked into the individual SUS scores, finding that five participants rated higher score for Gboard, five higher for PhraseFlow, and two rated the same scores, which meant the preferences among the participants were split.

6.5 Discussion of the In-lab Study

In comparison to the results of PhraseFlow V1 in Study 3, the results here were much improved. Even though PhraseFlow was a novel method, participants could type with it at about the same speed as with the Gboard word-level baseline, but made 19% less word errors (after autocorrection). The results suggest that the buffer commit method and the real time correction feedback in PhraseFlow v2.0 imposed lower cognitive load than the designs of PhraseFlow v1.0. The inter-word time interval was on par with word-level baseline. We coded the interview comments, finding that seven out of twelve participants commented that PhraseFlow had better correction accuracy than Gboard. We also found two main aspects of PhraseFlow that the participants still complained about: 1) The underlined active text was felt too long by some participants. As P6 commented, "*phrase level gives me more confidence, but I kept looking back to ensure it has the right suggestions*". 2) Manual correction was more difficult in PhraseFlow. When the previous typed text was not corrected or corrected to a wrong word, the participants had to move the cursor and manually correct the text, instead of just deleting with backspace key taps and retyping with Gboard.

We conducted an analysis on the second aspect in the study data using the transcription sequence log. If text before the last word was corrected but the space was not pressed, we counted that correction as a manual correction. We found in total 79 error instances that were corrected manually. Among them, 49 instances were autocorrection failures, *i.e.*, not corrected by the keyboard. For example, *wat* was not corrected to *war*, and *stattenmsy* to *statement*. This kind of error was caused by two overlapping reasons: the

Figure 6: The in-lab study results of Gboard and PhraseFlow. Error bars are one standard deviation

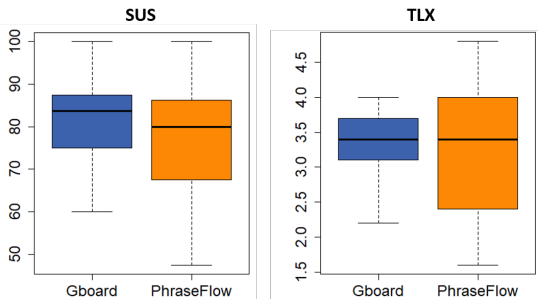

Figure 7: The SUS and TLX scores of the two keyboards. For SUS score, the higher the better; for TLX score, the lower the better

literal string typed was too far apart from the correct text, or the language model and decoding algorithm were not strong enough to handle the errors. Interestingly, there were also 30 instances that could have been corrected by the keyboard (as validated offline), but were manually corrected before the autocorrection. For example, *jo* to *no*, and *tat* to *that*. This showed that the participants did not trust enough on the correction ability of the keyboard, so they manually corrected the word while it was still in the active text.

These analyses suggest the active text span set in PhraseFlow 2.0 was too long, at least for the current scale and power of the language model used. We therefore decided to use smaller buffer size in the next study.

7 STUDY 5: DEPLOYMENT STUDY

The in-lab study results demonstrated that novice users could easily adopt PhraseFlow, and reached a similar level of speed performance of the word-level baseline of Gboard while benefiting from 19% fewer errors. However, transcription tasks as well as in-lab studies were artificial and constrained. To study PhraseFlow in daily tasks such as messaging, searching and writing emails, we conducted a 42 participants study in which we asked them to use PhraseFlow 2.0 Prototype as their main mobile keyboard for six days. We gathered their experience through surveys, which offered us further insights on what worked, what did not, and what future directions of phrase-level input should take.

7.1 Preparation

To prepare for the deployment study, we further improved PhraseFlow v2.0 prototype based on the in-lab study results. Specifically, we lowered n for the *n*th space commit method, and added personalization. The committing buffer length was decreased from committing at every 4th space to 2nd and 3rd to reduce the text area needed attention during typing. We ran the simulation test again using the touch points from the composition study, with the *n*th space set to 2, finding that the decoder had a WER of 6.47%, which had 16.7% relative error reduction compared to the 7.76% WER of Gboard. We also incorporated the personalization feature of Gboard in PhraseFlow. Personalization could learn the words that user typed and add them to the user-vocabulary, which was a necessary feature for daily text entry such as typing names, emails and abbreviations.

7.2 Participants

We posted the study description on several online forums, and received 150 responses, and contacted 58 participants who owned an Android phone, typed English, and knew how to install apk on their mobile phones. We further excluded 7 participants who used gesture typing exclusively in their daily mobile phone use, 1 participant who used multilingual keyboard, 1 participant who did not use autocorrection, and 7 participants who did not finish the keyboard apk installation step. We thus had 42 eligible participants (34 male, 8 female) for the study. All participants received a 25 USD gift card for the six-day deployment study.

7.3 Procedure

Before the deployment, we clearly expressed the goal of the study was to "evaluate an experimental feature of the keyboard for future improvement", and that participants should provide "objective and factual" evaluations in the later surveys, so as to minimize the experimenter demand effects [11]. We sent the participants the keyboard Android application Package (APK), together with explanations on the phrase-level decoding feature. We explicitly told the participants that we would not log any data from the keyboard to address their privacy concerns. To keep the study at a manageable length, we made the committing buffer length an between-subject variable by separating the participants into two groups: one with n set to 2 (committing at every 2nd space), and the other with n set to 3 (we refer to the conditions as *buffer-2* and *buffer-3* in the

following sections). There were 21 participants in each group. Participants were told to use PhraseFlow as their primary keyboard throughout the study. The study lasted for six days, with a survey sent out to the participants on the 2nd, 4th and the last day. The survey contained three questions asking for their perception and preference of PhraseFlow in comparison to previous word-level keyboard, and open-ended comments. All questions were shown in Table 4.

7.4 Results

On overall preference, most (78.6%) participants in the study liked (ratings of 4 and 5) the PhraseFlow prototype and a small number (7.1%) did not (ratings of 1 or 2). Table 5 gives more detailed rating statistics. The participants rated PhraseFlow positively on preference ("like to use", mean score ranging from 4.0 to 4.3 depending on days and the committing buffer length), perceived speed (mean score from 3.0 to 3.6) and perceived accuracy (mean score from 3.3 to 3.7). Note that a neutral 3.0 rating would match PhraseFlow to the participants' years of experience using word level keyboard in their everyday mobile interaction. Notably participants rated their overall preference of phrase level input higher than their perceived speed and accuracy improvement individually, suggesting their positive experience had more contributing factors than speed and accuracy alone.

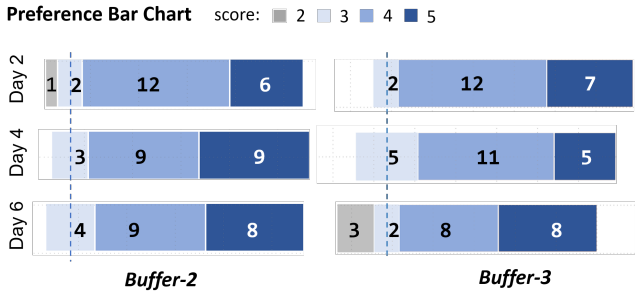

Figure 8: Distributions of PhraseFlow preference scores from 1 (*don't like it at all*) to 5 (*like it very much*). A score of 3 meant neutral opinion (represented with dashed lines)

Preference As illustrated in Figure 8, users' ratings of PhraseFlow varied with committing buffer length and over days of use. For example, on the overall preference ("like to use") and in the shorter committing buffer (*buffer-2*) group on Day 2, one user's rating was negative (2 on the scale of 1 to 5) , two neutral, 12 positive, and 6 very positive. On Day 4 and Day 6, the negative score in this group disappeared. The trend of the longer committing buffer (*buffer-3*) group, however, was the opposite. All users were positive or neutral on Day 2, but three turned negative on Day 6. User comments suggested "*false correction happens*" and "*backspace cannot revert the correction*" as the primary reasons for the decreased rating.

Speed According to Figure 9, both groups increased their scores gradually, especially for the *buffer-3* group. It appears that underlining the active text revealed the nature of *phrase-level correction* effectively enough to encourage user to type faster and trust the keyboard more during the study.

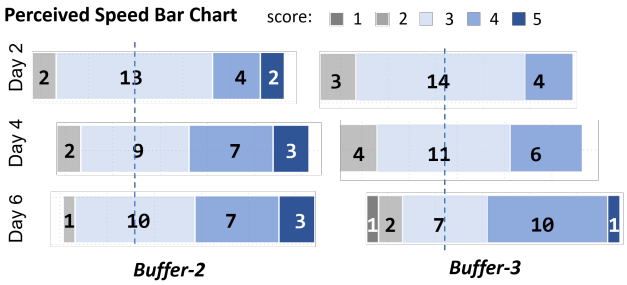

Figure 9: Distributions of speed responses from the three surveys. Compared to the word-level decoding keyboards the participants used before, PhraseFlow was rated from 1 (*much worse*) to 5 (*much better*). A score of 3 meant neutral opinion (represented with dashed lines)

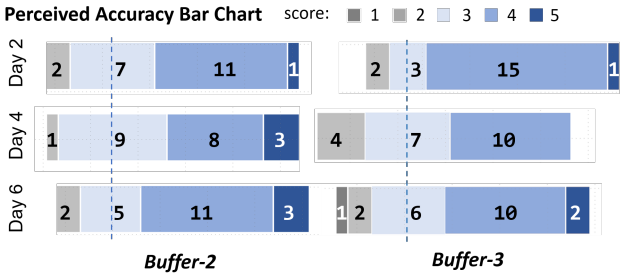

Figure 10: Distributions of accuracy responses from the three surveys. Compared to the word-level decoding keyboards the participants used before, PhraseFlow was rated from 1 (*much worse*) to 5 (*much better*). A score of 3 meant neutral opinion (represented with dashed lines)

Accuracy As shown in Figure 10, people in *buffer-2* group gradually increased their ratings from neutral to more accurate, or even much more accurate. However, people in *buffer-3* group decreased their scores, while they initially rated high accuracy in the first survey.

We analyzed users' comments using inductive coding. The major themes emerged included issues of *false corrections*, *correction frequencies* and *reverting support*. For false corrections, P3 mentioned that *but otherwise* was corrected to *bit otherwise*, and P18 commented that "*It's a little more cognitive overhead to go back to 3 words ago when there's a mistake, even though mistakes are less often.*" For correction frequencies, three participants mentioned that the phrase-level correction happened so few that they did not notice it, which also caused the drop of their accuracy scores in the n=3 group, as they expected initially that the keyboard would increase the correction accuracy a lot. Five participants also complained about that the backspace no longer reverted the correction, which was a feature offered by current Gboard. As P12 pointed out, "*Sometimes a previous word is corrected and it's not obvious how to change that word back easily.*"

Participants also mentioned that they got used to PhraseFlow after using the keyboard for a while. For example, P13 complained about the correction happened too frequent in the first two surveys,

Table 4: Survey questions for the deployment study

Survey Question	
Q1. Do you think you would like to use the "phrase level correction" feature in a future mobile keyboard?	Likert scale. From 1 (don't like it at all) to 5 (like it very much)
Q2. How do you think of your typing speed after using this keyboard with the new feature?	Likert scale. From 1 (much slower than before) to 5 (much faster than before)
Q3. How do you think of the accuracy of this keyboard with the new feature?	Likert scale. From 1 (much worse than before) to 5 (much better than before)
Q4. Do you have any comments when using PhraseFlow?	Open question

Table 5: The average scores ± 1 SD of the survey responses on preference, speed and accuracy of the PhraseFlow.

	Preference			Speed			Accuracy		
	Day2	Day4	Day6	Day2	Day4	Day6	Day2	Day4	Day6
buffer-2	4.1±0.8	4.3±0.7	4.2±0.7	3.3±0.8	3.5±0.9	3.6±0.8	3.5±0.7	3.6±0.8	3.7±0.8
buffer-3	4.2±0.6	4.0±0.7	4.0±1.0	3.0±0.6	3.1±0.6	3.4±0.8	3.7±0.7	3.3±0.8	3.5±1.0

but got used to it in the last survey: "*I have gotten more used to it auto correcting a text twice. Love the feature.*" Participants also worried about the inaccuracy when they type uncommon words in the first two surveys. As they continued using the keyboard, they reported that the experience was smoother because of the personalization feature. Although six days were long enough, five participants reported that they still corrected immediately once an error happened as with their previous keyboards. P25 commented that "*It's hard to suppress the urge to correct the first word, so I don't think I really get to the part where PhraseFlow does its best work.*"

8 DISCUSSION

Complementing the in-lab study, the deployment study results showed that despite being a novel user experience, the phrase-level decoding keyboard prototype PhraseFlow 2.0 was favorably received by most users during and after a few days of use. It appears the iterative design, development and research process has led to a version of phrase typing keyboard practical and beneficial enough as the primary keyboard in daily use by most users. In the following we drew further insights and lessons from the PhraseFlow project to inform future phrase level keyboard design. We also discuss the current limitations and future directions for the longer term.

8.1 Design Suggestions for Phrase-level Decoding Keyboard

Making the commit interaction consistent, and avoiding changing multiple words at a time. From the inter-key interval results, empirical studies showed that buffer commit method is better than committing multiple words at once. With the buffer commit method, the behavior of space press is consistent and predictable, which reduce the cognitive load of the user. It is also less burdensome to the user for not having to review the correction of multiple words. The simulation study also shows higher correction accuracy of the buffer style committing by allowing the decoder to take advantage of the subsequent context for each committed word.

Visual cues need to manage the user's attention judiciously. It is important to use visual cues to convey the states of the system (such as the active text), and notify when changes happen. This is consistent with recommendations in [3]. The choice of the visual effect also matter. Using underline to mark the span of the active text under decoding provides sufficient information of the range yet not too distracting (Figure 3(b)). On the other hand, correction effects should catch user's attention and thus a salient effect with abrupt onsets [20] need to be applied. In our study, background flash was proven noticeable and preferred by users (Figure 3(a)).

Making real time decoding progress transparent to the user. Instead of deferring the correction and displaying the raw text, the keyboard should display in real time decoded result on the typed text, even it is a partial result. In our studies, we found that when some users saw uncorrected raw text, they wanted to correct it right away, even they knew that the keyboard might correct it later. Showing the intermediate results reveals the decoding status of the keyboard, which increase the user confidence.

Minimizing false corrections. False correction happens when the correctly typed text is changed to a wrong correction, which will frustrates the user. In statistical decoding false correction is unavoidable, but its occurrence should be minimized. In addition to future language model quality improvement, consider the following ways to reduce false corrections: (1) raise the threshold of correcting an in-vocabulary word (2) enable personalization to handle out-of-vocabulary words.

Optimizing the decoding span length. While a longer span of active text provides the decoder with more context hence more correction information in principle [34, 35, 37], it is not the longer the active span the better. Even if a large n-gram language model approximately matches the user's input sequence, there is always a

chance for false-correction. Second, a longer span of active text imposes higher cognitive costs in terms of user attention and requires the trust from the user. The analysis of the in-lab study showed that users tended to immediately manually correct their errors before the decoder took actions. While the user generally focused on the latest word during typing [19, 32], paying attention to multiple words would also increase the cognitive load. Third, more distant errors that are uncorrected or falsely corrected imposes higher manual correction cost. We recommend the active text span be limited to two or three words, at least for the current mobile grade language models. Importantly, even an active decoding span of two words offers many fundamental benefits of phrase level typing identified earlier in the paper, including better ability to correct space key errors.

8.2 Future Directions of Phrase Level Typing

Larger and more accurate language models are needed to support higher quality phrase-level decoding. Traditional word-level decoding primarily depends on unigram and bigram language models. Phrase-level decoding requires trigram and even higher ngram models to prevent frequent back-off to unigrams and bigrams which by definition do not maintain longer distance word connectivity. However, increasing frequent high order ngram grows the model size exponentially. Neural network language models trained by deep learning techniques such as the smart compose model [10] is promising at modelling longer distance language context and may become practical on mobile devices memory and compute wise in the near future.

Understanding and designing win-loss ratio of phrase level typing. We expect future research would prove that larger and more powerful language models would enable longer buffer streams and stronger corrections, which in principle allows even faster and less precise typing at the same or better output text accuracy. However the user attention and manual correction cost of a longer span false correction could be nonlinear to the buffer length. Understanding the tradeoff and design the right win-loss criteria (i.e. how much accuracy boost should be gained before expanding to a longer span) is another important future research direction of phrase level typing.

Efficient error correction is needed to revert false correction. In the deployment study, we received many comments about the manual error correction in general and the inconvenience of lacking an reverting interaction in particular. The cost of manually modifying the phrase-level correction is high, thus it is important to offer the user the ability to easily revert the correction. For Gboard, pressing backspace immediately after a space press will revert the correction. However, as the buffer commit method would only commit the first word in the active text, assigning backspace as the reverting key conflicts with deleting characters in the active text.

8.3 Limitations

Although both in the in-lab study and the deployment study gave evidence of benefit to PhraseFlow V2,there are undoubtedly important limitations to the PhraseFlow project. Our search of effective phrase typing solutions is by no means exhaustive. There could be similar or even better UI and interaction designs than what we iterated to in PhraseFlow V2. One limitation of the deployment study was that we did not log quantitative data because of the privacy concern, which prevented us from gaining more insights on the realistic usage. PhraseFlow mainly focused on tap typing in English language. Hence, the lessons learnt from PhraseFlow might not apply to all languages. For some languages, the relevant context might be three or more words away, which requires larger language models. For languages like Chinese, phrase-level decoding is already incorporated into the Pinyin typing methods, but after the decoding, extra steps are required to commit the candidates. During our study, some participants suggested it would be desirable if the keyboard could correct multiple languages together, which was also beyond the scope of this paper. Finally, we started but did not get very far in phrase level gesture typing. Gesture typing decodes at whole word level to begin with [41] and does not have the error prone space key operations for both space insertion and word commit.

9 CONCLUSION

In this paper, we present research on phrase level typing based on multiple versions of the PhraseFlow keyboard prototype. The contribution is threefold: 1) We implemented a practical phrase-level decoder based on a widely used keyboard, and validated that phrase-level decoding had higher accuracy on correction tasks than word-level decoding, including space-related errors; 2) Through a iterative process, we designed visual feedback effects and interaction methods that successfully reduced the cognitive load of phrase level input; 3) Through the in-lab and deployment studies, we identified the cognitive and behavior patterns when people using PhraseFlow, and demonstrated the feasibility of the phrase-level decoding in real typing settings. As powerful machine learning techniques are boosting the field of natural language processing, we expect text input to significantly improve for everyday typing experience. While there are still improvements to make a phrase-level keyboard qualitatively better, PhraseFlow may provide a new baseline towards that goal.

ACKNOWLEDGMENTS

This work was based on the first author's research internship at Google which was co-hosted by Scott Jenson. The research was built on the codebase of Gboard. We thank many Google researchers and Gboard engineers, particularly, Jianpeng Hou, Yanchao Su, Yuanbo Zhang, and Vlad Schogol for their support, technical guidance, and collaboration.

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 Graphics Interface 2019* (Kingston, Ontario) *(GI 2019)*. Canadian Information Processing Society, Kingston, Ontario, 10 pages.

[2] Rui A. Alves and Teresa Limpo. 2015. Progress in Written Language Bursts, Pauses, Transcription, and Written Composition Across Schooling. *Scientific Studies of Reading* 19, 5 (2015), 374–391. https://app.dimensions.ai/details/publication/pub.1046423967

[3] Saleema Amershi, Dan Weld, Mihaela Vorvoreanu, Adam Fourney, Besmira Nushi, Penny Collisson, Jina Suh, Shamsi Iqbal, Paul N. Bennett, Kori Inkpen, Jaime Teevan, Ruth Kikin-Gil, and Eric Horvitz. 2019. Guidelines for Human-AI Interaction. In *Proceedings of the 2019 CHI Conference on Human Factors in*

Computing Systems (Glasgow, Scotland Uk) *(CHI '19)*. Association for Computing Machinery, New York, NY, USA, 1–13.

[4] Kenneth C. Arnold, Krzysztof Z. Gajos, and Adam T. Kalai. 2016. On Suggesting Phrases vs. Predicting Words for Mobile Text Composition. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology* (Tokyo, Japan) *(UIST '16)*. Association for Computing Machinery, New York, NY, USA, 603–608.

[5] Shiri Azenkot and Shumin Zhai. 2012. Touch Behavior with Different Postures on Soft Smartphone Keyboards. 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, 251–260.

[6] Nikola Banovic, Ticha Sethapakdi, Yasasvi Hari, Anind K. Dey, and Jennifer Mankoff. 2019. The Limits of Expert Text Entry Speed on Mobile Keyboards with Autocorrect. In *Proceedings of the 21st International Conference on Human-Computer Interaction with Mobile Devices and Services* (Taipei, Taiwan) *(MobileHCI '19)*. Association for Computing Machinery, New York, NY, USA, 1-11 pages.

[7] Xiaojun Bi, Shiri Azenkot, Kurt Partridge, and Shumin Zhai. 2013. Octopus: evaluating touchscreen keyboard correction and recognition algorithms via. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. Association for Computing Machinery, New York, NY, USA, 543–552.

[8] 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.

[9] Ciprian Chelba, Mohammad Norouzi, and Samy Bengio. 2017. N-gram Language Modeling using Recurrent Neural Network Estimation. *arXiv preprint cs* 1 (2017), 10 pages. arXiv:1703.10724v2 [cs.CL]

[10] Mia Xu Chen, Benjamin N. Lee, Gagan Bansal, Yuan Cao, Shuyuan Zhang, Justin Lu, Jackie Tsay, Yinan Wang, Andrew M. Dai, Zhifeng Chen, Timothy Sohn, and Yonghui Wu. 2019. Gmail Smart Compose: Real-Time Assisted Writing. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (Anchorage, AK, USA) *(KDD '19)*. Association for Computing Machinery, New York, NY, USA, 2287–2295.

[11] Nicola Dell, Vidya Vaidyanathan, Indrani Medhi, Edward Cutrell, and William Thies. 2012. "Yours is Better!": Participant Response Bias in HCI. 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, 1321–1330.

[12] Vivek Dhakal, Anna Maria Feit, Per Ola Kristensson, and Antti Oulasvirta. 2018. Observations on Typing from 136 Million Keystrokes. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems* (Montreal QC, Canada) *(CHI '18)*. Association for Computing Machinery, New York, NY, USA, Article 646, 1-11 pages.

[13] Leah Findlater and Jacob Wobbrock. 2012. Personalized Input: Improving Ten-Finger Touchscreen Typing through Automatic Adaptation. 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, 815–824.

[14] Andrew Fowler, Kurt Partridge, Ciprian Chelba, Xiaojun Bi, Tom Ouyang, and Shumin Zhai. 2015. Effects of Language Modeling and Its Personalization on Touchscreen Typing Performance. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems* (Seoul, Republic of Korea) *(CHI '15)*. Association for Computing Machinery, New York, NY, USA, 649–658.

[15] Mayank Goel, Leah Findlater, and Jacob Wobbrock. 2012. WalkType: Using Accelerometer Data to Accomodate Situational Impairments in Mobile Touch Screen Text Entry. 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, 2687–2696.

[16] 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)*. Association for Computing Machinery, New York, NY, USA, 194–195.

[17] Google. 2020. *Gboard Android Play Store Page*. Google. https://play.google.com/store/apps/details?id=com.google.android.inputmethod.latin

[18] Sandra G. Hart and Lowell E. Staveland. 1988. Development of NASA-TLX (Task Load Index): Results of Empirical and Theoretical Research. In *Human Mental Workload*, Peter A. Hancock and Najmedin Meshkati (Eds.). Advances in Psychology, Vol. 52. North-Holland, New York, 139 – 183.

[19] B. E. John and A. Newell. 1989. Cumulating the Science of HCI: From s-R Compatibility to Transcription Typing. *SIGCHI Bull.* 20, SI (March 1989), 109–114.

[20] John Jonides and Steven Yantis. 1988. Uniqueness of abrupt visual onset in capturing attention. *Perception & Psychophysics* 43 (1988), 346–354.

[21] Slava Katz. 1987. Estimation of probabilities from sparse data for the language model component of a speech recognizer. *IEEE transactions on acoustics, speech, and signal processing* 35, 3 (1987), 400–401.

[22] VI Levenshtein. 1966. Binary Codes Capable of Correcting Deletions, Insertions and Reversals. In *Soviet Physics Doklady*, Vol. 10. Springer, Soviet, 707.

[23] I. Scott MacKenzie. 2015. A Note on Calculating Text Entry Speed. https://www.yorku.ca/mack/RN-TextEntrySpeed.html

[24] I. Scott MacKenzie and R. William Soukoreff. 2003. Phrase Sets for Evaluating Text Entry Techniques. In *CHI '03 Extended Abstracts on Human Factors in Computing Systems* (Ft. Lauderdale, Florida, USA) *(CHI EA '03)*. Association for Computing Machinery, New York, NY, USA, 754–755.

[25] Srdan Medimorec and Evan F. Risko. 2017. Pauses in written composition: on the importance of where writers pause. *Reading and Writing* 30 (2017), 1267–1285.

[26] Microsoft. 2020. *SwiftKey Homepage*. Microsoft. https://www.microsoft.com/en-us/swiftkey

[27] G. A. Miller. 1956. The magical number seven plus or minus two: some limits on our capacity for processing information. *Psychological review* 63 2 (1956), 81–97.

[28] Nielsen Norman. 2020. 10 Usability Heuristics for User Interface Design. https://www.nngroup.com/articles/ten-usability-heuristics/

[29] Tom Ouyang, David Rybach, Françoise Beaufays, and Michael Riley. 2017. Mobile Keyboard Input Decoding with Finite-State Transducers. arXiv:1704.03987 [cs.CL]

[30] Richard W Pew. 1966. Acquisition of hierarchical control over the temporal organization of a skill. *Journal of experimental psychology* 71, 5 (1966), 764.

[31] Philip Quinn and Shumin Zhai. 2016. A Cost-Benefit Study of Text Entry Suggestion Interaction. 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, 83–88.

[32] Timothy Salthouse. 1986. Perceptual, Cognitive, and Motoric Aspects of Transcription Typing. *Psychological bulletin* 99 (06 1986), 303–19.

[33] John Sweller. 1988. Cognitive load during problem solving: Effects on learning. *Cognitive Science* 12, 2 (1988), 257 – 285.

[34] Keith Vertanen, Crystal Fletcher, Dylan Gaines, Jacob Gould, and Per Ola Kristensson. 2018. The Impact of Word, Multiple Word, and Sentence Input on Virtual Keyboard Decoding Performance. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems* (Montreal QC, Canada) *(CHI '18)*. Association for Computing Machinery, New York, NY, USA, 1–12.

[35] Keith Vertanen, Dylan Gaines, Crystal Fletcher, Alex M. Stanage, Robbie Watling, and Per Ola Kristensson. 2019. VelociWatch: Designing and Evaluating a Virtual Keyboard for the Input of Challenging Text. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems* (Glasgow, Scotland Uk) *(CHI '19)*. Association for Computing Machinery, New York, NY, USA, 1–14.

[36] Keith Vertanen and Per Ola Kristensson. 2014. Complementing Text Entry Evaluations with a Composition Task. *ACM Trans. Comput.-Hum. Interact.* 21, 2, Article 8 (Feb. 2014), 10 pages.

[37] 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)*. Association for Computing Machinery, New York, NY, USA, 659–668.

[38] Daryl Weir, Henning Pohl, Simon Rogers, Keith Vertanen, and Per Ola Kristensson. 2014. Uncertain Text Entry on Mobile Devices. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Toronto, Ontario, Canada) *(CHI '14)*. Association for Computing Machinery, New York, NY, USA, 2307–2316.

[39] Christopher D Wickens, Justin G Hollands, Simon Banbury, and Raja Parasuraman. 2015. *Engineering psychology and human performance*. Psychology Press, New York.

[40] Ying Yin, Tom Yu Ouyang, Kurt Partridge, and Shumin Zhai. 2013. Making Touchscreen Keyboards Adaptive to Keys, Hand Postures, and Individuals: A Hierarchical Spatial Backoff Model Approach. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Paris, France) *(CHI '13)*. Association for Computing Machinery, New York, NY, USA, 2775–2784.

[41] Shumin Zhai and Per Ola Kristensson. 2012. The Word-Gesture Keyboard: Reimagining Keyboard Interaction. *Commun. ACM* 55, 9 (Sept. 2012), 91–101.

[42] Mingrui Ray Zhang and Jacob O. Wobbrock. 2019. Beyond the Input Stream: Making Text Entry Evaluations More Flexible with Transcription Sequences. 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, 831–842.

[43] Mingrui Ray Zhang, Shumin Zhai, and Jacob O. Wobbrock. 2019. Text Entry Throughput: Towards Unifying Speed and Accuracy in a Single Performance Metric. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems* (Glasgow, Scotland Uk) *(CHI '19)*. Association for Computing Machinery, New York, NY, USA, Article 636, 11-22 pages.

[44] Suwen Zhu, Tianyao Luo, Xiaojun Bi, and Shumin Zhai. 2018. Typing on an Invisible Keyboard. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems* (Montreal QC, Canada) *(CHI '18)*. Association for Computing Machinery, New York, NY, USA, 1–13.