C-PAK: Correcting and Completing Variable-length Prefix-based Abbreviated Keystrokes

TIANSHI LI, Carnegie Mellon University, USA
PHILIP QUINN, Google, USA
SHUMIN ZHAI, Google, USA

Improving keystroke savings is a long-term goal of text input research. We present a study into the design space of an abbreviated style of text input called *C-PAK* (Correcting and completing variable-length Prefix-based Abbreviated Keystrokes) for text entry on mobile devices. Given a variable length and potentially inaccurate input string (e.g. "li g t m"), C-PAK aims to expand it into a complete phrase (e.g. "looks good to me"). We develop a C-PAK prototype keyboard, *PhraseWriter*, based on a current state-of-the-art mobile keyboard consisting of 1.3 million n-grams and 164,000 words. Using computational simulations on a large dataset of realistic input text, we found that, in comparison to conventional single-word suggestions, PhraseWriter improves the maximum keystroke savings rate by 6.7% (from 46.3% to 49.4%), reduces the word error rate by 14.7%, and is particularly advantageous for common phrases. We conducted a lab study of novice user behavior and performance which found that users could quickly utilize the C-PAK style abbreviations implemented in PhraseWriter, achieving a higher keystroke savings rate than forward suggestions (25% vs. 16%). Furthermore, they intuitively and successfully abbreviated more with common phrases. However, users had a lower overall text entry rate due to their limited experience with the system (28.5 words per minute vs. 37.7). We outline future technical directions to improve C-PAK over the PhraseWriter baseline, and further opportunities to study the perceptual, cognitive, and physical action trade-offs that underlie the learning curve of C-PAK systems.

CCS Concepts: • **Human-centered computing** → **Text input**; • **Information systems** → *Language models.*

Additional Key Words and Phrases: Mobile Text Entry; Typing Suggestions and Predictions; Abbreviated Phrase Input.

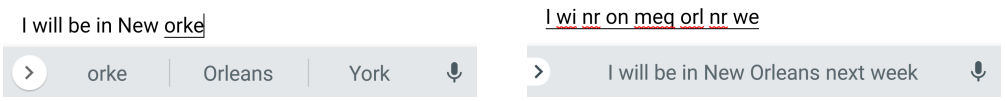
ACM Reference Format:
Tianshi Li, Philip Quinn, and Shumin Zhai. 2022. C-PAK: Correcting and Completing Variable-length Prefix-based Abbreviated Keystrokes. *[Final draft | Official copy to appear in] ACM Transactions on Computer-Human Interaction*
https://doi.org/10.1145/3544101

1 INTRODUCTION

A major goal of text input research is to reduce the number of keystrokes needed to enter text – predominantly achieved by presenting suggestions to the user from their partial input strings. Reducing keystrokes can be particularly valuable for users who experience a relatively high motor cost when entering text (e.g. motor-impaired users), and lessens the need for precise motor control and spelling.

Authors' addresses: Tianshi Li, Carnegie Mellon University, Pittsburgh, PA, USA, tianshil@cs.cmu.edu; Philip Quinn, Google, Mountain View, CA, USA, philip@quinn.gen.nz; Shumin Zhai, Google, Mountain View, CA, USA, zhai@acm.org.

1073-0516/2022/1-ART1
https://doi.org/10.1145/3544101



(a) Correcting and completing a word ("Orleans") with forward suggestions.

(b) Completing a partial sentence with multiple errors using C-PAK.

Fig. 1. A demonstration of partial input with forward suggestions (left) and C-PAK (right) for the sentence "I will be in New Orleans next week".

The keystroke saving functions of conventional mobile keyboards are limited to short-span *forward* suggestions, such as single word completion and next word prediction. In text input research another type of keystroke reduction had been extensively studied: *abbreviated phrase input* [1, 12, 30, 36, 37, 50, 51]. These methods allow a user to omit or change characters in the middle of a word, such as typing "I wil C U l8r" to enter "I will see you later". The most common abbreviation rules include omitting vowels and repeated consonant letters. While there is a significant amount of research (reviewed in the following section) on these abbreviation methods, they have not gained adoption in practice.

In this paper, we re-examine the abbreviated phrase input problem for the English language by studying a different style of abbreviation, called *C-PAK*: Correcting and completing variable length Prefix-based Abbreviated Keystrokes. This method is inspired by a feature that has received widespread adoption in Chinese input methods (desktop and mobile), but has received little formal research. Known as *Jianpin*, it allows a user to enter a varied number of *Pinyin* character prefixes to enter Chinese characters. Similarly, C-PAK allows a user's input to be a sequence of variable-length English word prefixes (which may contain spatial or spelling errors). For example, to enter "Have a nice weekend", a user may type only the initials of each word, "H a n w", a complete input that contains errors, "Habe a noce wekend", or any combination of prefixes (e.g. "Hab a nice w"). Users can retain their existing typing habits established on a conventional keyboard, but gain the flexibility to choose a smaller number of characters to type for each word (e.g. Figure 1). Furthermore, C-PAK has a much higher theoretical upper bound of keystroke savings than the conventional abbreviation rules that have been studied before. For instance, the upper bound of keystroke savings for C-PAK is 62.3%, while the upper bound for omitting all vowels (if not the first letter) and consecutive duplicate consonants is only 31.8% (measured on the Enron corpus in Section 3.2.1).

C-PAK research cannot be done in the abstract because the performance, behavior, and success of C-PAK depends on the technologies that power its decoder, and the user interface and interaction design that mediates the decoder's suggestions to the user. We therefore created *PhraseWriter*, a specific implementation of C-PAK based on today's mobile keyboard technologies, and conducted a set of studies using this implementation of C-PAK. The PhraseWriter project makes the following contributions to the research of C-PAK input:

(1) *N-gram-based C-PAK implementation.* The completion and correction power of C-PAK depends on its underlying language models and decoding algorithms. PhraseWriter bases its decoding on an n-gram language model [16, 21, 40, 45, 48]. Although not as powerful and large as neural network language models [e.g. 20, 44], n-gram-based decoding is much more powerful than lexicon-based decoding algorithms commonly used in abbreviation systems [e.g. 33]. N-gram language models are also small and efficient enough to execute on mobile devices, and therefore are practical to embed in keyboard products.

(2) *Computational studies of n-gram-based C-PAK implementation.* In order to measure the capabilities and limitations of the PhraseWriter implementation for C-PAK we conducted simulations based on realistic content (the Enron email corpus [22]) and user input behavior (Section 5). The simulations enabled us to estimate a set of performance limits that a user could achieve with PhraseWriter in several special boundary conditions. In an error-free abbreviated input condition, PhraseWriter could improve the keystroke savings rate by 6.7% in comparison to conventional single-word forward suggestions (from 46.3% to 49.4%; selecting from the top three candidates). In a complete but noisy input condition (typing all letters in each word but allowing for typos), PhraseWriter could decrease the word-level error rate by 14.7% (from 4.08% to 3.48%; selecting only the top candidate).

(3) *UI design.* The user interfaces of current mobile keyboards are designed with only a narrow space for word-level alternatives (typically three, as shown in Figure 1a). However, C-PAK works at a phrase level and phrase-level suggestions demand more space – introducing new interaction design challenges (e.g. a mechanism for revising incorrectly-predicted words within a suggested phrase). We explore the design space of user input and interactions (Section 3) and settle on a specific set of design choices for the *PhraseWriter* keyboard that conservatively models after current mobile keyboard designs (Figure 1b).

(4) *User behavior study.* C-PAK encourages longer span input with a different risk–reward profile to conventional forward-suggestions: a longer phrase with fewer letters per word can produce greater keystroke savings and lessen the burden of complete and correct spelling if the abbreviated input is successfully decoded to the intended phrase. However, a higher cost of revision is incurred if the input is decoded to a different phrase than what was intended. The optimal abbreviation for each intended phrase may demand a long learning curve to find. How users deal with such statistical uncertainty for greater potential reward in C-PAK has not been studied before. We conducted a lab study of the PhraseWriter keyboard with 14 experienced mobile keyboard users (but *novice* C-PAK users) to observe their early typing behavior against a conventional word-level forward suggestion keyboard. We found the participants could immediately take advantage of the keystroke savings offered by the PhraseWriter keyboard in their first session of use, achieving an average keystroke saving rate of 25% (vs. 16% on a conventional keyboard). Specifically, they intuitively and successfully abbreviated more with common phrases. At this early stage of learning C-PAK, the users' strategy was *conservative* and their overall text entry speed was slower (28.5 words per minute vs. 37.7; Section 6).

The PhraseWriter prototype keyboard, its decoding algorithms, and the two studies allow us to understand the potential of C-PAK and identify the gap between novice users' performance and expert users' ideal performance. We outline potential design and technical improvement directions for future research to help users take full advantage of C-PAK, such as providing better error correction support for multi-word input, allowing partial commits to reduce the risk of missing correct suggestions, and taking semantics and grammar into account when making suggestions.

2 RELATED WORK

2.1 Word Completion and Forward Suggestion

A key objective of our work on C-PAK is to minimize the number of keystrokes needed to enter complete text. This objective is shared with previous work on word completion and forward suggestion.

The earliest and simplest method of saving keystrokes is lexicon-based one-word completion: as the user types prefix characters of a word, the number of candidate words that match the prefix

string reduces – eventually to a small enough number that can be displayed and selected by a user [11, 24, 33, 34]. For example, if the user types "ack", the system could present "acknowledge", "acknowledging", "acknowledged", and "acknowledgement" as completion suggestions. The coverage of such completion systems depends on the size of the lexicon. More advanced word-level *n*-gram language models can enable *next* word prediction. For example, if the user types "how are", a system may be able to predict "how are you", "how are things", or "how are you doing" as likely phrase completions. Statistically, the larger the *n*-gram model (e.g. larger values of *n*), the stronger and more accurate the predictions will be.

Input completion is possible because of language redundancy. Past work has either attempted to estimate the upper bound of possible keystroke savings due to this redundancy theoretically [e.g. 10, 43], or conducted simulation studies to empirically measure the best keystroke savings on noisy input that can be achieved by a mobile keyboard decoder [e.g. 16]. The best model in studies conducted by Fowler et al. [16] achieved a 45.8% keystroke savings rate on the Enron corpus.

There is also research in understanding how forward suggestions are used in practice, with a common theme that the savings in motor actions do not necessarily lead to an improvement in text entry speed. Augmentative and Alternative Communication (AAC) research uses completion/prediction to improve input speed for users with physical impairments – but the benefits of suggestions are not always clear [17, 42]. Koester and Levine [23, 24, 25] suggested that the cognitive and motor costs of suggestion interfaces may outweigh their benefits, and their evaluations of several AAC systems found either marginal time performance gains for novice users, or time performance loss for experienced users. For general users, Quinn and Zhai [32] found that using word completion lowers text entry speed, but is subjectively preferred. These findings imply that speed is not the only measure for evaluating the text entry experience, and keystroke savings or a reduction of motor actions may have user experience benefits, even if they were slower.

In addition to short-span completion/correction, recent work has demonstrated the viability of multi-word prediction based on preceding context. These systems rely on larger and more powerful language models (e.g. neural networks) trained on larger datasets of constrained vocabularies [9].

2.2 Conventional Abbreviated Phrase-level Input

Extensive research has studied phrase-level abbreviated input for both AAC systems [1, 12, 50, 51] and general text entry systems [30, 36, 37]. Demasco and McCoy [12] presented a system that allows users to omit some words while typing out others completely, and then expands the entered keywords into a grammatical and semantically appropriate sentence (e.g. expanding "apple eat John" into "the apple is eaten by John"). Like many other projects on abbreviated input, they found a benefit in the reduction of motor movement, but at the cognitive cost of learning and memorization.

Many abbreviation systems use ad-hoc letter-level omission rules drawn from human-generated (crowdsourcing) abbreviations. Willis et al. [50, 51] recruited 21 participants to abbreviate sentences at three levels (short, medium, and long), and identified common abbreviation techniques such as vowel deletion and phonetic replacement – which they used to design an abbreviation expansion system. Their system could decode 96% of the words (found in top 5 suggestions) at a keystroke saving rate of 26%, and could decode 90% of the words at a keystroke saving rate of 39%. These results were generated using a 10,000 word dictionary.[1] They also performed experiments with 2,000, 20,000, and 30,000 word dictionaries. As expected, the precision of their algorithm decreased with the size of the dictionary. They noted that "the location of the correct word in the list of candidates went down substantially, to the extent that with 20,000 words, the process of selection of the correct words would probably negate any keystroke saving" [51, p. 6].

[1]In contrast, PhraseWriter's vocabulary consists of 164,000 words.

Shieber and Baker [36] studied a simple stipulated model of abbreviation: words could be created by (1) omitting all vowels, except when a vowel is the first letter of a word; and (2) omitting consecutive duplicate consonants (e.g. "admission" would be entered as "admsn"). Their decoding algorithm could achieve a 26.5% theoretical keystroke savings on the *Wall Street Journal* corpus with this style of input. Shieber and Nelken [37] studied a similar system using a mouse-operated graphical keyboard – achieving a 12.24% speed improvement over a baseline of clicking on every letter. The system required users to adopt a fairly rigid abbreviation system and their evaluations did not consider error correction that is necessary on mobile touch screen devices.

Pini et al. [30] built a mobile keyboard that could automatically detect ad-hoc phrase abbreviations using an SVM predictor and presented expansion suggestions generated using a Hidden Markov Model. The SVM predictor was trained on human-generated word abbreviations collected on Amazon Mechanical Turk and regular words from the British National Corpus. They achieved an accuracy of 89.7% in the best case. Their expander was trained on the Enron email dataset and evaluated using a sample of the dataset abbreviated using three different rules: (1) omitting all vowels and consecutive duplicate consonants (if not the first character); (2) keeping the first two or three letters; and (3) the first rule with additional omission of common English prepositions. The decoding accuracy (top 4 choices considered) was 93.7%, 87.2%, and 69.8% for the three abbreviation rules, respectively. They further showed in a user study of participants entering 12 short phrases that users could achieve a 32% keystroke savings, while their text entry speed decreased by 25%. After one-hour practice, four users achieved a time saving of 26%.

More recently, Adhikary et al. [1] trained a neural language model to expand noisy abbreviated input when users omit all spaces and randomly drop mid-word vowels. Their model achieved an 8.0% character-level error rate when dropping all vowels, and a keystroke savings rate of 38.2% in a simulation study. They then built a touchscreen keyboard with a decoder running on a remote server. In a user study, the participants were requested to dwell for one second on each key to simulate motor-impairment users who experience higher motor cost. After some practice, the participants achieved a slightly lower speed in the sentence completion mode than the word completion mode (9.6 WPM vs. 9.9 WPM) and the character error rate (CER) of the sentence mode was higher (7.2% vs. 0.3%).

These abbreviation methods have not been adopted in mainstream practice. Unlike prior work that only studied how people abbreviated phrases when the abbreviation task was salient [50, 51] or only conducted simulation studies to evaluate the performance of their decoding algorithms [12, 36, 51], we conduct both computational simulation studies and lab-based text entry user studies to evaluate our system in this paper. Furthermore, we employ a more ecologically valid study design which reveals more types of cognitive cost when performing abbreviated phrase-level input than prior work. For example, Pini et al. [30] only asked participants to enter very short phrases (i.e. 2–5 words), which unrealistically reduced the cognitive cost of deciding on the commit unit. In this paper, we use relatively long sentences as the target input (at least 3 words and at most 12 words, with a median length of 5 words). Some work did not always consider the error correction time [1, 37], while we not only count the error correction time when measuring the performance, but also contextualize the error correction cost in a detailed breakdown of time spent on different actions. Shieber and Nelken [37] and Adhikary et al. [1] forced the decoding event to happen only at the end of a sentence, while we allow users to freely choose to what extent they abbreviate phrases and when they make commits. Adhikary et al. [1] forced a one-second dwell time between adjacent actions to make normal users behave like users with motor impairments, which artificially reduced the gap in the average keystroke preparation time between the word mode and sentence mode. We did not have any restrictions on how to use the keyboard during the study.

2.3 C-PAK style Abbreviation in Chinese Input

For Chinese input, a version of C-PAK called *Jiǎnpīn* (简拼) is prevalent in practical *Pinyin*-based Chinese keyboards. *Jianpin* allows users to enter prefixes of the *Pinyin* transliterations of each Chinese character (e.g. "jp" for "jian-pin") in a flexible and systematic fashion. When these variable-length prefix strings of a Chinese word (typically consisting of more than one Chinese character) or a phrase are concatenated, the intended word or phrase can often be algorithmically predicted. The predicated words and phrases are presented to the user in a keyboard's suggestion bar. The key advantages of *Jianpin* are the high reduction of keystrokes that it offers, and the reduced demand for accurate spelling. For example, to enter the phrase "Chinese history" (中国历史), instead of typing the full *Pinyin* string *zhong-guo-li-shi*, the user can type *zgls* (the first *Pinyin* letter of each Chinese character) and select the top suggestion candidate "中国历史". However, the challenge of *Jianpin* is the high degree of ambiguity in these prefixes. For example, if the user wanted to enter 英国历史 (*ying-guo-li-shi* – "English history") by typing *ygli*, reasonable suggestions may include 一个类似 (*yi-ge-lei-shi* – "a similar…") and 雁过留声 (*yan-guo-liu-sheng* – a common idiom), which are quite distinct from the intended text. When this happens, the user needs to back-track and re-enter the full *Pinyin* string for some or all of the characters. No empirical studies have been reported in the literature on the cognitive aspects of *Jianpin* in terms of immediate or long term learning.

In early systems, simple look-up tables were used to decode these abbreviations into Chinese characters [49], but modern systems use statistical methods capable of handling varied amounts of abbreviation. Although little academic research has been published on *Pinyin* input, considerable industry innovation has taken place due to the one-to-many mapping from *Pinyin* syllables to Chinese characters.

Although little formal research is reported in the literature, *Jianpin* is a strong existence proof of a C-PAK style of input – at least for non-European languages. Our own informal investigation and experience tells us that the greater keystroke savings and reduced burden of correct *Pinyin* spelling does attract a large of amount of use – despite the greater risks of revision when *Jianpin* decoding fails to produce the intended characters. However, there are important differences between Chinese and European writing systems that may change the strength of an input method from one language to another. For example, English uses spaces as word delimiters whereas Chinese character sequences are contiguous across word boundaries. Such differences require different design considerations in both decoding and user interaction.

2.4 Error Correction

Error correction is an essential capability for mobile touch-based keyboards due to the lack of tactile feedback and motor stability. This capability is deeply embedded in keyboard decoders, which leverage spatial modeling [e.g. 4, 15, 52, 57] and language modeling [e.g. 8, 21] techniques to correctly decode noisy touch events into the user's intended input [e.g. 16, 18]. Practical keyboards also model spelling errors by adding letter insertion and deletion probability estimates in their decoding algorithms [e.g. 28]. Current mobile keyboards provide the resulting error correction capability in two forms: (1) *correction suggestions*: presenting corrected text as the top-*n* suggestions for a user to select, and (2) *automatic correction*: automatically correcting high-probability errors in the most-recently entered word after the user types a space or punctuation key.

Although longer context can potentially improve the performance of error correction, current keyboards limit the correction unit to a single word. Only a few special cases provide correction over word boundaries (such as correcting "new york" to "New York"). This is primarily because a user's mental model of keyboard behavior is that each word is committed after entering the space

key, and changes to text that has already been committed is therefore disruptive and unexpected. In contrast, C-PAK allows users to commit a variable number of words at a time, so users can benefit from the better error correction capability due to the longer context. The longer correction span may also raise greater perceptual and cognitive burden in user's evaluation and verification.

2.5 Phrase-level Text Input

C-PAK operates at phrase level (i.e. multiple words). This is in contrast to the basic cadence of today's smartphone keyboard that operates at a word-level: a user performs a sequence of movement inputs on the keyboard, and upon entering a space (a word delimiter), these inputs are decoded and *committed* as a word to an output window. Once a word is committed it is generally no longer open to correction – even if future context provides crucial new information about those earlier words. For example, consider the two phrases: "where are you from?" and "what are you doing?". At the point of decoding "wh", the ambiguity of "where" versus "what" cannot be resolved until the subsequent context (i.e. "from" vs. "doing") becomes available.

Prior to a space being entered the word entered will be in an "uncommitted" state until a selection is made from the suggestion bar above the keyboard. If a space is entered, the most likely suggestion will be committed. Some current keyboards (e.g. Google Gboard) have a "space omission" feature that allows users to enter multiple words at a time (although only reliably for common short phrases). For example, "howareyou" can be decoded into "how are you".

Researchers have attempted to extend the input cadence of touch keyboards from words to phrases. For example, Vertanen et al. [48] studied *complete* (unabbreviated) sentence level input as the fundamental input unit by (1) employing a large language model, executed on an external server; and (2) exploring different types of feedback, word boundary delimiters (e.g. a space key tap or a right swipe gesture), and commit gestures (e.g. a right or a up swipe gesture). Testing against common, memorable short phrases from the Enron dataset [47], they demonstrated that the large server-based decoder had a lower error rate than using Google Gboard's "space omission" feature to enter short phrases that were entered with similar speeds. Later, Vertanen et al. [45] studied decoding noisy input from a smartwatch at one-word, two-word, and sentence levels – finding that sentence-level input resulted in much lower error rates in an offline analysis, but with diminishing returns as the size of the language model increased.

Although the findings in Vertanen et al. [45, 48] may be dependent on the specific form factors (smartwatches), tasks (short, memorable phrases), and design choices (committing with swipe gestures), these studies on sentence level input provide evidence that there is a potential benefit in encouraging users to consider input in longer spans.

Zhang and Zhai [56] studied an input keyboard that was able to correct fully typed words based on their subsequent input in the phrase. Their simulation studies (using "remulation" – a replay of recorded touch input on different algorithms [7]) showed an over 16% error reduction from the word-level baseline. However, they observed increased cognitive load in monitoring phrase level changes that could hinder time performance.

3 THE DESIGN SPACE OF C-PAK

In this section we review the design choices of C-PAK with respect to the three key design dimensions of abbreviated phrase input – modeling, interaction, and interface. For model design, we discuss the special requirements of C-PAK and its effects on the choice of language model and decoding algorithm. For interaction design, we discuss the coverage of the underlying decoder dictionary/corpus, and the architecture of the partial input string. For user interface design, we discuss variations of suggestion presentation, suggestion commitment, and input editing.

3.1 Language Modeling and Decoding

An appropriate language model and decoding algorithm for C-PAK has two high-level requirements:

(1) It needs to make accurate predictions of a user's intended phrase. This is more challenging than predicting a singular intended word because the number of possible word sequences increases exponentially with the sequence length (*the curse of dimensionality*).
(2) It needs to run on mobile devices that have limited computing resources, while maintaining real-time performance and high accuracy.

A decoding algorithm converts raw input signals (e.g. touch points) into text. This typically combines the preceding context as represented by a language model, and the user's input for the current word as related to a spatial model, into the most likely word candidates – presented to the user in a suggestion bar. Upon detecting a word completion signal such as a space key or a tap on a candidate word in the suggestion bar, the most likely candidate is committed to the application window. Thereafter the committed word becomes the most recent part of preceding text, and the input, decoding, and commit process repeats for the next word.

In C-PAK the input unit – partial inputs of multiple words – and the corresponding decoding target is extended across multiple words to a phrase or even a sentence, which could either be treated holistically (e.g. a trained neural network on phrase-level input) or formed piece-wise from word prefixes (e.g. identifying the input for each word to complete word-level decoding, and then building phrase-/sentence-level output using a beam search).

N-gram language models are the status quo in modern mobile keyboards, representing sequences of text using $(n-1)$-order Markov chains, and computing the probability of a candidate word given the preceding $n-1$ words as they appear in a corpus. Such models are simple to develop, and feature both low computational requirements and decent performance [16]. However, a weakness of n-gram models is their inability to capture long context. As n grows, the possible number of n-gram combinations grows exponentially. In practice this means n cannot be very large – typically less than five – and n-gram language models for mobile input methods must go through intensive pruning to fit into the memory of mobile devices, using techniques such as Katz back-off [e.g. 21] or entropy pruning [e.g. 38]. A pruned n-gram language model contains only the most frequent higher-order n-grams seen in the training corpus. At run time, the probabilities of the pruned-out, less frequent or never seen n-grams in the training corpus are estimated by backing off progressively to available lower-order n-grams (e.g. bi-grams or even uni-grams). Although long phrase predictions can be made by concatenating short n-grams, the precise long span context is lost.

Recent research has found that neural-network-based language models have the potential to be used to maintain a longer context span [6, 8, 27, 39]. Specifically, models that consider bi-directional context (e.g. BERT [20]) may be suitable for C-PAK so that context from the end of a sentence can be used to complete abbreviations at the beginning. However, executing neural networks on mobile devices requires significant computational resources. Current commercial keyboards limit their use of neural-network models to features such as language identification [41] and spatial decoding [5], while still using traditional n-gram models for language modeling.

As the starting point of systematically studying C-PAK, we focus on n-gram models broadly deployed on mobile devices with limited computational resources despite their known limitations. We believe that if C-PAK can be proven practical on a limited amount of computational resources, larger and more powerful language models would only make it more effective.

3.2 Interaction Design

The interaction design of C-PAK concerns how users form phrase-level abbreviations (the *input string architecture*), and the expressions they can produce from those abbreviations (the *coverage*).

Rule	KSR
Omitting vowels and repeated consonant [36, 37]	31.8%
Omitting half vowels and spaces [1]	34.0%
Omitting vowels and spaces [1]	49.1%
Only retaining the first letter and spaces (C-PAK)	62.3%

Table 1. The theoretical maximum keystroke saving rate (KSR) for various abbreviation rules on the Enron dataset.

Although the most flexible system would allow users to write any arbitrary abbreviation to produce any text, this may not be practical to implement or use.

3.2.1 Input String Architecture. C-PAK is a prefix-based style of phrase abbreviation – namely, the abbreviation consists of a sequence of variable-length English word prefixes (as short as the initial letter, or as long as the complete word). This method gives users flexibility in choosing how much to abbreviate each word and potentially allows for more keystroke savings than conventional abbreviations such as the omission of vowels and repeated consonants.

A fundamental motivation for studying C-PAK input lies in its high performance ceiling in terms of keystroke savings. We measured the theoretical maximal keystroke saving rates that are offered by C-PAK in comparison with other abbreviation methods on the Enron dataset (the same dataset for the simulation studies in Section 5), and show the results in Table 1. Note that we did not take into account the performance limit of a decoder in this measurement – that is, we simply measured the highest keystroke saving rate that could be achieved by a certain abbreviation rule, and allows for potential decoding error.[2]

Expanding prefix-based word completion to phrase completion introduces a question of how these prefixes are formed and joined. Two key dimensions of this are (1) if the abbreviations are fixed in length and (2) whether they contain an explicit word delimiter. Using the phrase "looks good to me" as an example, the following illustrates a few possible designs—

- Prefixes composed of initials (fixed length) with no word delimiters: "lgtm".
- Variable-length prefixes with no word delimiters: "logtme".
- Variable-length prefixes with word delimiters: "lo g t me".

Fixed-length designs allow for more keystrokes to be saved through the omission of word delimiters across the phrase. If we assume each word contains an average of five letters [35] and only the initials are entered, the upper bound of keystrokes saved when a word delimiter is required is $1 - \frac{1+1}{5+1} = 66.7\%$, versus $1 - \frac{1}{5+1} = 83.3\%$ when the delimiter can be omitted. However, fixed-length input may fail to decode correctly when the words in a phrase have different lengths, low usage frequencies, or complex inflections. For example, given the intended sentence "we need to find the minimum set", an all-initial abbreviation "w n t f t m s" may be able to decode the former part of the sentence "we need to find the" because it contains only common short words and frequent n-grams, while the latter part "minimum set" has a longer word "minimum" and a more specific meaning, for which the initials do not provide sufficient information. In addition, if the chosen length is larger than one (or is variable), words shorter than that length become ambiguous with the adjacent characters. Consequently, fixed-length word abbreviations are more likely to be compatible with closed set coverage (discussed below).

[2]In Section 5.3 we further show that an n-gram-based C-PAK decoder can achieve a higher keystroke saving rate than the theoretical maxima of the conventional abbreviation methods, and correctly decode all input in the meantime.

When including word delimiters, a phrase can consist of word abbreviations with variable lengths. These may be simple prefixes of each word, or some other method of abbreviation (e.g. omitting vowels). This allows users to leverage their own knowledge and judgement to moderate the amount of information provided to the system in order to accommodate the different regularities of words or phrases, but by effectively adding an extra keystroke to each word (the delimiter).

3.2.2 Coverage. There are two primary styles of phrase coverage to consider: *open set* and *closed set*. With open-set coverage, any phrase consisting of words in a corpus can be completed with an appropriate abbreviation. Conversely, with closed-set coverage, only phrases that exist in a restricted phrase-level vocabulary can be entered (e.g. the most popular phrases in a corpus or personal history).

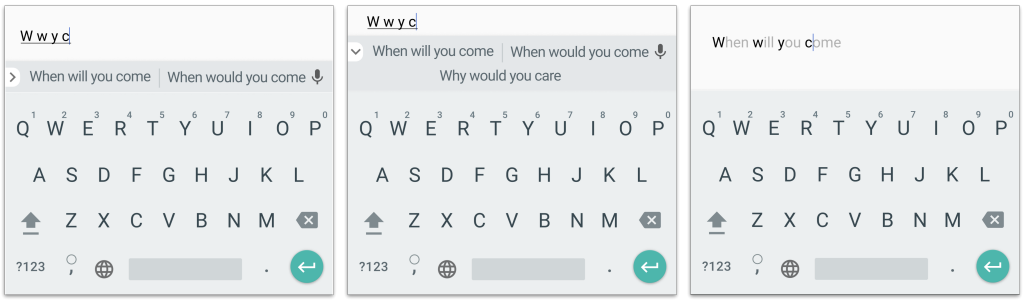
Open-set coverage grants users the most flexibility in their input, but necessitates longer abbreviations due to the increased ambiguity of the input. For example, the abbreviation consisting of initials "w w y g" can result in multiple decoding results with similar probabilities: "when will you go", "where will you go", "where were you guys", etc. Users would therefore need to invent their own abbreviations concurrently with their entry, and moderate the length of those abbreviations based on the regularity of the intended phrase and other contextual factors (such as the accuracy of their input).

Closed-set coverage can achieve greater motor cost savings by using shorter or fixed abbreviations, but at the cost of limiting users to a set of predefined phrases they must remember. To accomplish general input, a closed-set system must be combined with a conventional text entry system.

3.3 User Interface Design

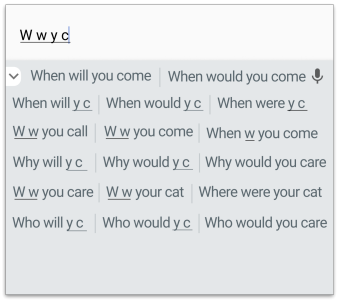
The user interface design of an C-PAK system needs to both display the state of the system (e.g. presenting alternate suggestions) and allow the user to interact with that state (e.g. selecting suggestions or correcting errors). This design needs to be made in consideration of the cognitive overhead that a user faces in learning the system, creating phrase abbreviations, and identifying and correcting errors. The UI design also needs to consider committing actions that change the composed phrase from a writing state to a committed state.

3.3.1 Suggestion Presentation. Phrase suggestions are likely to contain a variable number of words and may grow to be quite long – which creates challenges for presenting them in the limited screen space of mobile devices. This is especially difficult when there is ambiguity in the entered text that demands multiple phrase-level candidates to be presented.

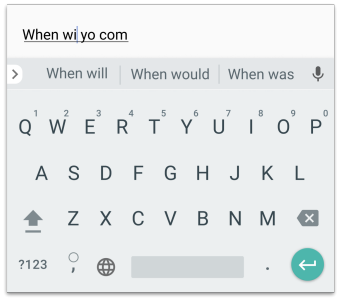


(a) Presented in a one-line bar. (b) An expandable suggestion bar. (c) Inlined with the text.

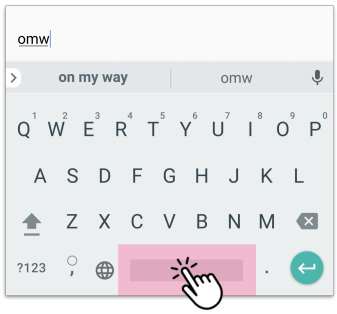
Fig. 2. Design variants regarding suggestion presentation.



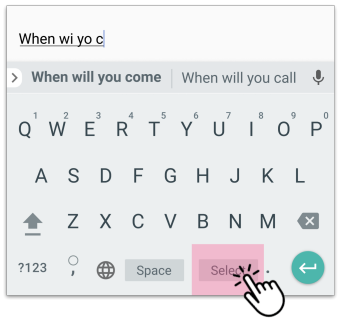
(a) Partial candidates presented when expanding the suggestion bar.



(b) Partial candidates presented when moving the cursor.



(c) Tapping on the spacebar to select a candidate (when spaces are omitted).



(d) Tapping on a separate commit key to select a candidate.

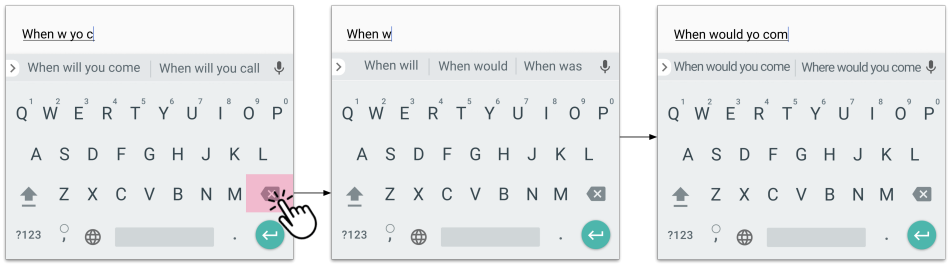
Fig. 3. Design variants regarding selecting and committing suggestions.

Current practice with word-level suggestions uses a one-line bar to display a fixed number of candidates. This could be repurposed for phrase-level suggestions while retaining a fixed-size bar (Figure 2a), or expanding the bar to show a fixed number of candidates (Figure 2b). Some recent interfaces (e.g. SmartCompose [9]) place suggestions inline with the text output field in the application area, which could be adjusted to show a single phrase suggestion (Figure 2c). This is a more assertive design as it is harder for the user to overlook the suggestions, and is therefore more intrusive than displaying them in a separate suggestion bar. It is also difficult to display more than one candidate inline when the composed text wraps at the right edge of the screen.

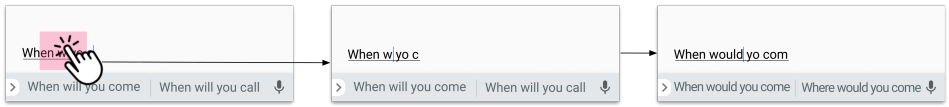
Although a fixed-size bar limits the visual intrusion of the interface, it is likely to only have enough space to show one or two phrase-level candidates. As longer input has more possible decoding results, this increases the risk of missing the intended phrase and discouraging users from using the system. Allowing the suggestion bar to expand permits more candidates to be shown – and therefore increases the probability of a correct candidate being shown – but at the cost of additional visual space and perceptual cost to the user.

3.3.2 Committing Suggestions. Although some current interaction practices can be retained in C-PAK, such as tapping on a suggestion to explicitly select and commit it, there are also some unique design problems: partial commits and implicit commits.

A multi-word suggestion may be partially correct, so being able to commit only its correct portions is an important feature (in addition to committing the entire phrase). A partial commit could either be triggered by a user selecting which part they want to commit (Figure 3b), or



(a) The user removes all characters after "When w" (left) in order to complete "would" (middle) before completing the rest of the phrase (right).



(b) The user can reposition the cursor (left) and complete the word "would" (middle) without disrupting the following text (right).

Fig. 4. Design variants for editing text before committing. The user intends to write "When would you come" from the partial input "When w yo c". However, more context is needed to disambiguate "would" from "will".

facilitated by a mix of complete and partial commit suggestions (Figure 3a – similar to current Chinese keyboards). The former offers more flexibility, but requires the selection of small targets to identify the range of text to commit.

When a system has sufficiently high confidence in a particular suggestion, it may implicitly commit it – similar to how auto-correction and auto-completion features in current keyboards operate when the a user taps on the space bar (Figure 3c). However, for phrase abbreviations created with explicit delimiters, space taps cannot be used to trigger an implicit commit. Our user studies suggest that this may impact the performance of faster typists more since they heavily rely on auto-correction (see Section 6.5.6). An alternative may be to split the space bar into two parts: one for entering spaces, and one for selecting the most confident suggestion (Figure 3d).

3.3.3 Editing Uncommitted Input. Given suggestions that are only partially correct, users may want to edit their input before or after selecting a suggestion. The current practice with word-level suggestion is generally to delete all of the text following error, correct the error, and then re-type the text (although users can precisely position the cursor to only delete the error, this is rare). For C-PAK, this will demand a user deletes and re-types a larger portion of text (Figure 4a). An alternative design would allow the user to position the insertion cursor within their abbreviated input and make a correction without disrupting the surrounding text (e.g. Figure 4b).

Users could also accept a suggestion first, and then fix the error. The most straightforward design is to treat the committed text as normal text, and to offer suggestions beginning with the committed word, as shown in Figure 5a. However, the example in Figure 5 also demonstrates that the committed word ("will") could be different from the intended word ("would"). Therefore, the alternative suggestions could be based on the context of the entire phrase or sentence, or the previous partial input, rather than the committed word (Figure 5b).

(a) The user accepts an incorrect suggestion (left) and repositions the cursor to the error (middle) to trigger alternate prefix suggestions for that word (right).

(b) The user accepts an incorrect suggestion (left) and repositions the cursor to the error (middle) to trigger alternate suggestions for that word based on the previous partial input (right).

Fig. 5. Design variants for editing text after a commit. The intended sentence is: "When would you come".

4 THE PHRASEWRITER DECODER

We built the PhraseWriter decoder to support a particular instance of the C-PAK interaction design space described above, focusing on abbreviation strategies where (1) words are delimited by a space and (2) abbreviations are of a variable length.

The decoder was built upon a production mobile keyboard, Google Gboard. Gboard's language model, consisting 164,000 unigrams and 1.3 million n-grams (with n ranging from 2 to 5). Its decoding algorithms and user interface have been tuned for and used by hundreds of millions of users. Building a C-PAK decoder on Gboard allowed us to study an example of the C-PAK method against a strong baseline with the same practical language model and lower-level algorithms.

The input to the decoding algorithm is a literal string, which is split into a sequence of n prefixes by spaces (which are assumed to be correctly placed): $p_1, \ldots, p_n$, where p_i represents the i-th prefix. Note that although we use the term "prefix" here, it may be as short as one letter or as long as a complete word.

Given a prefix sequence, the algorithm iterates through it to decode each prefix p_i into a word w_i. The goal is to find a sequence of words $\widehat{\mathbf{W}}$ that minimizes a cost function B given the literal input $p_1, \ldots, p_n$:

$$\widehat{\mathbf{W}} = \underset{w_1,\ldots,w_n}{\arg\min}\, B(w_1, \ldots, w_n | p_1, \ldots, p_n) \tag{1}$$

$$B(w_1, \ldots, w_n | p_1, \ldots, p_n) = \sum_{i=1}^{n} \left[L(w_1, \ldots, w_i) + S(w_i, p_i) \right] \tag{2}$$

$$L(w_1, \ldots, w_i) = -\log \left[P(w_i | w_1, \ldots, w_{i-1}) \right] \tag{3}$$

$L(w_1, \ldots, w_i)$ is the language score when decoding the word w_i if the preceding input is $w_1, \ldots, w_{i-1}$ (trained on a corpus of text), and $S(w_i, p_i)$ is the spatial score of the word–prefix pair (w_i, p_i) – detailed below.

We used a weighted finite-state transducer [28] to generate a list of word candidates for each prefix, and calculated a spatial score for each word–prefix pair. We used the existing n-gram ($n \leq 5$) language model that had already been carefully pruned to run efficiently on mobile devices, and supported back-off: when an n-gram did not exist in the vocabulary, its linguistic probability was estimated by the lower-order n-grams within it. To make the search process tractable [16] we adopted beam search pruning using a beam width of 20 (i.e. retaining the top 20 phrase candidates at each step).

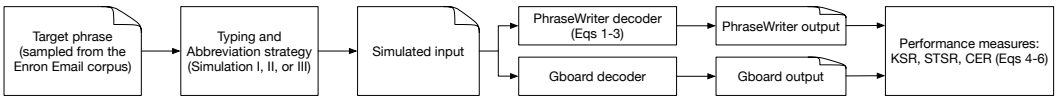


Fig. 6. Simulation study overview: The simulated input is generated using a target phrase from the Enron Email corpus; the C-PAK decoder is built atop the decoder of a commercial keyboard, Google Gboard.

The spatial score $S(w_i, p_i)$ was calculated by finding the minimum editing cost required to convert a prefix p_i into a candidate word w_i. This cost is derived from two types of editing actions: (1) insertion, deletion, and substitution of characters to correct spatial errors in the input, and (2) the cost of expansion to complete a word (e.g. "iss" into "issue"). These costs were fixed constants for each type of action. Note that substitution errors include proximity substitutions due to "fat finger" errors, and the cost was calculated accordingly. For example "t" in "H at you doimg" is a substitution for "r" of this kind.

It is important to note that the expansion cost to complete a word should be relatively small, but above zero. A small cost encourages the algorithm to expand short literal inputs into longer words as candidates, and making it greater than zero reduces the risk of unwanted completions – especially when the input is a word that is also a prefix of a longer word (e.g. "the" and "their", "in" and "inside"). The design consideration here is in ensuring that the algorithm prioritizes the literal input (e.g. "the" and "in") in the suggestions presented to the user, since a false-positive expansion can be significantly more frustrating than a false-negative.

5 SIMULATIONS STUDIES

We ran the PhraseWriter decoder described in the previous section on a corpus of sentences by simulating various touch input behaviors, and levels of abbreviation. The purpose of these studies was to understand the statistical characteristics of PhraseWriter as an instance of a C-PAK decoder, and to establish its basic motor action and error rate reduction limits in comparison to a baseline technique. We ran three simulations each focused on a special boundary condition of input behavior, with the first two simulations corresponding to two types of typing habits. These studies highlight the PhraseWriter decoder's maximum benefits on these boundary conditions. The following section examines the extent to which these benefits were realized in its first hour of use.

The simulations were run on a sentence set sampled from the Enron email corpus [22]. The corpus consists of real-world human communication and can be used with minimal privacy concerns. We randomly sampled 10% of the email bodies, segmented each body into sentences by punctuation characters surrounded by at least one space, and kept sentences that contained 2–10 words. All text was considered without casing. This yielded a corpus of 15,087 sentences containing 86,411 words (7,737 unique). Figure 6 illustrates the structure and process of our simulation studies.

5.1 Performance measures

5.1.1 Motor action savings. We used the keystroke savings rate (KSR; Equation 4) and the suggestion-tap savings rate (STSR; Equation 5) to measure motor action savings. KSR is a common metric of motor cost saving in the keyboard research literature that measures the proportion of taps that can be elided using a particular prediction technique [16, 43]. We considered taps on letter keys, the spacebar, and suggestions equivalently.

$$KSR = 1 - \frac{\text{total taps to enter the decoded text}}{\text{total taps to enter the reference text}} \tag{4}$$

STSR (Equation 5) reflects the savings enabled by the capability of committing text at phrase level – that is, the reduction in the number of suggestion taps per word of input. With conventional text entry, users commit one word at a time by tapping on the correct suggestion if they want to use automatic completion. However, PhraseWriter allows users to enter multiple words in a single commit. This does not reduce the total number of keystrokes required to enter the text (a tap on the spacebar is substituted with a tap on a suggestion), but avoids a context switch from the user which may present a larger cost [32, 45].

$$STSR = 1 - \frac{\text{selection taps with PhraseWriter}}{\text{selection taps with the baseline}} \tag{5}$$

5.1.2 Error rate. The character error rate (CER; Equation 6) measures the number of insertions, substitutions, and deletions (the Levenshtein distance) required to transform the text output from the decoder into the reference text, divided by number of characters in the reference text – excluding changes in case.

$$CER = \frac{\text{CharacterDistance}(\text{decoded}, \text{reference})}{\text{CharacterLength}(\text{reference})} \tag{6}$$

The word error rate (WER; Equation 7) measures word-level edit distance between the reference text and the decoded text, divided by number of words in the reference text – excluding changes in case.

$$WER = \frac{\text{WordDistance}(\text{decoded}, \text{reference})}{\text{WordLength}(\text{reference})} \tag{7}$$

5.2 Baseline

We compared the PhraseWriter decoder against the decoder of the same keyboard product that it was constructed on. Both decoders therefore used the same language model and finite-state transducer for word-level predictions, and all spatial parameters were shared. This removes the quality of the decoder's language model as a potential confound.

5.3 Simulation I: Motor Action Savings with Error-Free Partial Input

In order to discover the upper bound of the keystroke savings of PhraseWriter as a C-PAK decoder, we simulated an error-free partial input condition to find the minimum number of keystrokes required to correctly decode each sentence with each decoder. This reflects a typing strategy of accurately tapping each key (i.e. within its key boundary), and selects from the top three suggestions when they match with the user's intention. The user is assumed to evaluate the suggested candidates as frequently as needed.

For the baseline, it is straightforward to find the minimum number of keystrokes required: each letter is progressively typed within its boundary. Whenever one of the top three suggestions matches the intended word, it is selected.

For the C-PAK condition, we need to find the minimum set of prefix combinations that can produce the intended phrase, for which the number of possible combinations compounds exponentially with the number of words in the phrase. To do this we executed an exhaustive search to find the shortest partial input for the sentences in the corpus: for each sentence, we started with the initials of each word and progressively enumerated all possible prefix combinations until the first correct prediction was found in the top three suggestions. We cached the intermediate results and used heuristic pruning to accelerate the prefix enumeration and verification process.

Sometimes splitting a phrase into smaller commit units can further save keystrokes. For example, for the phrase "when will you come" the partial input "w w y c" has too many legitimate suggestions to be presented (e.g. "when would/will you come/call"); however, if the input is divided into "when

will" with "w w" and "you come" with "y c", the phrase can be entered successfully with just its initials. This does not affect the number of keystrokes required to enter the text (as taps on the spacebar have simply been substituted with taps on suggestions), but it requires the user to have a certain commit strategy.

We used a greedy algorithm to find these strategies and approximate the smallest number of suggestion taps needed for a particular set of prefix combinations. The algorithm tried to find the longest sequence of prefixes that could correctly complete part of the corresponding intended text, and then repeated this search until the entire phrase had been completed. This ensured that we could find the near-optimal suggestion-tap savings, while also achieving the best keystroke savings. However, users cannot be expected to know the optimal set of prefix combinations and commit units for all sentences. In practice, they may start to learn optimal prefixes for the most common phrases (e.g. "looks good to me" or "be right there") and expand from there as their expertise develops (discussed later).

5.3.1 Results. The best keystroke saving rate of the baseline decoder was 46.3%. In comparison, the best keystroke saving rate of the PhraseWriter decoder was 49.4% – a 6.7% improvement over the baseline.

The suggestion-tap saving rate (STSR) of the PhraseWriter decoder was 52.6%, with an average of 2.39 words entered per commit (rather than only 1 with the baseline decoder). When only considering sentences with at least four words, the suggestion-tap saving rate is raised to 58.9%, with an average of 2.70 words entered per commit.

5.3.2 Case Analysis. The following example shows how the two decoders saved keystrokes and suggestion taps differently. The text in black represents the characters actually typed (the literal input), and the brackets indicate one commit via a suggestion tap. To enter the sentence "I'll be at your place on Thursday to see her", the input to the baseline decoder that maximizes keystroke savings is:

```
[I'll] [be] [at] [your] [place] [on] [Thursday] [to] [see] [her]
```

That is, entering the first one or two characters of each word and choosing the correct suggestion. In contrast, the input to the PhraseWriter decoder that maximizes keystroke savings is:

```
[I'll be at your place on Thursday to see] [her]
```

That is, entering the first character of the first nine words before choosing a completion, and then entering the first letter of the final word and choosing a completion.

For this example, both PhraseWriter and the baseline decoder achieve substantial keystroke savings in this ideal situation (PhraseWriter saves one more keystroke). However, the cost of evaluating and selecting suggestions differs. The PhraseWriter decoder can complete the sentence with only two commits, and in the first commit it successfully completes nine words with only its initials entered. Conversely, with the baseline decoder users have to choose from three candidates and make explicit selections for each word. This suggests that it is not sufficient to judge the potential motor cost savings by only referring to the keystroke saving rate, and PhraseWriter can potentially reduce the cost of switching between text entry and suggestion selection.

Overall, Simulation I shows that in error free conditions, PhraseWriter can increase keystroke savings from the baseline's 46.3% to 49.4%, and provides a suggestion-tap saving rate of 52.6%, using exactly the same language model.

5.4 Simulation II: Error Correction with Noisy and Complete Input

While Simulation I showed the maximum keystroke and selection tap saving, this simulation measures PhrasedWriter's correction ability in a different boundary condition - all characters are entered so the language model's power is entirely focused on error correction. In contrast to

conventional word-level touch keyboards that decode one word at a time, PhraseWriter can take input at a phrase level. Intuitively, a greater bidirectional phrase level context could correct more errors from noisy input in touchscreen typing given the additional context from words across the phrase. In a very different simulation set-up, Vertanen and colleagues [45, 48] have shown their (complete) phrase level decoding could reduce CER from 2.3% to 1.8%. The current simulation studies the amount of error reduction the PhraseWriter decoder yields, as compared to a single-word forward-decoding baseline. Note that this single-word forward-decoding baseline still used the proceeding words up to the current word as context in decoding.

We made this measurement by simulating noisy (or "sloppy"), but complete, unabbreviated input. The simulated strategy is also a common one: typing quickly and imprecisely, paying minimal attention to suggestions, and relying on the correction capabilities of the decoder. Specifically, we simulated taps on the keyboard driven by a noisy input model such that they may fall outside the target key boundary.

For the baseline, the top suggestion was accepted after each word (i.e. on the space between words). For PhraseWriter, the top suggestion was accepted after the entire sentence was typed. In both cases, the entire sentence was typed without abbreviation.

The noisy input was generated from a Gaussian model with the same distribution parameters as Fowler et al. [16] to reflect human-like typing behavior with a 9.8% error rate, as studied in Azenkot and Zhai [4]. We did not simulate insertion or deletion errors.

5.4.1 Results. The CER of the baseline decoder was 1.76%, versus 1.53% for PhraseWriter. Although the baseline result was already very strong in comparison with prior studies [16], the PhraseWriter decoder still decreased the character error rate by 13.1%. The WER of the baseline decoder was 4.08%, versus 3.48% for PhraseWriter (a decrease of 14.7%).

As another reference, the raw error rate without correction (i.e. the noisy input generated using a Gaussian model) would be 9.63% CER and 39.10% WER.

5.4.2 Case Analysis. The following examples show a target sentence, an example of the characters typed (given the noisy input model), and the output from the baseline and PhraseWriter decoders. The errors in the noisy input and errors in the decoded outputs are underlined.

Target	`is it possible to have an answer by this afternoon`
Typed	`us ur ppssible to hsve wn anawer by this afternoon`
Baseline	`us ur possible to have an answer by this afternoon`
PhraseWriter	`is it possible to have an answer by this afternoon`

This example illustrates a case where the baseline erred when the PhraseWriter decoder did not: a spatial error at the beginning of the sentence that happened to convert the intended input to another common word ("is" → "us"). The incorrect inference on the first word further increased the difficulty of decoding the second word correctly using the baseline. However, the PhraseWriter decoder performed better due to the re-ranking of candidates based on the subsequent context.

The following example shows a case where both decoders made errors:

Target	`we will need the gas by next march or april`
Typed	`ww will need the gaa bt next mafch or aoril`
Baseline	`we will need the gas but next match or april`
PhraseWriter	`we will need the gas by next match or april`

In this example, the PhraseWriter decoder corrected "bt" into "by" correctly when the baseline decoder mistakenly corrected it into "but". Both "but" and "by" are reasonable suggestions given the previous context, but with the benefit of the subsequent context it is obvious that "by" makes more sense than "but". However, both decoders failed to correct "mafch" to "march". This example

Frequency	Coverage	Count	Examples
≥ 5 users	70.75%	77,928	enron will have; as we move forward; make a reservation
≥ 20 users	54.07%	9,594	continuing to; let me know who; pleased with
≥ 50 users	37.49%	1,404	sent me; north america; for enron; are trying to
≥ 80 users	20.99%	184	we need to; the last; talk to; thanks for the; we should

Table 2. Common phrase coverage and examples. The coverage is the percentage of words that can form a common phrase with adjacent words. For example, given the target sentence "let me know if you have any questions", both "let me know if" and "if you" are common phrases with more than 80 users – but the word "if" is only counted once when calculating the coverage.

shows that although the future context carries sufficient information, the current language model was not powerful enough to always make semantically meaningful suggestions.

Overall, Simulation II shows the bilateral sentence level decoding by PhraseWriter can improve WER from the word-level decoding baseline by 14.7%, as illustrated in the specific case analyses, even though both are powered by the same n-gram language model.

5.5 Simulation III: First Letter Noisy Input vs. Phrase Commonality

This simulation measures PhraseWriter's phrase completion ability under another boundary condition: for each word in a phrase, only the first character is entered (e.g. "l g t m" → "looks good to me"). Given an n-gram language model's pruning and back-off algorithms [e.g. 21], we expect that common phrases can be more accurately completed from their prefixes. We ran the first initial-character-only prefix input against sentences with different levels of commonality. Each sentence was 10 words long to ensure a reasonable challenge for the decoding task given a 5-gram language model.

To establish a list of common phrases independently, we used the number of individuals ($n = 90$) in the corpus who used a phrase as a measure of its commonality, rather than referring to the language score calculated by the language model. All sequences of between 2 and 10 consecutive words appearing in the corpus were considered phrases. We selected phrases that were used by at least 5, 20, 50, and 80 people to represent four levels of phrase commonality. Table 2 shows the coverage of these phrase levels – the percentage of words that can form a common phrase with adjacent words.

Although the number of unique common phrases is not high, the high coverage reflects a frequent use of them. If the PhraseWriter decoder can complete these phrases, it suggests that users could achieve considerable savings by focusing on a limited set of common phrases.

We simulated the input of each phrase using the noisy input model described for Simulation II. Only the first character of each word was input (including for the baseline decoder) and the top suggestion was selected after each step.

5.5.1 Results. We calculated the keystroke savings rate and character error rate on different subsets of the corpus: (1) all text, (2) common phrases (four levels of commonality), and (3) uncommon expressions (text not appearing in any common phrase). Each subset is tied to a different commonality, and introduces a different view of the results.

The PhraseWriter decoder consistently performed better on all subsets of the dataset over the baseline in terms of error rate (Table 3), and the benefits of PhraseWriter were more salient for common phrases.

	Baseline	PW	Δ	KSR
Uncommon text	73.89%	73.18%	0.71pp	69.34%
All text	57.61%	51.58%	6.03pp	61.58%
Common phrases (≥ 5 users)	54.35%	45.79%	8.56pp	57.73%
Common phrases (≥ 20 users)	52.06%	41.80%	10.26pp	54.75%
Common phrases (≥ 50 users)	49.33%	36.59%	12.74pp	50.94%
Common phrases (≥ 80 users)	46.37%	30.64%	15.73pp	46.14%

Table 3. Simulation III CER results (when the KSR is fixed for each subset – one letter is typed for each word). The more common the intended input, the larger the CER improvement (Δ) for PhraseWriter (PW) over the baseline decoder. The lower keystroke saving rate (KSR) as commonality increases suggests that common phrases generally consist of shorter words.

From Table 3 we can also see that the keystroke savings rate with a fixed, one-letter prefix is smaller for common phrases than other subsets, which indicates that common phrases often consist of words that are relatively short, and the common phrase group with a higher commonality contains even shorter words in general.

5.5.2 Case analysis. Here we show how the decoding accuracy varies with the commonality of the expression with two examples. All errors are underlined.

Target `I'll be at your place on Thursday to see her`
Typed `i b a y p o t t s h`
Baseline `I believe a year or I think the same here`
PhraseWriter `I be at your place on Thursday to see how`

In this example the target sentence contains many common expressions such as "I'll be" (55 users), "at your" (60 users), "your place" (17 users), "on Thursday" (66 users), and "to see" (88 users). The PhraseWriter decoder almost completed the entire sentence correctly, while the baseline completely missed the target sentence. This example shows that longer context allows the PhraseWriter decoder to complete common expressions correctly with very little input – corresponding to the major improvement demonstrated in the statistical results.

Target `the battle is named after cape trafalgar in southern spain`
Typed `t b i n a c t i s s`
Baseline `to be in new a couple times I should say`
PhraseWriter `today but I need a couple things I said she`

In this example the target sentence conveys a more specific meaning and contains fewer common phrases. Both the baseline and PhraseWriter decoders produce an incorrect prediction.

We also tested a variation of this example, using the first two-letters of each word:

Target `the battle is named after cape trafalgar in southern spain`
Typed `th ba is na af ca tr in so sp`
Baseline `The back is napping after can try in so so`
PhraseWriter `The baby is named after can trade in some spots`

Although the predictions are still far from the target sentence, the PhraseWriter decoder was able to complete the common expression "is named after". This suggests that users may need to adjust the length of their input based on the commonality of the expression.

Overall, this simulation study shows PhraseWriter as an instance of C-PAK decoder has an higher potential success rate with more common phrases.

5.6 Discussion

Using these computational experiments we can obtain a better understanding of the C-PAK style input method. The findings help us understand the upper bound performance that can be achieved by PhraseWriter (Simulations I and II), as well as a strategy implication for using PhraseWriter (Simulation III). To illustrate the implications of our findings from a more holistic perspective, we reflect on the five main task components that users undertake when using a predictive keyboard:

(1) The preprocessing and planning of the characters to be typed.
(2) The execution of typing actions to enter the characters.
(3) The evaluation of suggested completions.
(4) The selection or rejection of correct or incorrect completion suggestions, respectively.
(5) The cognitive task of estimating, learning or recalling the abbreviated strings.

Simulation I showed that PhraseWriter can achieve more keystroke savings than word-level input, suggesting that it could reduce the frequency of task component 2 and the cost of 1 – relaxing the requirement for spelling accuracy. PhraseWriter can also reduce the context switching between text entry and suggestion selection as suggested by the 52.6% STSR, which means that it could reduce the frequency of task component 4. However, each suggestion evaluated during task component 3 will be longer (phrase vs. word). Furthermore, task component 5 is a unique challenge to PhraseWriter (and to C-PAK style input in general).

Simulation II suggested that the PhraseWriter decoder can reduce more errors than word-level input. The different benefits achieved in the first two simulations also suggests that different target users (frequent word completion users and full sentence typing typists) may find C-PAK style input more helpful in different aspects of their text entry experience.

Finally, Simulation III showed that C-PAK style input may be especially effective for entering common phrases.

While the simulation studies show the improved performance upper bounds of PhraseWriter as a practical implementation of C-PAK style input from the traditional keyboard, both in keystroke savings and in error correction abilities, the magnitude of these improvements are limited by the decoding technology. How much further improvement can be made to these upper bounds with more advanced technologies, such as large neural network language models, is a topic for future research.

6 AN EMPIRICAL STUDY OF C-PAK

Among the five task components introduced above, component 5 (the cognitive task of estimating, learning or recalling the abbreviated strings) is probably the most challenging to measure and analyze. With C-PAK, the fewer characters a user chooses to enter, the greater motor cost saving they can achieve – while also taking a higher risk of a decoding dead-end that requires backtracking and recovery. As a result, a user may need to dedicate more cognitive resources to choosing a strategy on a savings and risk spectrum while typing, potentially resulting in a deeper learning curve and performance impairment. Understanding how users get started with C-PAK on this spectrum is the main objective of this empirical study.

A prototype PhraseWriter keyboard was implemented as an Android input method by modifying an existing keyboard (Google Gboard – the baseline), and was designed to seamlessly integrate with the original system at both a technical and experiential level (Figure 7). As discussed in Section 3, there are many options for the design and implementation of such a system. We do not aim to exhaustively test all design variations, but instead chose a reasonable combination – leaning towards maintaining the existing design and interaction model of the keyboard:

- As described in the last section, an *n*-gram language model consisting of 1.3 million *n*-grams (with *n* ranging from 2 to 5) was used in constructing PhraseWriter. It is the same language model as in Gboard, a production keyboard optimized over many years for decoding quality and compute performance. This also meant that the same model could be used for both PhraseWriter and traditional decoding, removing a potential confound when examining user performance.
- Open-set coverage allowed users to enter any text that existed within the language model lexicon (164,000 unigrams).
- Users could enter phrases using variable-length prefixes, with a space to delimit words. This offered users the greatest flexibility in switching between C-PAK style input and conventional entry as necessary.
- We retained the existing user interface for suggestions, and presented phrase suggestions in a single-line suggestion bar. A dynamic number of suggestions were shown, but this was limited by the available space. No expanded suggestion list or partial-commit capability was supported. Suggestions were selected by either tapping on them, or tapping on the *enter* key (only for the most confident option).

As a user entered text, the uncommitted literal string was displayed in the text view with an underline (e.g. "H a" and "H a n w" in Figure 7). The most confident suggestion was displayed in bold if it scored above a predefined confidence threshold. The user could choose to type and select single-word suggestions as with a traditional system, or keep typing partial prefix strings separated by spaces until a complete phrase appears in the suggestion bar. A key difference from the baseline is that auto-correction/completion could not be triggered by pressing on the space key.

6.1 Participants & Apparatus

Fourteen participants (seven female) participated in the experiment. All considered themselves to be experienced typists on the baseline mobile keyboard (Gboard), and all but one considered themselves to be native/bilingual English users. Participants received a gift card for their time.

The experiment was run on a Google Pixel 3 running Android P, measuring 5.7 cm wide and 14.5 cm tall.

6.2 Conditions

There were two conditions in the study:

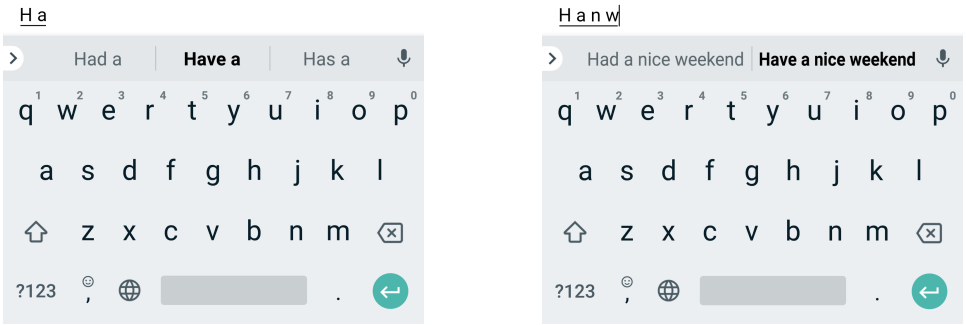


Fig. 7. Screenshots of the PhraseWriter user interface as a user enters "Have a nice weekend" from the partial input "H a n w".

(1) A *baseline* condition required participants to type on the baseline keyboard, which supported all typical forward suggestion and error correction features (i.e. word completion and next word prediction).
(2) An *experimental* condition required participants to type on our PhraseWriter keyboard, employing the modifications to the baseline keyboard described above. Participants were encouraged to use C-PAK style input, but were not forced to use the phrase abbreviations or to commit multiple words at a time. They could enter fewer letters (at least one) per word, pack many words into each commit, or type words completely as they felt comfortable.

Both conditions used the same spatial model for the user's input, and the same language model to generate predictions.

6.3 Phrase Set

We randomly sampled 75 phrases from the Enron email corpus. Ten were used as practice phrases for PhraseWriter, 60 were used for the main timed typing sessions to measure performance and subjective experience, and the remaining 5 were used as reflection phrases for a post-experiment interview. The phrase sets were identical across all users, but the order of the 60 experimental phrases was randomized for each user. We intentionally chose phrases with varying length and included relatively long sentences (a minimum of 3 words, a maximum of 12 words, with a median of 5 words).

6.4 Procedure

The study used a within-subject design and took approximately 45–60 minutes per participant.

Participants first learned how to use PhraseWriter by watching a video that demonstrated the process of entering an example sentence. They then typed the 10 practice phrases using PhraseWriter. During this process the experimenter encouraged them to explore different abbreviation and commit strategies, but did not make any specific recommendations. All participants used Gboard (the baseline keyboard) as their primary smartphone keyboard on a daily basis, so we considered all of them expert Gboard users and did not prepare a practice session for the baseline keyboard.

Participants then completed six timed blocks. The first and last blocks were completed on the baseline keyboard, with the four intermediate blocks on PhraseWriter. For all blocks, participants were instructed to type in the manner most comfortable to them. There were no requirements on how often they should use word/phrase completion. Gboard was only used for one block at the beginning and one at the end (rather than a fully counter-balanced design) to mitigate the confounding effect of the increased familiarity of the study task on their performance. Furthermore, we wanted to collect more data on PhraseWriter's performance and potential learning effects (although it later proved to be too short for much learning), and asking participants to also do four blocks of typing using Gboard would have make the study substantially longer and effortful without generating interesting data.

Each phrase was presented to participants on a slide shown on a laptop (on a different screen), and they were asked to memorize it. They were then asked to type it on the keyboard without referring to the slide – although this was not strictly enforced and they were allowed to check the slide if required. This attempted to stimulate the challenge of recalling a word's spelling that users encounter in practice.

After the timed typing blocks, participants were asked to type the five reflection phrases and explain their decisions on how to abbreviate a phrase and when to make commits while doing so.

6.5 Results

6.5.1 Motor Action Savings. Figure 8 shows the mean keystroke savings for each experimental block. Overall, the mean keystroke saving rate of PhraseWriter (25%, 95% CI [21%, 28%]) was higher than that of the baseline (16%, 95% CI [9.4%, 23%]): $t(13) = 3.35$, $p = .005$.

The average number of words entered per commit (i.e. per explicit suggestion tap) was 3.5 (95% CI [2.8, 4.2]) with PhraseWriter, and participants were able to save 23.0% more of their explicit suggestion taps than the baseline. Although reducing the number of suggestion taps does not increase the keystroke saving rate, it replaces potentially costly actions with word-delimiter taps (i.e. spaces) – and can therefore be beneficial.

6.5.2 Character Error Rate. The mean character error rate with PhraseWriter (0.41%, 95% CI [0.17%, 0.65%]) was lower than that of the baseline (0.75%, 95% CI [0.35%, 1.15%]) – although the difference was not significant: $t(13) = 1.89$, $p = .08$.

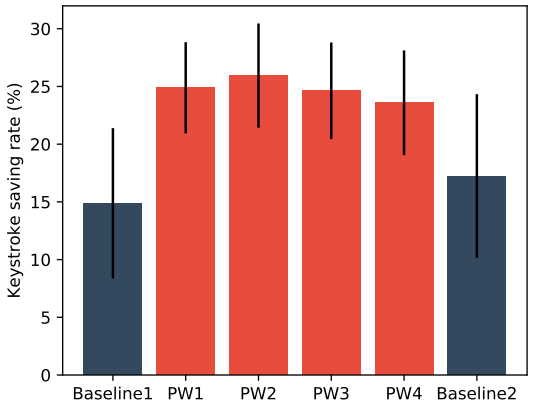
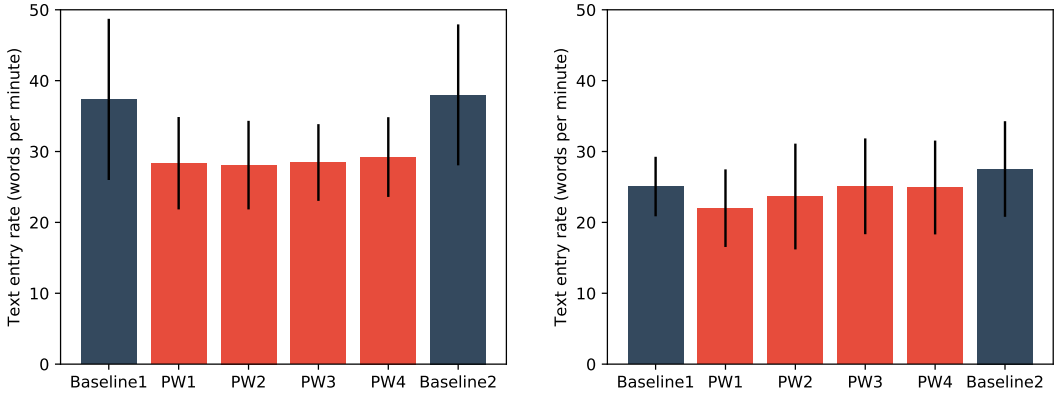


Fig. 8. The mean keystroke saving rate (±95% CI) in each experimental block.



(a) All participants. (b) High word completion use participants.

Fig. 9. The mean text entry rate (words per minute; ±95% CI) in each experimental block across (a) all users ($n = 14$), and (b) users with a high use of word completions ($n = 8$).

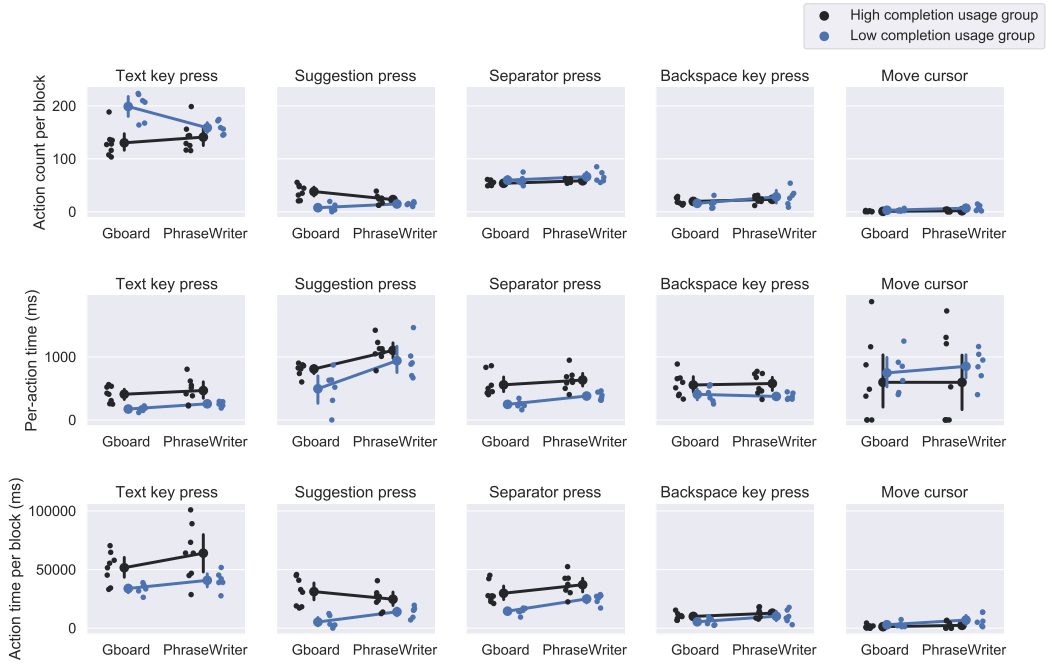


Fig. 10. The total action count per block (top row), average preparation time (middle row), and total action time per block (bottom row), for each type of action in each condition (±95% CI).

6.5.3 Text Entry Speed. Figure 9a shows the average text entry speed for all participants in each experimental block. Overall, the mean text entry speed measured by words per minute was lower with PhraseWriter (28.5, 95% CI [22.8, 34.2]) than with the baseline (37.7, 95% CI [27.3, 48.1]): $t(13) = -3.44$, $p = .004$.

6.5.4 Further Analysis on Time Performance. We found that participants could be divided into two groups with different word completion utilization behaviors based on the first baseline block. The first group ($n = 6$) typed almost all words completely and had a keystroke savings rate lower than 2.3%, while the second group ($n = 8$) used word completions actively and had a keystroke saving rate above 14.9%. We refer to these groups as the "low completion group" and "high completion group", respectively.

Figure 9b shows the average text entry speed for participants belonging to the high completion group. In this group, their text entry speed with PhraseWriter (23.9 words per minute, 95% CI [17.7, 30.2]) was not significantly different to that of the baseline (26.3 words per minute, 95% CI [21.6, 31.0]): $t(13) = -2.18$, $p = .07$.

Further analysis revealed that the high completion group required a longer time interval between keystrokes (mean 556 ms, 95% CI [432, 681]) than the low completion group (mean 305 ms, 95% CI [242, 369]; $t(12) = 3.86$, $p = .002$), which suggests that frequent users of word completion features type slower in general, potentially due to the higher motor costs [32]. This supports our Simulation II (Section 5.4) assumption that faster typists use word completions less frequently.

To understand what accounts for the performance difference between the high and low completion groups, we calculated the action time and action count for five types of actions: (1) text key presses, (2) suggestion presses, (3) seperator key presses, (4) backspace key presses, and (5) cursor

movements (see Figure 10). The bottom row in Figure 10 (action time per block) shows that for both groups the performance gap between the baseline and PhraseWriter is predominantly caused by three actions: text key presses, suggestion presses, and separator key presses. For the low completion usage group, the top row (action count per block) shows that these users saved more keystrokes and kept a consistent number of explicit suggestion selections when using PhraseWriter. For both groups, the average time spent on each action (middle row) was consistent for all types of action between the two conditions, except for suggestion presses, which suggests that the cognitive overhead of selecting a multi-word suggestion may be the main cause of the slower time performance with PhraseWriter.

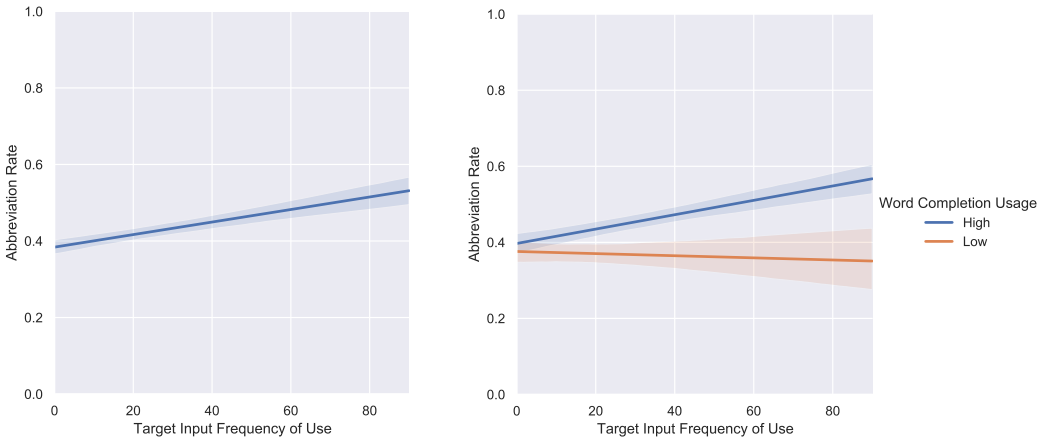
The figure also shows that C-PAK may have different implications on the text entry experience for different types of users. For the low completion group, C-PAK allowed them to significantly reduce the number of text key press actions. This suggests a benefit for fast typists to type out fewer characters explicitly. For the high completion group, the suggestion press count was lower, while the separator key press count was higher. This is consistent with the one of the hypothesized benefits of C-PAK, which is to replace the costly suggestion selection actions with lower-cost separator actions (e.g. entering spaces).

6.5.5 Relationship Between Abbreviation Rate and Phrase Commonality. In Simulation III (Section 5.5) we demonstrated that PhraseWriter is particularly powerful for completing common phrases, which means users can theoretically save more keystrokes when entering common phrases using PhraseWriter than using word-level completion. Accordingly, we curated a dataset that contained the literal input and the target input in pairs for each commit action to examine the relationship between the abbreviation rate of the literal input and the commonality of the target input (word or phrase). This analysis provides empirical evidence about novice users' ability to leverage the benefits of C-PAK for common phrases.

The abbreviation rate is defined as the number of characters in the literal input minus the number of words, normalized by the number of characters in the target. This is effectively a normalized version of the keystroke savings rate that ranges from 0 to 1 regardless of the word count and the length of the word. The commonality of a phrase is defined as the number of individuals that used the phrase in Enron dataset, which is the same measure used in Simulation III (Section 5.5).

To analyze this dataset, we built a linear regression model: the dependent variable is the abbreviation rate, and the independent variables include the commonality of the target input (ranging from 0 to 90), whether the user has high use of word completion in the first baseline block, and the session number (ranging from 1 to 4 since we only study PhraseWriter here). The results show that the commonality of the target input has a statistically significant positive relationship with the abbreviation rate ($\beta = .0016$, 95% CI [0.000, 0.003], $p = .008$), as shown in Figure 11a. The session number has a significant negative relationship with the abbreviation rate ($\beta = .025$, 95% CI [-0.049, -0.001], $p = .045$), and we only found a significant interaction between commonality and word completion usage ($\beta = -0.0022$, 95 %CI [-0.003, -0.001], $p < .001$) – illustrated in Figure 11b. We also built a linear regression model for the character-level error rate against the same factors, but did not observe any significant main or interaction effects.

Our analysis suggests that without special training, novice PhraseWriter users could quickly save more keystrokes when entering common phrases while maintaining a stable accuracy. Furthermore, we learned that frequent users of word completion achieved higher abbreviation rates for common phrases than infrequent users of word completion, suggesting that their knowledge about how to create effective abbreviations obtained from word-level completion may transfer to phrase-level completion. This also helps explain why the time performance between the baseline and the PhraseWriter was closer for people in the high completion group.



(a) All participants. (b) Participants with different word completion use

Fig. 11. The relationship between the abbreviation rate and the commonality of the target input (±95%CI).

6.5.6 Qualitative Feedback. In analyzing the qualitative feedback collected from participants after the experiment, most realized that PhraseWriter worked better for common phrases and felt more confident entering common phrases with abbreviated input. Even if the intermediate suggestion was incorrect, they reported finishing the entire phrase and relying on post-hoc correction. However, what they considered a common phrase differed between participants. For example, one participant considered the entire phrase "Both of us are still here" as a common expression, while another only considered "Both of us are" as common.

Long commits were generally avoided (P14: *"As a user, I will probably type 3–4 words and take the suggestion if the suggestion looks right."*). Several reasons were mentioned: fewer candidates can be shown as the length of the phrase increases; long phrases inherently have more uncertainty and are more likely to reach a "dead-end"; it took more time to examine long suggestions; and there was an increased error correction cost.

During the study, participants developed some knowledge about what partial input to use for certain phrases, such as using "A y" for "Are you". Some reported that there was a learning process and that they *"became better at predicting the system's performance after some practice"* (P5). Two participants said they tended to type short words completely, even if they knew it was not needed.

Three participants mentioned they felt being able to type less was itself a benefit. One participant mentioned that he felt more comfortable typing with PhraseWriter because suggestion selections were less often needed. However, four participants reported an increase in mental demand when using PhraseWriter.

Two salient themes emerged regarding possible improvement. The first was integrating grammar restrictions when decoding the target phrase. For example, one participant complained about the suggestion "Please revise according" (Figure 12a) because *"given the grammatical constraint, it should be pretty obvious (that the last word should be 'accordingly' rather than 'according')"* (P4). The second was going beyond word prefixes, and also supporting common abbreviations – such as using "r u" to represent "are you". These common abbreviations seemed to be more intuitive for users therefore should be considered integrating into C-PAK style input (P10: *"Knowing that I can use shorthand, I tend to use those common abbreviations although I don't do that a lot in my real life"*).

6.6 Discussion

This lab study allowed us to observe how novice users entered 40 phrases using a C-PAK keyboard. The results demonstrated not only novice users' ability to learn and use C-PAK style input, but also the limitations, mostly cognitive, of the current design.

Even within the first hour of using it, users of PhraseWriter could save more keystrokes (mean 25%) than the baseline (mean 16%). This difference is larger than that of the simulation results, but neither is at their theoretical maxima (49.4% and 46.3%, respectively, see Section 5.3). On average, users of PhraseWriter realized about half of its abbreviation or keystroke-savings potential – whereas the word-level input baseline only realized 35%. It is worth noting that the value of motor control savings depends on the user and context – for example, the value of each keystroke saving for motor impairment users could be very high.

Participants' time performance with PhraseWriter was always lower than the baseline. This finding is consistent with other previous user studies of predictive and abbreviated text input methods [1, 32, 37, 56]. This may be due to the increased cognitive cost as suggested by the increased preparation time for some actions (Figure 10), their existing expertise with traditional text entry methods, or the learning curve of PhraseWriter.

Participants were able to save 23% of the explicit suggestion taps with PhraseWriter than the baseline. Furthermore, the average number of words per commit we observed in user studies (3.5, all target sentences containing at least three words) was similar to the optimal words per commit as identified in the simulation studies (2.39 for target sentences containing at least two words, 2.7 for target sentences with at least four words, Section 5.3). Our post-study interviews corroborated the above findings by showing that participants had intuitions about the trade-off between entering longer text with more context vs. shorter text with more suggestions.

Both our qualitative and quantitative analysis demonstrated that participants also identified common phrases as a potential advantage for C-PAK, which is in line with the findings of our simulation studies (Section 5.5). Although due to the length of the controlled lab study, they did not have much of an opportunity to establish a mapping between the commonality of a phrase to the optimal abbreviation to enter it.

Another observation is that C-PAK may have different implications for fast typists and slow typists in terms of motor cost savings. Fast typists were able to save more text key press actions, while slow typists, who are already frequent users of word completion, did not seem to further increase their keystroke savings when using C-PAK as a new style of input. Conversely, PhraseWriter allowed frequent word completion users to replace costly suggestion selection actions with separator press actions, which is another type of motor saving that we hypothesized and quantified with the STSR. In addition, we also found that frequent users of word completion had a better intuition about the benefits of saving more keystrokes with common phrases. This sort of non-linear effect suggests that C-PAK may help different typists in different ways. Therefore, adjusting the presentation of word or phrase input suggestions based on the users' typing habits may help quickly adapt to the most appropriate method to reduce motor cost for the target user group.

Overall, the study found that the initial users of PhraseWriter could indeed make use and take advantage of the greater amount of C-PAK keystroke and selection tap savings, despite the uncertain envelope beyond which the reduced input strings would not produce the intended text. They adopted a conservative strategy to minimize the risk of crossing over the risk envelope. On average they were about only half way to the cross-over points. They also had an intuitive sense where they could save more keystrokes by abbreviating more on more common phrases. It appears that they understood the more common a phrase is, the higher the redundancy, the more keystrokes can be saved without the risk of crossing over the threshold into a dead-end.

7 GENERAL DISCUSSION AND FUTURE WORK

Text input is an age-old and never ending research and innovation topic. A new input paradigm takes place when the performance advantage arising from the new design space clearly overcomes the user learning cost [54].

There are many aspects and dimensions of text input performance, of which *efficiency* is key. There have been two broad approaches in today's mobile keyboards to improving efficiency: automatic completion of incomplete input, and relaxation of input precision – both based on machine intelligence embodied by models and algorithms, particularly language modelling. On the other hand, as shown in large scale web-based data collections and analyses [13, 29], today's mobile keyboards powered by auto-correction, word completion, and gesture typing are still lagging far behind desktop keyboards in speed and error rate.

In this project we set out to systematically research the opportunities and challenges offered by C-PAK-style input that allows variable length word prefix abbreviations in a phrase – an under-researched paradigm of input that previously had only been used for Chinese. We have shown that C-PAK can be implemented on today's mobile keyboard technology with potential keystroke savings greater than conventional abbreviations (such as omitting vowels or repeated letters), or the more commonly-used forward word completion. C-PAK allows a variable degree of abbreviation, ranging from only the initial character to all of the characters in a word. When a user types all of the characters in each word, it converges with phrase-level input, another research direction that had drawn research interest [e.g. 46, 56]. C-PAK shares some common characteristics with full phrase input – larger commit chunks and more correction power, but also the increased risk of larger errors that take time to recover. In the remainder of this section we further discuss what theoretical and empirical conclusions we can draw from the PhraseWriter project.

7.1 Theoretical and Empirical Benefits

The main benefit of C-PAK-style input is the greater keystroke and selection tap savings that users can achieve. The simulation studies showed that the PhraseWriter decoding algorithm could provide motor action savings and error corrections above that of forward suggestions. In terms of empirical feasibility, our lab study results suggest that novice users can quickly grasp the idea of C-PAK and take advantage of it – developing good intuitions and strategies to enter phrases with partial input. Given the complexity and uncertainty of C-PAK-style input, it is somewhat surprising that novice users could take advantage of the paradigm from the start, albeit conservatively. The fast convergence of participants to a decent keystroke saving rate, and the use of more abbreviated input for common phrases suggests that users' language knowledge accelerated the exploration process. This echoes the simulation results on common phrases, suggesting people have an intuitive sense of the information entropy in a phrase. The more common a phrase, the lower the entropy – and the more redundancy the user could feel in the phrase which lead them to more aggressively abbreviate it.

When using the baseline forward-suggestion keyboard there is a large gap between the theoretical keystroke savings (42.1%) and the actual savings (16%), potentially because users need to make an explicit selection on every word to fully exploit word completion. That is, users must attend to and act on the presented suggestions after every word. In contrast, PhraseWriter reduces this gap (i.e. increasing the keystroke savings rate by 56%, and with 23% fewer explicit suggestion taps than the baseline) by enabling a commit to contain multiple words at the same time – reducing the total number of times users need to spend attending to the suggestions.

Furthermore, the relative trade-off between the motor movement cost in making keystrokes versus the cognitive cost in attending, evaluating, and confirming suggestions depends on the

(a) Ungrammatical suggestion. The intended input is "Please revise accordingly"

(b) Suggestion differs from the intended input "Where do you want to meet".

Fig. 12. Some failure examples of C-PAK.

user's ability. For users with motor control impairments, prediction and suggestion interfaces are used more often [23, 25], and therefore C-PAK-style input could be more beneficial in accessibility applications.

In our lab study, users with two different typing habits naturally emerged when using the conventional keyboard: one group of users rarely used any word completion and generally typed faster, and the other group tended to use word completion frequently and generally typed slower. This serves as a good example of how individual differences may affect how users can benefit from C-PAK (or any predictive system). For the former group, a better strategy may be typing all words completely in most occasions and taking advantage of the better error correction capability of a C-PAK system. For the latter group, a better strategy may be use more phrase-level completion to save more keystrokes and reduce the interruptions due to suggestion evaluation and selection [45].

7.2 Limitations and Challenges

Although C-PAK style input has many promising benefits, the current PhraseWriter system design derived from conventional keyboard design did not allow users to fully realize those benefits into a time performance gain. It appears that an *n*-gram model and the conventional interface design might be appropriate for forward suggestions, but not sufficient to handle the flexibility and uncertainty surfaced by C-PAK style input.

Editing Costs. As a single C-PAK commit can contain multiple words, users may need to fix errors in earlier parts of their input. For example, the cost of error correction caused 25.2% of the extra time cost in PhraseWriter (the total time of the backspace key press and move cursor actions in Figure 10), which suggests a need for better support around editing the partial input when it does not generate the correct suggestion, and in editing the text committed when it is slightly different from the intended input (Figure 12b).

In current keyboards, correcting an early error requires users to either position the cursor after the character they want to edit, or to delete everything proceeding the error – both of which harm performance and overshadow the benefits of keystroke savings from suggestions. To enhance editing support, an interface may introduce an editing mode that allows users to indicate where they want to edit at word level. For example, the SHARK^2 system [26] and PhraseFlow [56] used a one line composing window through which text output flows to the application, allowing opportunities to correct or edit more than the last word (a similar interface is a common feature of keyboards for East Asian languages). As these edits occur prior to a commit, they could be used to improve the suggestions in other parts of the phrase.

PhraseWriter is biased toward the current production keyboard interface design, while accommodating the new C-PAK features. There have been alternative design proposals in the literature on error correction [2, 3, 31, 55]. While some of them are not practical yet, we think the "smart restorable backspace technique" [3] could be quite practical for a production keyboard, and could be especially helpful to C-PAK-style input.

Cognitive Costs. Our lab study results showed that the average preparation time for actions was generally higher with PhraseWriter, especially for the suggestion press action, which suggests higher cognitive costs than the baseline. This might be because users have more options for entering each word, including: typing the word completely, typing a partial word and selecting a word completion suggestion, or typing a partial word and deferring the commit. The risk of not encountering a correct suggestion later in the input further increases the burden on users to find a good input strategy. This problem could be addressed by allowing users to perform partial commits to reduce the risk of missing correct suggestions.

Another source of cognitive load may be the longer suggestions that users need to review. An interface may therefore control the number of suggestions displayed based on the confidence and utility of the suggestions [9, 32] so users will evaluate fewer suggestions, or highlight different parts among suggestions to streamline the comparison of suggestions.

Learning Curve. Novice users will need to spend time familiarizing themselves with phrase-level abbreviations. Their success will depend on how close they type to the optimal level of abbreviation. Our experiments suggest that PhraseWriter works better on common expressions, which is a natural strategy. Furthermore, the high coverage of a small set of common phrases shows the feasibility of learning phrase-level abbreviations if a user only focuses on common phrases. Future work may investigate how to help users estimate the commonality or predictability of a phrase, and how to encourage them to use them opportunistically. Our lab study has shown promising signals that users could leverage their language knowledge to expedite the exploration process. In addition, the Chinese *Jianpin* method proves a special case that C-PAK is practically viable despite these questions, but further research is still needed.

7.3 Future Work

This paper has presented the first systematic study of C-PAK input. While it is also the most systematic study of abbreviation input (see Section 2), it is still limited relative to the complexity and scope of C-PAK. We divided our study of C-PAK style input into two parts: (1) a simulation study of the PhraseWriter decoding algorithm, and (2) an open-ended lab study of PhraseWriter's usage patterns and preferences. This allowed us to study both the maximal potential of the method (based on *n*-gram technology), as well as actual usage of variable length prefix abbreviations in the initial stage of learning C-PAK. While the current research shows the technical feasibility and the initial usability of C-PAK based on today's production keyboard technologies, pressing future work is needed in two interwoven directions to enable and understand the full potential of C-PAK input.

7.3.1 Further empirical studies. We laid out a vast design space for C-PAK by listing various design choices in Section 3. However, the design choices embodied in PhraseWriter only represents one point in the the design space of C-PAK. Therefore our findings may not generalize to the rest of the space. The strengths and weaknesses of other design choices remain to be investigated by future research. Even with the current PhraseWriter design and implementation of C-PAK, there are still open research questions. For example only about half of the maximum keystroke savings (49.4%) shown in the PhraseWriter simulation study was measured in our lab study of initial user behavior and performance. Longitudinal studies of user's interaction behavior and strategy, ideally in the context of daily real use, are needed to understand the user cost and benefit of C-PAK by further investigating how and when users use C-PAK over time, their eventual saving rates, common strategies of abbreviation and commit actions, and the relationship between a user's language skills and their use of C-PAK [14, 19].

Our participants in the PhraseWriter study were mostly young (in their 20s or 30s) and had native/bilingual-level English proficiency. Future research is needed to investigate the use of C-PAK of older people and people with varying levels of language skills. Applications of C-PAK in accessibility interface design where keystroke savings might be even more important is another area that needs dedicated and contextualized research.

7.3.2 Future technical and system research. While the three simulations showed that PhraseWriter consistently outperformed its forward-prediction-based baseline in reducing keystrokes, errors, and select-commit actions, particularly for common phrases – the improvements were modest. Error analysis shows that the *n*-gram language model used does not always generate candidates that are grammatically correct or semantically meaningful. In the case analysis section of Simulation II, we gave an example where PhraseWriter did not fully leverage the context to generate semantically proper suggestions ("we will need the gas by next match or april"); and in the case analysis section of Simulation III, we gave an example where it did not generate a grammatical suggestion ("I be at your place on thursday"). Although the PhraseWriter decoder got most of the text correct, these errors would be unexpected to users.

We show another ungrammatical suggestion example in Figure 12a with a trickier problem: the suggestion "Please revise according" could be valid if there are more words coming, but is annoying to users when they know the sentence is complete and the last word should be an adverb ("accordingly").

The *n*-gram language model used in PhraseWriter is a reasonable starting point as it is the status quo technology in most mobile keyboards (including the baseline keyboard we used), but it may have also limited the fluency of its predictions as well as the maximum keystroke savings rate. Due to the small model size and limited coverage, mobile grade *n*-gram models tended to take frequent back-off to uni-grams which have no contextual connection therefore causing disfluencies and grammatical errors in its predictions and completions.

Developing and evaluating C-PAK decoders based on more advanced language modeling techniques is therefore a pressing future line of research on C-PAK input. Having brought rapid progress in domains such as machine translation and dialogue systems, deep learning neural network-based language models [20, 39, 44, 53] offer an exciting potential to improve the efficiency and fluency of C-PAK output. Such improvements may fundamentally change the cognitive, learning and UI design challenges revealed in the current PhraseWriter project.

8 CONCLUSION

We have studied *C-PAK*, a text input method that can exploit bidirectional context to expand partial input into complete phrases, and is tolerant to spatial errors. The performance of C-PAK in actual use can be affected by multiple factors, including the effectiveness of the decoding algorithm, the user interface design, and users' familiarity with the algorithm. Using PhraseWriter, a specific implementation of C-PAK atop a production mobile keyboard, we conducted two studies to understand the potential and challenges of C-PAK from two perspectives. The first was a set of computational simulations, which explored the theoretical limits and statistical strength of PhraseWriter's improvements over conventional forward suggestions. Although PhraseWriter generally improved the overall accuracy and motor action savings presented by suggestions, the most salient improvements were found for common phrases. We then conducted a lab study to understand how novice users approached PhraseWriter. We found they could quickly grasp the general idea and achieve decent text entry speed in their initial use. With PhraseWriter, users were able to save significantly more keystrokes, commit multiple words at a time to save on the costly suggestion taps, and identify common phrases as a potential advantage for C-PAK. We

showed that different users benefited from PhraseWriter in different aspects (slow typers and frequent completion users vs. fast typers and infrequent word-completion users), which supported our design choices emphasizing the flexibility of abbreviation choices. We also identified that the cognitive cost of forming abbreviations on the fly to achieve keystroke savings and avoid decoding errors may account for the reduction in time performance. Accordingly, we outline future research directions to improve the time performance when maintaining keystroke saving benefits from both design and technical perspectives.

In more general terms, this paper makes the following contributions to smart keyboard research, by means of design, system prototyping, computational simulation, and empirical measurements:

(1) Identified, defined and explored the interface, interaction, and technology design space of C-PAK as a paradigm to text input.
(2) Designed and implemented a C-PAK decoder, a specific algorithm and interface design instance readily attainable on the current state of art mobile computing technologies, as a research tool and a specific path in the C-PAK design space.
(3) Computationally measured the fundamental bounds of error correction and motor cost savings of PhraseWriter as a current version of C-PAK decoder, under various conditions and against realistic text input test-sets, as compared to its corresponding word-level completion and correction.
(4) Empirically gained insights into users' ability to take advantage of and benefit from the implementation of PhraseWriter as one path in the C-PAK design space.

9 ACKNOWLEDGEMENT

The work was primarily conducted during the first author's internship at Google. We thank our many Google colleagues, particularly Tom Ouyang, for their advice and contributions to this project.

REFERENCES

[1] Jiban Adhikary, Jamie Berger, and Keith Vertanen. 2021. Accelerating Text Communication via Abbreviated Sentence Input. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Association for Computational Linguistics. https://doi.org/10.18653/v1/2021.acl-long.514

[2] 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. *Graphics Interface 2019* (2019).

[3] 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*. ACM. https://doi.org/10.1145/2858036.2858407

[4] 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 - MobileHCI '12*. ACM Press. https://doi.org/10.1145/2371574.2371612

[5] Françoise Beaufays and Michael Riley. 2017. The Machine Intelligence Behind Gboard. https://ai.googleblog.com/2017/05/the-machine-intelligence-behind-gboard.html (visited on 2017-05-24).

[6] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. 2003. A Neural Probabilistic Language Model. *Journal of Machine Learning Research* 3, Feb (2003), 1137–1155.

[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*. ACM. https://doi.org/10.1145/2470654.2470732

[8] Ciprian Chelba, Mohammad Norouzi, and Samy Bengio. 2017. N-gram Language Modeling using Recurrent Neural Network Estimation. (2017). arXiv:cs.CL/1703.10724v2

[9] 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*. ACM. https://doi.org/10.1145/3292500.3330723

[10] Ann Copestake. 1997. Augmented and Alternative NLP Techniques for Augmentative and Alternative Communication. In *Natural Language Processing for Communication Aids*. 37–42. https://www.aclweb.org/anthology/W97-0506

[11] J.J. Darragh, I.H. Witten, and M.L. James. 1990. The Reactive Keyboard: a predictive typing aid. *Computer* 23, 11 (1990), 41–49. https://doi.org/10.1109/2.60879

[12] Patrick W. Demasco and Kathleen F. McCoy. 1992. Generating text from compressed input: an intelligent interface for people with severe motor impairments. *Commun. ACM* 35, 5 (1992), 68–78. https://doi.org/10.1145/129875.129881

[13] 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*. ACM. https://doi.org/10.1145/3173574.3174220

[14] M.A. Drouin. 2011. College students' text messaging, use of textese and literacy skills: College students' text messaging. *Journal of Computer Assisted Learning* 27, 1 (2011), 67–75. https://doi.org/10.1111/j.1365-2729.2010.00399.x

[15] 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*. ACM. https://doi.org/10.1145/2207676.2208520

[16] 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*. ACM. https://doi.org/10.1145/2702123.2702503

[17] Nestor Garay-Vitoria and Julio Abascal. 2005. Text prediction systems: a survey. *Universal Access in the Information Society* 4, 3 (2005), 188–203. https://doi.org/10.1007/s10209-005-0005-9

[18] 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 - IUI '02*. ACM Press. https://doi.org/10.1145/502716.502753

[19] Sarah De Jonge and Nenagh Kemp. 2010. Text-message abbreviations and language skills in high school and university students: Texting in High School and University Students. *Journal of Research in Reading* 35, 1 (2010), 49–68. https://doi.org/10.1111/j.1467-9817.2010.01466.x

[20] Uday Kamath, Kenneth L. Graham, and Wael Emara. 2022. Bidirectional Encoder Representations from Transformers (BERT). In *Transformers for Machine Learning*. Chapman and Hall/CRC, 43–70. https://doi.org/10.1201/9781003170082-3

[21] S. 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. https://doi.org/10.1109/tassp.1987.1165125

[22] Bryan Klimt and Yiming Yang. 2004. The Enron Corpus: A New Dataset for Email Classification Research. In *Machine Learning: ECML 2004*. Springer Berlin Heidelberg, 217–226. https://doi.org/10.1007/978-3-540-30115-8_22

[23] Heidi Horstmann Koester and Simon Levine. 1996. Effect of a word prediction feature on user performance. *Augmentative and Alternative Communication* 12, 3 (1996), 155–168. https://doi.org/10.1080/07434619612331277608

[24] H. H. Koester and S. P. Levine. 1993. A model of performance cost versus benefit for augmentative communication systems. In *Proceedings of the 15th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*. IEEE, 1303–1304. https://doi.org/10.1109/iembs.1993.979147

[25] H. H. Koester and S. P. Levine. 1994. Modeling the speed of text entry with a word prediction interface. *IEEE Transactions on Rehabilitation Engineering* 2, 3 (1994), 177–187. https://doi.org/10.1109/86.331567

[26] Per-Ola Kristensson and Shumin Zhai. 2004. SHARK^2: a large vocabulary shorthand writing system for pen-based computers. In *Proceedings of the 17th annual ACM symposium on User interface software and technology - UIST '04*. ACM Press. https://doi.org/10.1145/1029632.1029640

[27] Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model. In *Proceedings of the 11th Annual Conference of the International Speech Communication Association*.

[28] Tom Ouyang, David Rybach, Françoise Beaufays, and Michael Riley. 2017. Mobile Keyboard Input Decoding with Finite-State Transducers. (2017). arXiv:cs.CL/1704.03987v1

[29] Kseniia Palin, Anna Maria 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 the 21st International Conference on Human-Computer Interaction with Mobile Devices and Services*. ACM. https://doi.org/10.1145/3338286.3340120

[30] Stefano Pini, Sangmok Han, and David R. Wallace. 2010. Text entry for mobile devices using ad-hoc abbreviation. In *Proceedings of the International Conference on Advanced Visual Interfaces - AVI '10*. ACM Press. https://doi.org/10.1145/1842993.1843026

[31] Felix Putze, Tilman Ihrig, Tanja Schultz, and Wolfgang Stuerzlinger. 2020. Platform for Studying Self-Repairing Auto-Corrections in Mobile Text Entry based on Brain Activity, Gaze, and Context. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. ACM. https://doi.org/10.1145/3313831.3376815

[32] 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*. ACM. https://doi.org/10.1145/2858036.2858305

[33] Frode Eika Sandnes. 2015. Reflective Text Entry: A Simple Low Effort Predictive Input Method Based on Flexible Abbreviations. *Procedia Computer Science* 67 (2015), 105–112. https://doi.org/10.1016/j.procs.2015.09.254

[34] Frode Eika Sandnes. 2018. Can Automatic Abbreviation Expansion Improve the Text Entry Rates of Norwegian Text with Compound Words?. In *Proceedings of the 8th International Conference on Software Development and Technologies for Enhancing Accessibility and Fighting Info-exclusion*. ACM. https://doi.org/10.1145/3218585.3218586

[35] C. E. Shannon. 1951. Prediction and Entropy of Printed English. *Bell System Technical Journal* 30, 1 (1951), 50–64. https://doi.org/10.1002/j.1538-7305.1951.tb01366.x

[36] Stuart M. Shieber and Ellie Baker. 2003. Abbreviated text input. In *Proceedings of the 8th international conference on Intelligent user interfaces - IUI '03*. ACM Press. https://doi.org/10.1145/604045.604103

[37] Stuart M. Shieber and Rani Nelken. 2006. Abbreviated text input using language modeling. *Natural Language Engineering* 13, 2 (2006), 165–183. https://doi.org/10.1017/s1351324906004311

[38] Andreas Stolcke. 2000. Entropy-based pruning of backoff language models. *arXiv preprint cs/0006025* (2000).

[39] Martin Sundermeyer, Ralf Schlüter, and Hermann Ney. 2012. LSTM neural networks for language modeling. In *Interspeech 2012*. ISCA. https://doi.org/10.21437/interspeech.2012-65

[40] Kumiko Tanaka-Ishii. 2006. Word-based predictive text entry using adaptive language models. *Natural Language Engineering* 13, 1 (2006), 51–74. https://doi.org/10.1017/s1351324905004080

[41] Apple Input & Intelligence — Natural Language Processing Team. 2019. Language Identification from Very Short Strings. https://machinelearning.apple.com/2019/07/24/language-identification-from-very-short-strings.html (visited on 2019-07-24).

[42] Keith Trnka, John McCaw, Debra Yarrington, Kathleen F. McCoy, and Christopher Pennington. 2009. User Interaction with Word Prediction: The Effects of Prediction Quality. *ACM Transactions on Accessible Computing* 1, 3 (2009), 1–34. https://doi.org/10.1145/1497302.1497307

[43] Keith Trnka and Kathleen F. McCoy. 2008. Evaluating word prediction: framing keystroke savings. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics on Human Language Technologies Short Papers - HLT '08*. Association for Computational Linguistics. https://doi.org/10.3115/1557690.1557766

[44] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*. 5998–6008.

[45] 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*. ACM. https://doi.org/10.1145/3173574.3174200

[46] 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*. ACM. https://doi.org/10.1145/3290605.3300821

[47] Keith Vertanen and Per Ola Kristensson. 2011. A versatile dataset for text entry evaluations based on genuine mobile emails. In *Proceedings of the 13th International Conference on Human Computer Interaction with Mobile Devices and Services - MobileHCI '11*. ACM Press. https://doi.org/10.1145/2037373.2037418

[48] 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*. ACM. https://doi.org/10.1145/2702123.2702135

[49] Xiaolong Wang. 1988. Lùn hànyǔ pīnyīn, sān pīn, shuāng pīn, jiǎn pàn de tǒngyī biāodá xíngshì [A unified Chinese pinyin input method]. *Zhōngwén Xìnxī Xuébào* 2, 1 (1988), 26–31. http://jcip.cipsc.org.cn/CN/Y1988/V2/I1/26

[50] Tim Willis, Helen Pain, Shari Trewin, and Stephen Clark. 2002. Informing Flexible Abbreviation Expansion for Users with Motor Disabilities. In *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 251–258. https://doi.org/10.1007/3-540-45491-8_52

[51] Tim R. Willis, Helen Pain, and Shari Trewin. 2005. A Probabilistic Flexible Abbreviation Expansion System for Users With Motor Disabilities. In *Electronic Workshops in Computing*. BCS Learning & Development. https://doi.org/10.14236/ewic/ad2005.4

[52] 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*. ACM. https://doi.org/10.1145/2470654.2481384

[53] Heiga Zen, Yannis Agiomyrgiannakis, Niels Egberts, Fergus Henderson, and Przemysław Szczepaniak. 2016. Fast, Compact, and High Quality LSTM-RNN Based Statistical Parametric Speech Synthesizers for Mobile Devices. In *Interspeech 2016*. ISCA. https://doi.org/10.21437/interspeech.2016-522

[54] Shumin Zhai and Per Ola Kristensson. 2012. The word-gesture keyboard: Reimagining keyboard interaction. *Commun. ACM* 55, 9 (2012), 91–101. https://doi.org/10.1145/2330667.2330689

[55] 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. ACM. https://doi.org/10.1145/3332165.3347924

[56] Mingrui Ray Zhang and Shumin Zhai. 2021. PhraseFlow: Designs and Empirical Studies of Phrase-Level Input. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. ACM. https://doi.org/10.1145/3411764.3445166

[57] 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*. ACM. https://doi.org/10.1145/3173574.3174013