Gesture Input for Text Phrases

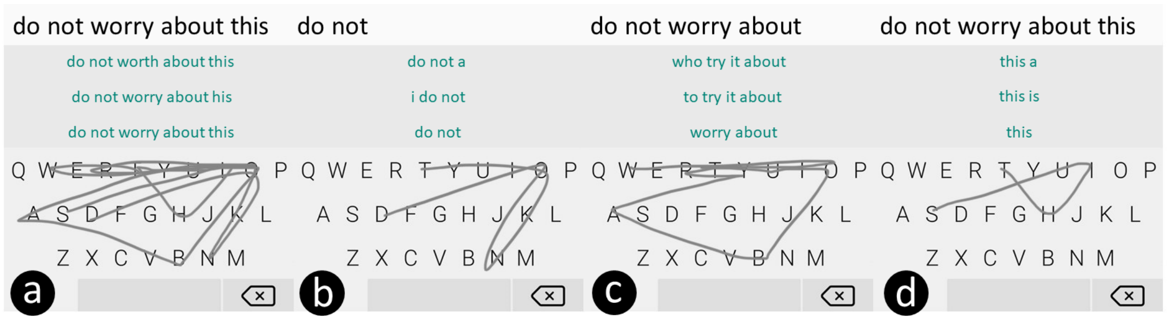

Figure 1: Phrase-gesture typing allows text to be entered phrase by phrase. (a) The phrase "do not worry about this" can be entered by swiping through all the letters of the words in the phrase without the need to use the space key between words. (b) to (d) Alternatively, the phrase can be entered via a mixture of word- and phrase-level gestures, e.g., a phrase gesture of "do not" followed by a phrase gesture of "worry about", and a word gesture of "this". The user can select the intended text from the candidate list shown above the phrase-gesture keyboard. The items in the list are shown in descending order with the top-ranked candidate placed at the bottom to make it closer for the finger to select.

ABSTRACT

We study phrase-gesture typing, a gesture typing method that allows users to type short phrases by swiping through all the letters of the words in a phrase using a single, continuous gesture. Unlike word-gesture typing, where text needs to be entered word by word, phrase-gesture typing enters text phrase by phrase. To demonstrate the usability of phrase-gesture typing, we implemented a prototype called PhraseSwipe. Our system is composed of a frontend interface designed specifically for typing through phrases and a backend phrase-level gesture decoder developed based on a transformer-based neural language model. Our decoder was trained using five million phrases of varying lengths of up to five words, chosen randomly from the Yelp Review Dataset. Through a user study with 12 participants, we demonstrate that participants could type using PhraseSwipe at an average speed of 34.5 WPM with a Word Error Rate of 1.1%.

CCS CONCEPTS

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

KEYWORDS

text entry, gesture input, language model, machine learning

1 INTRODUCTION

Gesture typing is a common text input method on touchscreen devices. Since its introduction in the early 2000s [36], gesture typing has gained wide adoption in commercial products including Gboard, SwiftKey, ShapeWriter, Swype, SlidIt, and TouchPal. Unlike typing by tapping keyboard keys (tap typing), gesture typing allows intended words to be expressed via finger stroke shapes that can be drawn less precisely on a touchscreen, where the tactile feedback of the keys is lacking. With practice, skill transition may occur from the novice model of visually guided tracing to the expert mode of recall-based gesturing that is quicker and can be performed with-out visual attention. The development of gesture typing over the years has led to variations suitable for new device form factors, such as laptops [13] or TV remote trackpads [41], and new use scenarios where both hands are available [7] or visual attention is not guaranteed [41].

In this paper, we study a new variation of gesture typing, called phrase-gesture typing. Unlike the existing approach, where an input phrase is typed through individual gestures, each representing an intended word, the phrase-gesture typing allows users to type not only words but also phrases through a single, continuous gesture composed of concatenated segments for each word of a short phrase, similar to handwritten signatures with a single stroke. Phrase-gesture typing has several benefits. First, users now have the choice not to lift the finger off the touchscreen and reengage for the next word and can swipe on the screen as long as they are willing to. This means after each word, users can immediately move the finger to the first letter of the next word once the current word is completed, or they can break as usual by lifting their finger. Phrase-gesture typing is a natural extension of the basic word-gesture typing and can be useful, especially for typing frequent phrases (e.g., "on my way", "thank you") and recall-based gesturing. Further, with phrase-gesture typing, text input speed can increase without the need to disengage and engage the touchscreen for every other word. Lastly, from the decoding perspective, with more contextual information in the input data, the keyboard decoder can better determine the users' intended words.

Despite these benefits, enabling phrase-gesture typing is challenging from both technical and interface design perspectives. Considering the technical challenge, it is hard to decode a long input stroke into a meaningful phrase without the user explicitly specifying delimiters (spaces to separate the words). None of the existing keyboard decoders supports phrase-level decoding for gesture typing. Further, the noises in the input data could accumulate with the concatenation of word gestures, making it even harder to correctly decode the users' intended words. From the interface design perspective, the usability of phrase-gesture typing relies on several design parameters that need to be carefully considered, including how long a phrase is allowed to type, how the feedback of the entered text is given, and how editing is supported.

To address these challenges, we developed PhraseSwipe, a gesture keyboard prototype that allows users to type via phrase-level gestures on smartphones (Figure 1). With PhraseSwipe, a user can swipe through all the letters of the words in an intended phrase of different lengths up to five words with a single continuous gesture. Alternatively, the user can type using a mixture of word- and phrase-level gestures. Switching among gesture typing of words and phrases with different lengths can occur at any point and the system can decode them accordingly without any knowledge about the number of words in the input. For example, a user can type "happy to see you" using a phrase gesture "happy to see", followed by a word gesture "you", or just a single phrase gesture "happy to see you".

The core of PhraseSwipe is a phrase-gesture decoder, developed using a transformer-based, end-to-end neural decoder that is capable of translating an input gesture directly into an ordered sequence of words. Unlike the existing neural language models in non-keyboard applications where both input and output are text, our decoder takes touch gestures as input and can work on any smartphone keyboard of different sizes and key ratios. While the typical approach of developing a massive neural language model involves a significant amount of time and effort, we demonstrate that an existing large language representation model, like BERT, can be fine-tuned with simple adaptions to enable phrase-gesture decoding without the huge model to be built and trained completely from scratch.

The effectiveness of our decoder still relies on a training dataset of phrase gestures, which, unfortunately, does not exist. To overcome this challenge, we trained our model with phrase gestures simulated based on minimum-jerk theory [27]. Our model was trained with over five million phrases randomly chosen from the Yelp Review Dataset [40] with the length of each training phrase ranging from one to five words. The decoder runs on a local server and generates real-time output, shown to the user on the phone through a list of top candidate phrases ranked based on confidence.

Through a system evaluation, we demonstrate the accuracy of our decoder and show that it's more efficient when phrases are entered using a single gesture than using several shorter ones. Further, to evaluate the usability of PhraseSwipe, we conducted a user study with 12 participants. Our results revealed that participants could achieve an average speed of 34.5 WPM with 1.1% uncorrected errors, which is 2.5 WPM faster than a baseline word-gesture typing method.

The main contributions of this work include: (1) a phrase-gesture typing keyboard developed for smartphones and a decoder trained for phrase-gesture typing; and (2) a user study demonstrating the effectiveness and usability of our implementation.

2 RELATED WORK

Text input as a part of HCI research has been widely studied in the past several decades. Work has been done on a variety of different topics, including interaction techniques and keyboard layout optimization for new use scenarios and device form factors. Our review of the existing literature primarily focuses on the state-of-the-art in gesture typing and decoding methods.

2.1 Gesture typing

Gesture typing was first introduced by Zhai and Kristensson in the early 2000s for mobile touchscreen devices [19, 36-38]. Unlike tap typing, where the letters in a word are entered by a user selecting the corresponding key on the keyboard, gesture typing allows uses to enter the word directly (rather than individual letters) by gesturing through the desired keyboard keys. Aside from its wide adoption on smartphone products, gesture typing has been extended by the research community to new use scenarios and device platforms. For example, Bi, et al. [7] converted the original gesture keyboard into a split keyboard suitable for larger devices like tablets to allow two thumbs to work together to enter a word. Zhu, et. al. [41] presented a method to enable gesture typing on a trackpad of a TV remote. Unlike gesturing on a touchscreen device, the keyboard is not visible on a TV remote but their method allows gesture typing to be carried out without the need for the users to pay visual attention to the keyboard or finger movement.

While much of the existing work focuses on touch surface devices, gesture typing is not limited to touch input. For example, Markussen, et al. demonstrated that gesture typing can be performed in the mid-air with a system capable of tracking the hand movement projections on a vertical display [22]. Chen et al. [12]'s work demonstrates that gesture typing can also be carried out using a pair of handheld controllers, making it a good candidate for text input in VR environments. Within the same space, Yu, et al. [35] studied how to develop interactions and decoding methods to enable gesture typing on VR headsets using the movement of the user's head. Aside from the head, the tile motion of a user's wrist [34] or finger [17] can also provide control for performing gesture typing, allowing text input to be carried out on small wearable devices using only one hand.

Among all the existing methods, what appears to be similar to our approach is a feature provided by SwiftKey [4], with which, the users can enter a sequence of words through a single gesture. The difference, however, is significant in two ways: (1) From a user's perspective, SwiftKey requires the user to specify a delimiter between two adjacent words by gesturing through the space key. Crossing the space key for every other word inherently increases swiping distance, which is an extra effort that may lead to fatigue over time and may unnecessarily impact typing speed. Our work differs in the way that no delimiter is needed when a phrase gesture is drawn, meaning that the burden is on the system (not the user) to handle the ambiguity in the input data. (2) Beyond the input method, the decoding principle of SwiftKey is also different. It uses a word-level decoder, which could be less accurate than a phrase-level decoder because the context of the entire input phrase is not used for decoding [30, 32]. In contrast, with our method, the decoding result of the earlier words is constantly updated for better accuracy as the gesture continues. To the best of our knowledge, our research is the first in the literature to study phrase-gesture typing and decoding.

2.2 Keyboard Decoder

One of the most significant challenges in developing text input methods for touchscreen devices is that user input data is noisy. To address this problem, keyboard decoders were developed. A typical keyboard decoder is composed of a spatial model, which provides the probability distribution over all keys on a keyboard, and a language model, which provides the probability distributions of a sequence of words for a certain language. Goodman and colleagues [15] were the first who studied how tap typing errors can be reduced by combining the spatial model and language model through the Bayes' rule. This approach has been widely adopted in modern commercial keyboards. Gboard [3], as an example, contains a spatial model and a low order n-grams language model, that was designed to be compact to handle real-time processing on mobile devices [24].

Keyboard decoders are far from perfect. As such, work has also been done to address some of the most significant issues. For example, Gunawardana et al. [16] demonstrated that an aggressive spatial model could sometimes prevent users from typing their desired text. Their approach using an anchored key-target method could effectively address this problem. Concerning the language model, accuracy is also an issue but recent studies have shown that the accuracy of language models can be largely improved using machine learning methods based on neural networks [10, 11, 14]. With some of the new deep language models, the contextual information in the input data can now be better used to determine the intended text. Machine learning methods have also been used to develop better decoders for word-gesture typing. For example, Alsharif and colleagues [5] showed that combining recurrent networks such as Long Short Term Memories [18] with conventional Finite State Transducer decoding [23] could lead to an improvement of accuracy up to 22% over the existing shape-matching-based approach.

While most popular keyboard decoders rely on delimiters, researchers have investigated ways to allow users to type contiguous words without using spaces. Thus, the systems need to handle phrase-level decoding [30-32, 39]. An example of the existing research in the phrase-level decoder is the work from Vertanen, et al. [32]. Their tap-typing keyboard is composed of a spatial model, a 12-gram character language model, and a 4-gram word language model. Through a user study, the authors showed that omitting the space key between words led to a faster entry rate. They also found that even novice users could adapt to writing sentences quickly without intermediate feedback for each word. Aside from the technical aspect, Zhang and Zhai [39] studied user interface design options for tap-typing keyboards with a phrase-level decoder and demonstrated that poorly designed interfaces could hinder user performance. The authors showed that the feedback design that could avoid cognitive overhead is key to the success of phrase-level typing. Again, unlike our work, the previous decoders were developed only for tap typing.

In summary, our review shows that within the existing literature, most, if not all, efforts have been made to develop better methods for word-gesture typing or better phrase-level decoders for tap typing. Our research advances the existing knowledge by studying phrase-gesture typing and decoding methods.

3 PHRASESWIPE INTERFACE DESIGN

We start by presenting our design of the PhraseSwipe interface on a smartphone and discuss the unique user experience of phrase-gesture typing introduced by the way how text is entered, committed, viewed, and edited.

3.1 Typing

Typing using PhraseSwipe is performed by swiping through all the letters of the words in an intended phrase. However, drawing a gesture that covers all the words in the phrase is not the only option that the users can type. This is because the users may sometimes have to stop at a random word within a phrase and resume to complete the remaining words. This could happen especially in mobile scenarios, where the users can be easily interrupted. Additionally, our implementation also allows the users to gesture a single word, meaning that the users can type using a mixture of word- and phrase-level gestures. As an example, the phrase "I trust your judgment" could be typed using a single phrase or a mixture of shorter phrases, such as "I trust", followed by "your judgment" or a phrase "I trust your", followed by a word "judgment". This provides the users with the flexibility needed in the mobile context.

An important consideration of phrase-gesture typing is the length of the phrases allowed for the users to type. In principle, phrases of any length should be allowed. However, an ideal design should consider optimizing both user experience and system implementation. We chose five in our implementation because a study investigating the usability of phrase-level decoding showed that showing phrases longer than five words could be harder for the users to follow and may introduce a sense of uncertainty about decoding progress [39]. From the system perspective, while longer phrases may, in contrast, lead to better decoding accuracy due to more available context, studies have found that phrases longer than five words may not necessarily lead to significant improvements in decoding accuracy [32].

3.2 Feedback

Aside from showing the trajectory of the finger movement, the output of the decoder needs to be shown to the user as feedback of their input. Several different strategies can be used to control when the feedback is shown. One option is to show the real-time decoding output of an ongoing gesture whenever the finger moves. With this strategy, the users get the most frequent update on the fly but the cost of frequent attention to the output could impair typing speed [26]. An opposite approach presents the feedback only after the gesture is completed (e.g., the user lifting the finger from the touchscreen). The downside is that the user will not see any output candidate when drawing a gesture but the advantage is that they will not be distracted, which may lead to faster typing speed [26] without sacrificing input accuracy [32]. However, this strategy might not be preferred by the user due to the lack of transparency of the decoding progress [39]. We took a midground approach, where feedback is given only when the system thinks the user is swiping across a target letter. This is determined by comparing the real-time speed/jerk with the average speed/jerk of the current gesture. When the current speed is below average (i.e., the user moves significantly slower) and the jerk is below 1/3 of the average jerk (i.e., the user is not actively accelerating/deaccelerating) (value determined through a pilot study), the decoding output of the ongoing swipe is shown to the user on the screen. Our feedback consists of three candidate phrases/words in a vertical layout (Figure 2) ordered by the probability calculated by the decoder (details in the next section). The top-ranked candidate is placed at the bottom of the list to make it closer to the keyboard for the user to see and select.

3.3 Committing Input Text

Tapping one of the three candidates commits the input text (Figure 3a-b). This is the same as the current word-gesture typing interface. Alternatively, the users can skip this action and start the next gesture directly. This way, the top-ranked phrase/word will be committed automatically.

3.4 Editing and Deleting

Editing is needed when an error occurs or when the committed text needs to be revised. With our implementation, the users can first select a target word or multiple adjacent ones and then draw the gesture of a new word to replace the selected one(s). As an attempt to save the users' time from drawing a new gesture, we implemented an auto-correction feature, with which, the system provides the users with a list of candidate words that are similar

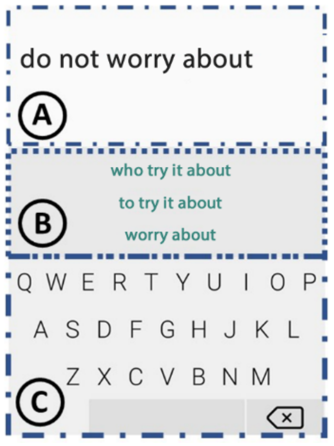

Figure 2: The PhraseSwipe interface is composed of (a) a text field; (b) a list of candidates; and (c) a QWERTY keyboard supporting phrase gesture input.

to the selected one (ranked based on the minimum word distance [6]). If the user's intended word is in the list, the user can simply select it without the need to draw a new gesture (Figure 3c). Lastly, if the user wants to delete the selected word(s), they can simply tap the delete button. If no word is selected, tapping the delete button removes the last word.

4 PHRASESWIPE DECODER

We developed our decoder via a machine-learning-based approach. Our goal was to investigate if an existing neural language model developed for non-keyboard applications can be fine-tuned to satisfy our needs as a phrase gesture decoder. Repurposing an existing model is beneficial because it saves the massive resources needed for a huge model to be built and trained from scratch. In our implementation, our model was built upon a powerful transformer-based neural language model. More specifically, we designed the decoder as an end-to-end framework that translates an input gesture directly into an ordered sequence of words. We aimed to transfer the strong representation capabilities of a neural model proved in other natural language tasks (e.g., GLUE) to keyboard decoding to effectively handle the noises widely existing in users' input gestures and the inherent ambiguity of non-delimiter phrase input.

4.1 Neural Language Model

Decoding a phrase-level input gesture is essentially a sequence-to-sequence task (Seq2Seq), which, in our case, aims at transforming a sequence of touch points into a sequence of English words. To solve the Seq2Seq problem, we adopted an encoder-decoder architecture in our system. The encoder of this architecture turns an input gesture into a hidden vector representation in a continuous space, whereas the decoder reverses the process by turning the vector into an ordered sequence of output words. To boost the performance of our model, we further adopted a pre-trained transformer-based language model, BERT [14], and used it for both encoder and decoder [28]. We chose BERT for its exceptional capabilities of language representation proved in many NLP applications. However, the

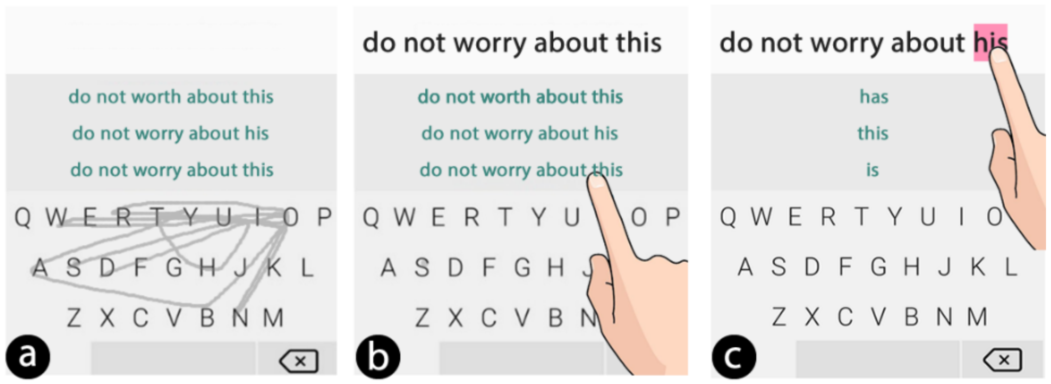

Figure 3: (a) – (b) After a phrase gesture is entered by the user, they can tap one of the items in the candidate list to commit the input text. (c) Auto-correction candidates appear if the user selects a word in the committed text.

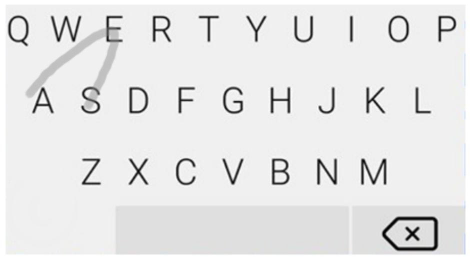

Figure 4: The gesture trajectory of the word "sea" passes the keys "s". "e", "w", and "a" so our system converts the x,y representation of the trajectory into a series of "s", followed by a series of "e", a series of "w", and then a series of "a".

issue is that like many other massively trained language models, BERT was designed to handle language tokens as input, whereas, the input data from PhraseSwipe is touch point from Euclidean space. Our approach to this problem is to use an ordered sequence of English characters instead of integer coordinates as input for the language model. When a user enters a gesture, our system converts the x and y coordinates of the gesture trajectory into a series of nearest keys on the keyboard, represented by an ordered sequence of characters. All 26 characters are set as special tokens to the tokenizer and the encoder of BERT, meaning that each character in the input sequence is an individual special token input to the model. In the example shown in Figure 4, the gesture for the word "sea" was translated into a series of "s", followed by a series of "e", a series of "w", and then a series of "a". The number of each letter that appears in this representation is determined by the number of touch points sampled inside the corresponding key as well as the speed, at which, a gesture is drawn at a certain time and location. Note that a significant advantage of using letter sequences is that letter sequences are independent of keyboard size and ratio. Therefore, our decoder works on any smartphone keyboard.

4.2 Data Collection

Data collection is another challenge due to the lack of phrase gestures available at a massive scale needed to train a neural language model. While such a dataset can be created over time with people's real typing gestures, it is impractical at the current stage of the research, where our goal is to show feasibility. Therefore, we opted to simulate phrase gestures and generate a dataset.

Our method is based on a gesture production model developed by Quinn and Zhai [27] for gesture typing. While the model was initially developed for word gestures, the principle remains applicable to phrase gestures. To produce a phrase gesture, our system first generates a set of intermediate points (via-points) around the center of the keys involved in the gesture. Noises were introduced to the location of the via-points as offsets to the key centers, which were generated under bivariate Gaussian distribution established for tap typing [9]. The trajectory segments connecting two adjacent via-points were generated by following the minimum jerk theory of motor control to minimize the total amount of jerk (the third time derivative of a point) in the produced trajectory [29].

Using our method, we generated 5 million phrase gestures using the Yelp Review Dataset [40]. Each phrase used in our training samples contains up to 5 consecutive words, randomly segmented from the review text. This way, our dataset contained phrases that were semantically incomplete (e.g., instead of "better late than never", we might have "better late than"), increasing the generalizability of the decoder to handle the situation where the user may break a sentence into chunks at will (i.e., no grammatical or semantical principle). We included a million samples for each phrase length ranging from two to five. Further, to allow for word-level gesture decoding, we included a million single words, also randomly selected from the Yelp review dataset. Non-alphabetic characters were removed from the training samples. Each produced gesture contained 500 points generated based on the default Android keyboard on a Nexus 6P smartphone with a 5.7-inch touchscreen (phones of other sizes will also work). We then converted the trajectories into the sequences of nearest characters using the method described above.

Typing errors are inevitable as the users may accidentally skip a key, include an extra key, or substitute a letter for another. To handle these situations and increase the robustness of our decoder against human errors, we alerted the training phrases by randomly injecting these three types of errors. To do so, we traversed all the letters in a phase, and for each letter, we assigned a probability (5%)

Table 1: The percentage of the target words appeared in the top 1 and top 3 entries of the candidate list. The data was obtained with testing phrases picked randomly from the Amazon review dataset, Movie dialog, and Yelp review dataset. The number inside the parentheses indicates the length of the testing phrases. Word Error Rate was calculated using the data from the top-ranked candidates with decoding errors.

Dataset (Phrase length)	Top-1	Top-3	Top-1 Word Error Rate
Movie (1)	89%	93.4%	10.9%
Movie (2)	80.5%	88.1%	13.3%
Movie (3)	76.5%	85.7%	11.9%
Movie (4)	71.7%	82%	11.8%
Movie (5)	65.7%	75%	12.5%
Amazon (1)	89.5%	93.3%	10.4%
Amazon (2)	79.7%	86.5%	13.6%
Amazon (3)	75.4%	82.9%	12.6%
Amazon (4)	69.4%	78.8%	12.5%
Amazon (5)	64.5%	75.6%	12.4%
Yelp (1)	94.8%	97.2%	5.2%
Yelp (2)	88.7%	94.5%	7.5%
Yelp (3)	86.9%	93.1%	6.2%
Yelp (4)	85.1%	91.4%	5.6%
Yelp (5)	82.4%	90.4%	5.5%
Average	80%	87.2%	10%

for that letter to be either missed, substituted, or inserted with an unwanted prefix. The characters used to create the substitution and insertion errors were randomly sampled from adjacent characters.

4.3 Model Training

Both the encoder and decoder of our model architecture were initialized with a pre-trained BERT-base model. Our keyboard decoder was then fine-tuned on our training data for 3 epochs, with an AdamW optimizer [20] at a learning rate of 5e-5. The training was conducted on a 4-GPU machine (NVIDIA Tesla V100), with a total batch size of 32 (8 per device). Our software was implemented using an open-source transformers library [33]. Initial testing showed that the entire system runs at a latency of around 120ms to 160ms including the latencies caused by computation and network communications. This is fast enough to provide real-time feedback needed for PhraseSwipe.

4.4 Evaluation

We tested our decoder with 30000 phrases, within which, 10000 were randomly sampled from the Yelp Review Dataset [40], 10000 were sampled from the Amazon Review Dataset [40], and the remaining 10000 from the Movie Dialog Dataset [2]. Including the testing phrases from the Amazon Review Dataset and Movie Dialog Dataset allowed us to measure how well the decoder can handle out-of-domain input. Within each dataset, we had 2000 samples for each phrase length from two to five words. We also included another 2000 single words. To measure the robustness of our decoder against human errors, we randomly included three types of human errors in the testing phrases as well. We show the results in Table 1.

By averaging the results across all the phrase lengths and tested datasets, we found that 80% of the intended phrases appeared as the top-ranked candidate provided by the decoder. So theoretically speaking, most of the time, the users can directly accept the top candidate without encountering any error. Even with the errors, over 87% of the intended phrases ended up within the top-3 candidates, meaning that users can still find their target words quickly in the suggestion area. Looking deep into the top-ranked candidates with decoding errors, we found a Word Error Rate of around 10% per phrase. The Word Error Rate was calculated by dividing the smallest number of word deletions, insertions, or replacements needed to correct the input text by the number of words in the phrase [8]. The error rate is not high considering our testing phrases were all quite short with no more than five words. When a decoding error occurs, the users can fix it using the editing tools described in Section 3.4.

With the input from the Movie and Amazon dataset, the performance of our decoder was not as good when compared to that of the Yelp dataset but it is still reasonably good. Concerning the question regarding whether a higher level of accuracy can be achieved when a given phrase is entered using a single gesture versus several short ones (i.e., more vs fewer words in the input data). Our result suggests that the answer is yes. Using the results from the Movie data as an example, when the phrases of five words were entered using a single continuous gesture, 65.7% of the targets appeared as the top-ranked candidate (see Table 1 second column), whereas in theory, when entering five consecutive words one by one, the number (score) will drop down to 55.8% ($89\%^5 = 55.8\%$, where, 89% is the percentage of the intended words appeared as the top candidate with word-gesture typing). This finding remains mostly true for both top-1 and top-3 scores. For example, we found that typing a phrase of any length using a single gesture led to a top-1 score of 75%, which is 3% higher than typing using a mixture of shorter phrases or phrases and words. Figure 5 shows the top-1 and top-3 scores obtained for all the possible mixtures of gesture length to type phases of up to five words.

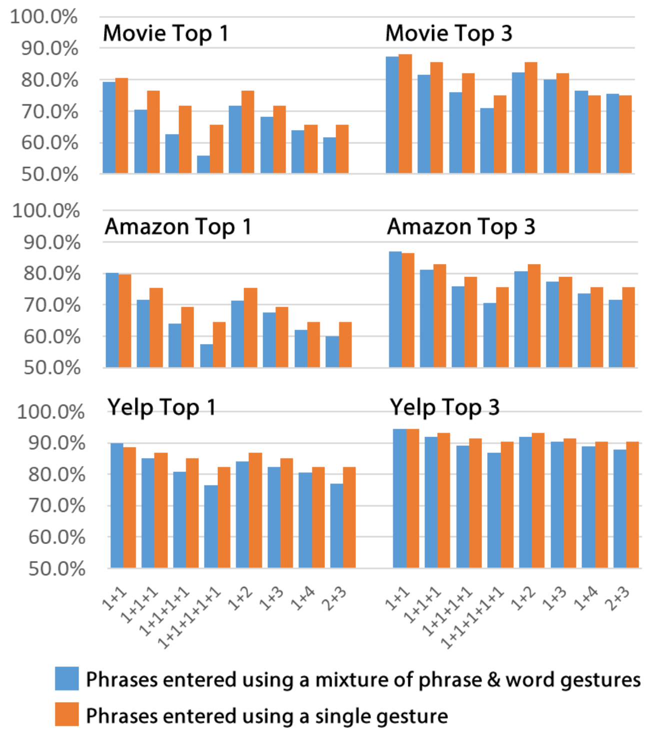

Figure 5: The percentage of the target words appeared in the top 1 and top 3 entries of the candidate list obtained for all the possible mixtures of gestures length to type phrases of up to five words. The x-axis labels show unordered mixtures of phrase and word gestures. For example, "1+1" indicates typing a phrase of two words using two separate word gestures. Similarly, "1+1+1+1+1" indicates typing a phrase of five words using five separate word gestures. "1+3" indicates typing a phrase of four words using a word gesture followed by a phrase gesture of three words. "2+3" indicates typing a phrase of five words using a phrase gesture of two words followed by another phrase gesture of three words.

5 USER STUDY

To further understand the usability of PhraseSwipe, we conducted a user study. The goal of the study was to measure how well people could enter text using our implementation of phrase-gesture typing on a smartphone. We were also interested in learning the users' typing behaviors with this new type of text input method, concerning how they type phrases and coordinate phrase and word gestures to input text. To set a reference for us to better understand the benefits and costs of gesture typing via phrases, we also included word-gesture typing as a baseline.

5.1 Participants

We recruited 12 right-handed participants (9 male and 3 female) aged between 20 and 26. All the participants are familiar with smartphone keyboards and QWERTY layout. One of them has previous experience with word-gesture typing.

5.2 Apparatus and Task Conditions

The study was conducted using a Huawei Nova 8 smartphone with a keyboard of 70.4mm wide and 40mm high. The sampling rate of the touchscreen is 240Hz. All the collected gestures were resampled to 500 points to match the training set of the neural decoder. Redundant touch points caused by pauses were removed. Our phrase-level gesture decoder ran on a separate machine with a GTX 1050TI graphics card (4GB). The smartphone and the machine were connected through WiFi.

During the study, participants sat in a chair in a comfortable position and performed a transcription task with their right index finger under one of the three conditions: (1) *Phrase-Gesture Typing*; (2) *Free-Style Typing*; and (3) *Word-Gesture Typing*. In the *Phrase-Gesture Typing* condition, participants were asked to enter a testing phrase using a single gesture. An experimenter supervising the study made sure that each phrase was entered in this way. In the *Free-Style Typing* condition, participants were not restricted to either method, meaning that they could choose to complete a testing phrase using a single gesture, a series of word gestures, a series of shorter phrase gestures, or a mixture of phrase and word gestures in whatever way they wanted. Lastly, in the *Word-Gesture Typing* condition, participants entered a testing phrase word by word. We implemented word-gesture typing by following the method described in SHARK2 [19]. Note that we didn't include other phrase input methods in the study (e.g., VelociTap [31] or PhraseFlow [38]) as none of them supports gesture typing. There are other variations of word gesture decoder (e.g., SwiftKey [4]) but we chose SHARK2 because it is the most widely adopted gesture typing method. For the corpus, we used the top 15,000 words from the American National Corpus [1], which covers over 95% of common English words. All the information, including the testing phrases, text entered by participants, and the top three candidates generated by our system, were shown on the smartphone screen above the keyboard (Figure 6). In all the conditions, editing was performed using our editing tool described in the previous section.

5.3 Procedure and Design

Prior to the experiment, participants were given time to practice until they felt confident to type using the methods. During the study, participants transcribed 60 phrases (20 per condition) picked randomly from MacKenzie's phrase set [21]. Since the data set does not have single words or phrases of less than 3 words, our testing phrases are all 3 to 5 words long. The same set of 60 phrases was used for all participants. The three testing conditions were counterbalanced among participants. The experimental session lasted around 60 minutes. Participants were encouraged to take breaks whenever they wanted during the study. In total, we collected 12 participants × 20 phrases × 3 conditions = 720 phrases. Upon completion of the study, participants filled out a post-experiment questionnaire where they indicated subjective ratings for Efficiency, Accuracy, Demanding, and Learnability (1: very low, 5: very high) using a continuous numeric scale. Decimal ratings like 2.8 were permitted.

5.4 Result

We analyzed the data using a series of one-way repeated-measures ANOVA and Bonferroni corrections for pair-wise comparisons.

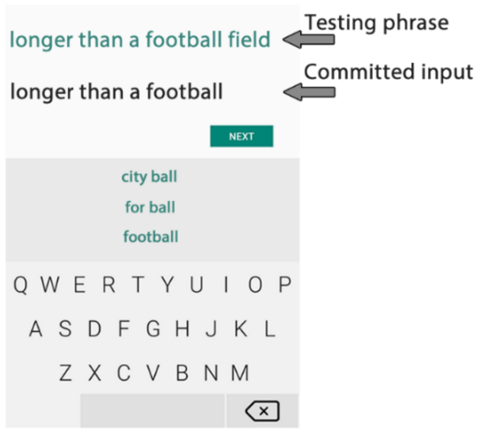

Figure 6: The software interface used in our study.

Mauchly's test did not indicate any violation of sphericity for text entry speed (X^2 (2)=0.59, p=0.74) or error rate (X^2 (2)=2.99, p=0.22).

5.4.1 Text-Entry Speed. ANOVA yielded a significant effect of the typing conditions ($F_{2,\,22} = 12.7$, $p < 0.001$, $\eta_p{}^2 = 0.5$). Post-hoc pairwise comparisons revealed significant differences between *Word-Gesture Typing* and *Phrase-Gesture Typing*, *Word-Gesture Typing* and *Free-Style Typing* (both $p < 0.05$). There was no significant difference between *Phrase-Gesture Typing* and *Free-Style Typing* ($p = 1$).

Overall, the average text entry speed across all the tested conditions was 33.5 WPM (s.d. = 5.5). In particular, participants achieved 31.9 WPM (s.d. = 4.3) using *Word-Gesture Typing*, 34.5 WPM (s.d. = 3.4) using *Phrase-Gesture Typing*, and 34 WPM (s.d. = 3.2) using *Free-Style Typing*.

The speed of *Word-Gesture Typing* was close to what was reported from a large-scale field study (32.2 WPM) in the literature [25]. It was approximately 2.5 WPM slower than phrase-gesture typing. One of the main reasons is that when typing phrase by phrase, the participants did not need to lift their finger from the screen as often as typing word by word, which saved time. Further, we found that the speed of *Free-Style Typing* is on a par with the speed of *Phrase-Gesture Typing*. To understand the reason, we examined the data closely and found that most of the phrases in the *Free-Style Typing* condition were entered using a single gesture despite their length (average number of gestures per phrase is 1.12). After talking to our participants, we realized that most of them found it handy enough to simply draw a single gesture instead of breaking it down into small pieces. Note that the finding of this typing behavior is preliminary as it may not be generalizable to longer phrases but, at least, our result suggests that phrase gestures of up to five words can be performed with ease.

5.4.2 Word Error Rate. As in Section 4.4, Word Error Rate was calculated by dividing the smallest number of word deletions, insertions, or replacements needed to correct the input text by the length of the phrase in word [8]. ANOVA did not show any significant effect of the typing conditions ($F_{2,\,22} = 2.1$, $p = 0.1$, $\eta_p{}^2 = 0.1$). Overall, the average error rate across all the tested conditions was 1% (s.d. = 6). In particular, the error rate for the *Word-Gesture Typing*, *Phrase-Gesture Typing*, and *Free-Style Typing* conditions were 0.3% (s.d. = 0.6), 1.1% (s.d. = 2.1), and 1.6% (s.d. = 2.1) respectively.

Note that the word error rate reported in Section 4.4 appears to be higher than what was found in the user study. This is because Section 4.4 compares the ground truth with the top prediction from the decoder, while in our user study, the participants could choose the best input from multiple candidates. Many tested phrases in the simulation study were semantically incomplete (explained in Section 4.2). This could have also led to a lower performance for the decoder. We did the simulation again using MacKenzie's phrase set and got a 1.5% word error rate. This is well aligned with the uncorrected word error rate reported here.

To better understand the cause of the errors in the *Phrase-Gesture Typing* condition, we looked carefully into how errors were missed by the participants. An important finding was that errors were not obvious to catch in a phrase. This can be explained by the following example, where the intention is to enter the phrase "prescription drugs require a note" but "s" was missed in the word "drugs" in one of the three candidates alongside the correct one. Unless the user inspects them carefully, it was easy to slip. Most errors recorded in our data are of this type. Other examples include typing "his" instead of "this" or typing "broke" instead of "broken".

When an error was caught, editing could be performed by either redrawing a gesture to replace the selected word or using the auto-correction feature if the intended word was provided by the decoder. We found that out of 84 edits that occurred in the *Phrase-Gesture Typing* and *Free-Style Typing* conditions, over 88% of them were carried out using the auto-correction feature, which suggests that our decoder was able to effectively identify the users' target words and provided them to the users if it knew where the error was. The time saved to perform another gesture to fix the error also contributed to the faster typing speed in the *Phrase-Gesture Typing* condition. Considering that the error rates and the number of edits were both low, the decoder seems to do well on handling the gestures from the users, even as our model was trained using artificial gestures.

5.4.3 Typing Behavior. Phrase-gesture typing typically requires the user to swipe longer than word-gesture typing, which may impose extra cognitive overhead on the user, especially when planning for the next movement, because there is more to recall, search, and draw. For example, when a user does not know where to strike next, they may slow down or pause the finger to mentally or visually search for the next key. To assess the cognitive overhead of phrase-gesture typing, we analyzed the participants' typing behaviors using average Finger Movement Speed and Pause Rate. The Finger Movement Speed of a gesture was calculated as the length of the gesture trajectory divided by the corresponding gesture completion time. The Pause Rate of a gesture was calculated as the percentage of the period when the finger moved slower than 1/10 of the finger movement speed.

Our result showed that, on average, the Finger Movement Speed for the *Word-Gesture Typing*, *Phrase-Gesture Typing*, and *Free-Style Typing* conditions was 64.2 mm/s (s.d. = 6.1), 59.4 mm/s (s.d. = 6.4), and 61.2 mm/s (s.d. = 6.6) respectively, showing that participants were 7.5% and 4.6% slower in the *Phrase-Gesture Typing* and *Free-Style Typing* conditions than in the *Word-Gesture Typing* condition.

ANOVO showed there was a significant difference among the three tested conditions ($F_{2,\,22} = 8.6$, $p < 0.05$, $_p^2 = 0.4$). The Pause Rate for the *Word-Gesture Typing*, *Phrase-Gesture Typing*, and *Free-Style Typing* conditions was 13.9% (s.d. = 1.6), 13.2% (s.d. = 0.9), and 13.4%(s.d. = 1.3) respectively. No significant difference was found among the three tested conditions ($F_{2,\,22} = 2.1$, $p = 0.1$, $_p^2 = 0.1$). This result suggests that participants moved their finger slower with the increase of gesture length but didn't stop or hesitate in the middle of a gesture. Although phrase gesture typing seems to introduce extra cognitive overhead on the participants, the impact does not seem to outweigh the benefit as text entry speeds were faster in the two conditions where phrase-gesture typing was used, than in the word-gesture typing condition.

5.4.4 Subjective Feedback. The post-experiment questionnaire filled out by all the participants shows that the users welcomed the unique experience provided by PhraseSwipe. They also indicated a high level of interest in using phrase gesture typing if it was made available on smartphone keyboards. Below, we report our findings in detail. The continuous numeric scale data was analyzed using one-way repeated-measures ANOVA and Bonferroni adjustment for pairwise comparison.

The participants gave an average of around 3.8 (5 being most efficient) to *Phrase-Gesture Typing* and *Free-Style Typing* as the two most efficient methods. These ratings were significantly higher than the ratings for *Word-Gesture Typing* (avg. 3.4; both p < 0.05). All the participants told us that they felt faster when typing phrase by phrase than word by word. This is consistent with our quantitative results described in the previous section. In contrast, the partici-pants gave an average of 3.9 (5 being most accurate) to *Word-Gesture Typing* as the most accurate method among the three. This rating was significantly higher than both *Phrase-Gesture Typing* (avg. 3.6; p < 0.05) and *Free-Style Typing* (avg. 3.5; p < 0.05) as the participants were more confident that fewer errors were left uncorrected when typing word by word.

Additionally, the participants found that none of the three methods were physically demanding to use but they rated *Phrase-Gesture Typing* significantly less demanding (avg. 2.1 with 1 being the least demanding) than *Word-Gesture Typing* (avg. 2.6). Over half of the participants explicitly said that they found phrase gestures easy to draw. For example, P1 said that "*I liked that I could type with-out raising my finger between words*". P12 told us that "*It was very handy to type without worrying about using the space key*". P6 and P9 mentioned that drawing phrase gestures reminded them of hand-writing, which was a part of the reasons why they liked it. Finally, the participants gave *Word-Gesture Typing* an average of 3.9 (5 be-ing the most learnable) in response to "rate each method for its learnability". The rating was significantly higher than the rating of *Phrase-Gesture Typing* (avg. 3.4; p < 0.05). As expected, learning how to use phrase-gesture typing may be a burden at the beginning for some users but for the others, they found it easy and fun to learn.

6 DISCUSSION, LIMITATIONS, AND FUTURE WORK

We present insights we learned from the execution of this research, discuss the limitations, and propose future work.

6.1 Usability and Practicality

Phrase-gesture typing as a less understood text input method warrants deeper investigations in the future. In this work, we took an initial step toward demonstrating its technical feasibility and understanding its usability. Through our experiment, we show that users could quickly enter phrases of up to five words using phrase gestures. Beyond the knowledge provided in this work, an interesting avenue for future research is to investigate how gesture length may have an impact on the performance and usability of this type of text input method. Questions related to the efficiency, accuracy, learnability, or fatigue caused by phrase gestures of different lengths are all important and need to be answered before the user experience can be better optimized. One of the unique benefits of typing using gestures, at least word by word, is skill acquisition over time, which allows the users to eventually develop themselves into experts for faster typing speed. Thus, an exciting research question to be answered in the future concerns whether the transition of expertise occurs in phrase-gesture typing and how to facilitate skill transition to better serve the needs of the users.

Our current implementation runs the decoder on a server. Though our study participants didn't report any noticeable network delay, it could be a problem in practical settings as the inference time of a large model and network delay may occur at a scale large enough to impact their typing experience. We are currently investigating methods that could deliver the promise for on-device inference (e.g., model compression, structure search).

6.2 Interface Design

As a key influencer of usability, the interface design of a phrase gesture keyboard also warrants more investigation. Our immediate future research concerns the optimal number of candidates that should be shown to the user. Our current implementation follows the standard developed for tap and word-gesture typing by showing three candidates. The tradeoff is obvious as extra screen real estate needs to be occupied to accommodate the increasing amount of text in the output phrases. While it is less of an issue for today's smartphones as the screens are large, it may potentially become a problem on devices with smaller screens, such as smartwatches. As a part of our future research, we will investigate how much the size of the candidate list shown to the user may affect their typing performance and whether an optimal size exists for the general population.

Aside from the candidate list, we are also interested in investigating the ways, in which, how feedback is provided may impact the performance of phrase-gesture typing. While our current implementation shows intermediate feedback during the execution of a gesture, the study shows that the users often chose to wait until the completion of a gesture to examine the output of the system. While a part of the reason was that maintaining close attention to the output may slow down their typing speed, another possible reason could be the lack of auto-complete in our current prototype. Thus, the participants had no motivation to check the feedback during typing. So more research needs to be done in this space to better understand the impact of intermediate feedback and its timing.

The current implementation of PhraseSwipe supports word-level editing for correction and deletion. In some situations, character-level editing may be needed for additional flexibility and efficiency. While not included in our current prototype, character-level editing can be supported with an addition of a cursor that is triggerable through a long touch, similar to what is widely available in smartphone keyboards.

6.3 Decoder

Our decoder was trained with artificial gestures generated using a computer. Through the controlled user study, we showed that it was a cost-efficient method for the rapid development of an ef-fective neural decoder. However, the limit is that it is unlikely for our current model to be able to handle individual variances among different users or user groups without training data from real users. A personalized keyboard decode could, in theory, improve the per-formance of our system for individual users. Future work will focus on data collection using our prototype to acquire phrase gesture data at scale. We are also interested in developing an adaptive al-gorithm that can effectively shift a general decoder to a personal one. Further, including the committed text as input to the decoder may potentially improve the decoding accuracy. It is our plan for the future to investigate ways to improve our decoding methods by including the data beyond the scope of the current phrases.

In principle, our decoder could work on phrases of any length. While in this current study, we only focused on phrases of five words or less, we see it as an interesting opportunity in the future to investigate the decoding performance and user behaviors on longer phrases. Possible research questions include how to maintain a high decoding accuracy despite the length of input phrases, and whether and when users would prefer typing using long phrase gestures.

Some of the obvious language issues were not well handled by the current decoder. As such, grammatically wrong phrases could be found in the candidate list. For example, the phrase "the treasury department its broke" ("its" is the error) was shown alongside the correct one, in which, "its" was replaced by the correct word "is". As shown in our study, we cannot rely on the users to capture such errors as they are often not obvious inside a phrase. If the users picked the candidates with errors, they may end up spending extra time fixing the errors or in the worst case, the errors may be left uncorrected. Future research will investigate methods to improve the accuracy of our decoder.

Lastly, we acknowledge that phrase gesture typing can be performed on Gboard [3] in Chinese. Our work provides insights into decoding phrase gestures in English, which has unique challenges. For example, in the most used Hanzi characters for Simplified Chinese (6763 from GB2312), there are only 413 distinct Pinyin syllables. In contrast, there are 15831 in English. Further, unlike English, most Pinyin syllables consist of initials and finals that are very different from each other (e.g., in "zhan", "zh" is initial, "an" is final). Both make it relatively easy to segment a non-delimiter letter sequence into separate Pinyin syllables and decode them with a language model. We see that other languages may have similar characteristics as English and may benefit from our research.

7 CONCLUSION

In this paper, we created a new gesture typing method that allows the users to enter text using phrase gestures. We call this method phrase-gesture typing. Unlike word-gesture typing, where text is entered word by word, phrase-gesture typing allows text to be entered phrase by phrase. Through a prototype developed for smartphones, we were able to demonstrate the technical feasibility of phrase-gesture decoding and understand the usability of text input through phrase gestures. Our prototype is composed of a frontend interface designed specifically for phrase-gesture typing and a backend decoder developed based on a transformer-based neural language model. We showed that phrase-level decoding can achieve higher accuracy than word-level decoding in gesture typing. We also showed that training the model using artificial gestures generated using a computer is a cost-efficient way to develop an effective neural decoder for phrase-gesture keyboards. Lastly, the result from our user study shows that users were able to type faster using phrase gestures than word gestures. Our findings provide the important groundwork for the future development of new text input methods on mobile devices.

REFERENCES

[1] American National Corpus. Retrieved April 2, 2022 from http://www.anc.org/.

[2] Cornell Movie–Dialogs Corpus. Retrieved April 2, 2022 from https://www.cs.cornell.edu/~cristian/Cornell_Movie-Dialogs_Corpus.html.

[3] Gboard - the Google Keyboard. Retrieved April 2, 2022 from https://play.google.com/store/apps/details?id=com.google.android.inputmethod.latin&hl=en_US&gl=US.

[4] Microsoft SwiftKey Keyboard. Retrieved April 2, 2022 from https://play.google.com/store/apps/details?id=com.touchtype.swiftkey&hl=en_US&gl=US.

[5] Ouais Alsharif, Tom Ouyang, Françoise Beaufays, Shumin Zhai, Thomas Breuel and Johan Schalkwyk. 2015. Long short term memory neural network for keyboard gesture decoding. In 2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), IEEE, 2076-2080.

[6] Xiaojun Bi, Shiri Azenkot, Kurt Partridge and Shumin Zhai. 2013. Octopus: evaluating touchscreen keyboard correction and recognition algorithms via remulation. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI'13), ACM, 543-552.

[7] Xiaojun Bi, Ciprian Chelba, Tom Ouyang, Kurt Partridge and Shumin Zhai. 2012. Bimanual gesture keyboard. In Proceedings of the 25th annual ACM symposium on User interface software and technology (UIST'12), ACM, 137-146.

[8] Xiaojun Bi and Shumin Zhai. 2016. IJQwerty: What difference does one key change make? Gesture typing keyboard optimization bounded by one key position change from Qwerty. In Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems (CHI'16), ACM, 49-58.

[9] Xiaojun Bi and Shumin Zhai. 2016. Predicting finger-touch accuracy based on the dual Gaussian distribution model. In Proceedings of the 29th Annual Symposium on User Interface Software and Technology (UIST'16), 313-319.

[10] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry and Amanda Askell. 2020. Language models are few-shot learners. arXiv preprint arXiv:2005.14165.

[11] Ciprian Chelba, Mohammad Norouzi and Samy Bengio. 2017. N-gram language modeling using recurrent neural network estimation. arXiv preprint arXiv:1703.10724.

[12] Sibo Chen, Junce Wang, Santiago Guerra, Neha Mittal and Soravis Prakkamakul. 2019. Exploring word-gesture text entry techniques in virtual reality. In Extended Abstracts of the 2019 CHI Conference on Human Factors in Computing Systems (CHI EA'19), Association for Computing Machinery, Glasgow, Scotland Uk, Paper LBW0233.

[13] Wenzhe Cui, Jingjie Zheng, Blaine Lewis, Daniel Vogel and Xiaojun Bi. 2019. HotStrokes: word-gesture shortcuts on a trackpad. In Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems (CHI'19), Association for Computing Machinery, Paper 165.

[14] Jacob Devlin, Ming-Wei Chang, Kenton Lee and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805.

[15] 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, 194-195.

[16] Asela Gunawardana, Tim Paek and Christopher Meek. 2010. Usability guided key-target resizing for soft keyboards. In Proceedings of the 15th international conference on Intelligent user interfaces (IUI'10), Association for Computing Machinery, Hong Kong, China, 111–118.

[17] Aakar Gupta, Cheng Ji, Hui-Shyong Yeo, Aaron Quigley and Daniel Vogel. 2019. RotoSwype: word-gesture typing using a ring. In Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems (CHI'19), Association for Computing Machinery, Paper 14.

[18] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. Neural computation, 9 (8). 1735-1780.

[19] 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, 43-52.

[20] Ilya Loshchilov and Frank Hutter. 2017. Decoupled weight decay regularization. arXiv preprint arXiv:1711.05101.

[21] I Scott MacKenzie and Shawn X Zhang. 1999. The design and evaluation of a high-performance soft keyboard. In Proceedings of the SIGCHI conference on Human factors in computing systems (CHI'99), 25-31.

[22] Anders Markussen, Mikkel Rønne Jakobsen and Kasper Hornbæk. 2014. Vulture: a mid-air word-gesture keyboard. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI'14), ACM, 1073-1082.

[23] Mehryar Mohri. 1997. Finite-state transducers in language and speech processing. Computational linguistics, 23 (2). 269-311.

[24] Tom Ouyang, David Rybach, Françoise Beaufays and Michael Riley. 2017. Mobile keyboard input decoding with finite-state transducers. arXiv preprint arXiv: 1704.03987.

[25] 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 (Mobile-HCI'19), 1-12.

[26] 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 (CHI'16), 83-88.

[27] Philip Quinn and Shumin Zhai. 2018. Modeling Gesture-Typing Movements. Human-Computer Interaction, 33 (3). 234-280.

[28] Sascha Rothe, Shashi Narayan and Aliaksei Severyn. 2020. Leveraging pre-trained checkpoints for sequence generation tasks. Transactions of the Association for Computational Linguistics 8. 264-280.

[29] Emanuel Todorov and Michael I Jordan. 1998. Smoothness maximization along a predefined path accurately predicts the speed profiles of complex arm movements. Journal of Neurophysiology, 80 (2). 696-714.

[30] 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 (CHI'18), 1-12.

[31] 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 (CHI'19), 1-14.

[32] 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 (CHI'15), 659-668.

[33] Thomas Wolf, Julien Chaumond, Lysandre Debut, Victor Sanh, Clement Delangue, Anthony Moi, Pierric Cistac, Morgan Funtowicz, Joe Davison and Sam Shleifer. 2020. Transformers: State-of-the-art natural language processing. In Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations (EMNLP System Demonstration'20), 38-45.

[34] Hui-Shyong Yeo, Xiao-Shen Phang, Steven J. Castellucci, Per Ola Kristensson and Aaron Quigley. 2017. Investigating tilt-based gesture keyboard entry for single-handed text entry on large devices. In Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems (CHI'17), Association for Computing Machinery, 4194–4202.

[35] Chun Yu, Yizheng Gu, Zhican Yang, Xin Yi, Hengliang Luo and Yuanchun Shi. 2017. Tap, dwell or gesture? Exploring head-based text entry techniques for HMDs. In Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems (CHI'17), Association for Computing Machinery, 4479–4488.

[36] Shumin Zhai and Per-Ola Kristensson. 2003. Shorthand writing on stylus keyboard. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI'03), Association for Computing Machinery, Ft. Lauderdale, Florida, USA, 97–104.

[37] Shumin Zhai and Per Ola Kristensson. 2012. The word-gesture keyboard: reimagining keyboard interaction. Commun. ACM, 55 (9). 91–101. 10.1145/2330667.2330689

[38] Shumin Zhai, Per Ola Kristensson, Pengjun Gong, Michael Greiner, Shilei Allen Peng, Liang Mico Liu and Anthony Dunnigan. 2009. Shapewriter on the iphone: from the laboratory to the real world. In CHI '09 Extended Abstracts on Human Factors in Computing Systems (CHI EA'09), Association for Computing Machinery, 2667–2670.

[39] 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 (CHI'21), 1-13.

[40] Xiang Zhang, Junbo Zhao and Yann LeCun. 2015. Character-level convolutional networks for text classification. In Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1 (NeurIPS'15), MIT Press, Montreal, Canada, 649–657.

[41] Suwen Zhu, Jingjie Zheng, Shumin Zhai and Xiaojun Bi. 2019. i'sFree: Eyes-free gesture typing via a touch-enabled remote control. In Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems (CHI'19), Association for Computing Machinery, Paper 448