TypeAnywhere: A QWERTY-Based Text Entry Solution for Ubiquitous Computing

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

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

Jacob O. Wobbrock
The Information School
University of Washington
Seattle, WA
wobbrock@uw.edu

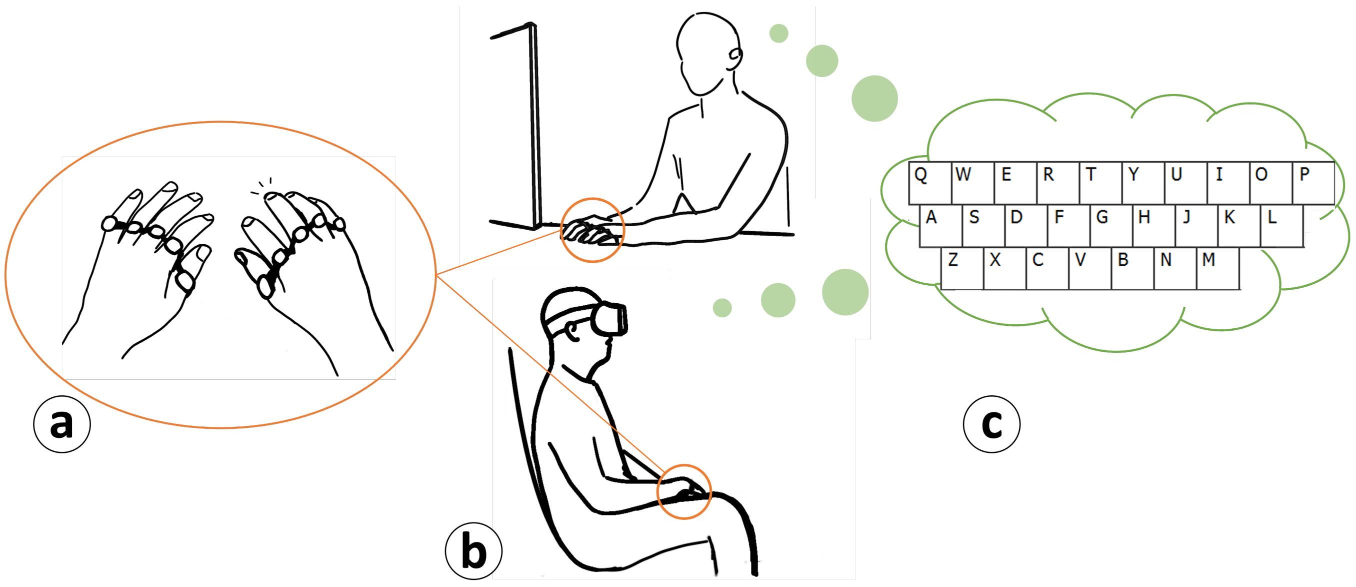

Figure 1: TypeAnywhere's core concepts. (a) Users wear lightweight devices on their fingers that detect finger taps. (b) With TypeAnywhere, users can type on any surface such as a tabletop or their lap. (c) Users type with their own QWERTY key-to-finger mappings, resembling physical keyboard typing.

ABSTRACT

We present a QWERTY-based text entry system, *TypeAnywhere*, for use in off-desktop computing environments. Using a wearable device that can detect finger taps, users can leverage their touch-typing skills from physical keyboards to perform text entry on any surface. TypeAnywhere decodes typing sequences based only on finger-tap sequences without relying on tap locations. To achieve optimal decoding performance, we trained a neural language model and achieved a 1.6% character error rate (CER) in an offline evaluation, compared to a 5.3% CER from a traditional *n*-gram language model. Our user study showed that participants achieved an average performance of 70.6 WPM, or 80.4% of their physical keyboard speed, and 1.50% CER after 2.5 hours of practice over five days on a table surface. They also achieved 43.9 WPM and 1.37% CER when typing on their laps. Our results demonstrate the strong potential of QWERTY typing as a ubiquitous text entry solution.

CHI '22, April 29-May 5, 2022, New Orleans, LA, USA
© 2022 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-9157-3/22/04.
https://doi.org/10.1145/3491102.3517686

CCS CONCEPTS

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

KEYWORDS

Text entry, neural networks, ubiquitous computing, wearable, QWERTY.

ACM Reference Format:
Mingrui "Ray" Zhang, Shumin Zhai, and Jacob O. Wobbrock. 2022. TypeAnywhere: A QWERTY-Based Text Entry Solution for Ubiquitous Computing. In *CHI Conference on Human Factors in Computing Systems (CHI '22), April 29-May 5, 2022, New Orleans, LA, USA.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3491102.3517686

1 INTRODUCTION

Computing is now almost ubiquitous [54], from wearables and Internet-of-Things devices to smartphones and virtual reality headsets, computers are woven into nearly every aspect of modern life. Since the earliest days of interactive computing, a fundamental method of communication with computers has been text entry. But as we move further off the desktop, off tablets, off smartphones, and indeed onto devices of almost every kind, satisfying text entry solutions remain elusive [69]. There is as-yet no unified text entry solution that is efficient, effective, and always available. Existing text entry solutions either are not mobile enough (e.g., a full-size hardware keyboard), or require the user to learn new typing skills (e.g., [26, 33, 48, 64]), or offer painfully low throughput (e.g., watch-based methods [5, 17, 55]). While speech-based input is sometimes an option, it is often not socially acceptable and can be a risk to privacy [37].

QWERTY-based touch-typing on physical keyboards remains the most common text input skill of computer users [11], utilizing multiple hands and fingers and relying on tacit knowledge of learned key positions. Although the QWERTY layout was not invented for optimal speed [9], the fact that people are so familiar with the layout has motivated text entry researchers to build upon QWERTY [2, 19, 53, 60, 61, 65]. We therefore want to leverage the physical QWERTY keyboard typing experience in our effort to discover a solution for ubiquitous computing text entry, but without presuming the availability of mechanical keys. What if we could type on an imaginary QWERTY keyboard on *any* surface, just like we type a physical keyboard?

To explore this question, our work developed *TypeAnywhere*, a QWERTY-based text entry solution for everyday use in ubiquitous computing contexts (Figure 1). TypeAnywhere employs wearable sensors on two hands that detect finger-tap actions, a decoder that converts tap sequences into text, and a corresponding interface for text editing. TypeAnywhere's hardware is the commercial *Tap Strap*[1] product, which uses accelerometers for tap detection. We feed a detected finger-tap sequence to a neural decoder modified from the BERT model [10], which then displays the output text on the typing interface. Similar to *Type, then Correct* [71], we also designed a text correction interaction for TypeAnywhere that avoids the need for cursor navigation.

An important feature of TypeAnywhere is that it does not rely on position information to decode the intended letters being typed by the user. Rather, TypeAnywhere relies only on the index of the finger being tapped and the context provided by the text entered thus far. Because TypeAnywhere only uses the index of the tapping finger without relying on (x, y) position information, the user can perform the tap at any location, so long as they use the correct finger. This scheme simplifies the design space, enabling us to both generalize the decoder easily for different finger-to-key mappings, and to achieve high accuracy on text decoding. Unlike other QWERTY-based text entry research projects [14, 15, 63, 65, 76], which had to conduct user studies to collect spatial information for model-building, we were able to train the model without collecting data from user studies. Instead, we curated the training data by converting each letter in a phrase set to its corresponding finger ID. In this way, we were able to train the model on a large text corpus containing over 3.6 million samples [75] and adapt different finger-to-key mappings by altering the finger IDs in the training set. We performed computational evaluations on the neural decoder, achieving a 1.6% character error rate (CER) on the Cornell Movie-Dialogue Corpus [8], compared to a 5.3% CER using a conventional n-gram language model.

Without the tactile feedback provided by mechanical keys, whether users can effectively take advantage of TypeAnywhere's neural decoding power required us to conduct an empirical evaluation. So to evaluate TypeAnywhere, we performed a longitudinal user study with five participants over five days. In our study, participants used TypeAnywhere on a table surface for a half-hour each day, achieving an average of 70.6 WPM and 1.50% CER. The best performer achieved 91.4 WPM and 3.15% CER after five days of use (or 2.5 total hours). To test the convenience of TypeAnywhere, participants also performed typing on their laps, reaching 43.9 WPM and 1.37% CER. Our results demonstrate that users can quickly learn TypeAnywhere with minimal practice and achieve high performance after practicing for a total of 2.5 hours. As a result, we think TypeAnywhere has potential as a text entry solution for ubiquitous computing environments. To support TypeAnywhere's adoption, we provide an open-source implementation including our neural language model.[2]

2 RELATED WORK

There have been many projects aiming to provide alternative text entry solutions beyond the traditional desktop [67]. In this section, we first review existing text entry interactions with wearable devices. We then offer a deeper look at QWERTY-style interactions that mimic the physical keyboard typing experience. Finally, we provide a brief review of recent advances in neural decoders and language models.

2.1 Text Entry with Wearable Devices

Text entry research for wearable devices has gained a lot of attention because of the potential for always available input. One text entry design for wearables involves typing on an external device. For example, Twiddler [33] used a one-handed chording keyboard, where the user gripped the device and pressed multiple keys simultaneously to enter a character. The reported average speed was 26 WPM after 400 minutes (6.67 hours) of practice. Yu et al. [66] demonstrated one-dimensional gestures for text entry on smart glasses. Smartwatch text entry also falls into the wearables category [19, 24, 29, 39, 63, 64], where the user performs text entry through the interface of the watch device. For example, COMPASS [64] utilized a rotary dial on the watch for character selection, reaching 12.5 WPM with a dynamically positioned cursor. WatchWriter [19] implemented both tap- and swipe-based [26] text entry on a watch, achieving 24 WPM with 3.7% character error rate (CER). Velocitap [51] studied committing one word, multiple words, and one sentence at a time for smartwatch input.

Another category of text entry interactions for wearables is freehand typing with external sensors. For example, TipText [61] (average speed 13.3 WPM) and BiTipText [60] mapped the QWERTY

[1]https://www.tapwithus.com/

[2]https://github.com/DrustZ/TenFingerTyping

layout onto a user's index finger and enabled "subtle" text entry with thumb-to-index taps sensed by capacitive overlays. BiTipText reached an average speed of 23.4 WPM. QwertyRing [21] was a ring that detected taps of the index finger and reached 21 WPM. Finger-T9 [58] (average speed 5.42 WPM) mapped the layout of a 9-key number pad to a user's finger segments for the thumb to tap. WrisText [17] detected wrist motions by proximity sensors and enabled joystick-like whirling input for text entry, reaching an average of 15.2 WPM. PinchType [12] (average speed 12.54 WPM) grouped letters that would be typed by the same finger on a QWERTY keyboard, and detected finger pinches for QWERTY-style input. SCURRY [25] was a glove-like device that detected finger taps with gyroscopes and accelerometers, and users entered text by pointing and clicking. ARKB [28] projected a see-through augmented reality keyboard using a head-mounted display, and tracked finger movements through cameras. TelemetRing [49] was a wireless ring-shaped keyboard that detected finger taps, similar to the finger-worn devices used in this project. However, TelemetRing used a chording system for text entry and did not report users' performance (as neither did SCURRY and ARKB).

In sum, the search for effective text entry methods with wearable devices is both longstanding and ongoing. Challenges to finding satisfactory methods include learning new skills [17, 33, 66] and slow performance (i.e., less than 20 WPM). One promising direction for overcoming these challenges is to leverage users' existing QWERTY keyboard typing skills, as we do in this work.

2.2 QWERTY Touch Typing

Ten-finger touch-typing on a QWERTY layout has been studied extensively since at least the 1970s [20, 43–45]. To understand how people perform touch typing with different finger-to-key mappings, Feit et al. [13] observed 30 participants typing on a physical keyboard. They found that people who did not type with the standard finger-to-key mapping (Figure 2)[3] could also reach high levels of performance, over 70 WPM. Through clustering they identified six mapping strategies for the right hand and four strategies for the left. In a follow-up study [11], they found that rollover key-pressing was a key factor for fast typing, and faster typists often used more fingers to type. In other work, Findlater et al. [15] performed empirical studies of ten-finger typing on touch screens and found that the typing speed (58.5 WPM) was 31% slower than the physical keyboard without any feedback.

There are several projects also trying to enable QWERTY-style 10-finger typing without using a keyboard. The most similar concept to TypeAnywhere is from Goldstein et al. [16], where they proposed a QWERTY-style typing interaction using a pair of gloves. However, their evaluation was only conducted on a mock-up without a functioning prototype. In our work, we realized the TypeAnywhere concept in a fully working system, including with personalization, editing interactions, and advanced language models. The Canesta prototype [42] projected a keyboard on a table to enable QWERTY-style typing. TOAST [46] was an eyes-free keyboard that enabled the user to type on large touch screens without a visible keyboard. With its Markov-Bayesian algorithm for spatial decoding,

[3]The standard finger-to-key mapping can be found at https://agilefingers.com/articles/touch-typing-finger-placement

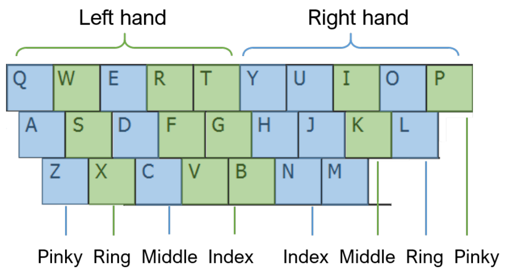

Figure 2: The standard finger-to-key mapping of the letters on a QWERTY keyboard. The mapping of the thumbs to the spacebar is not shown.

participants reached an average speed of 44.6 WPM. However, this technique could only be applied on devices with large touch screens. Relatedly, Findlater et al. [14] showed how machine learning could be used to design touch screen keyboards on interactive tabletops that adapt at runtime to evolving finger positions. ATK [65] provided for mid-air typing using computer vision hand-tracking, reaching an average speed of 29.2 WPM. However, mid-air typing lacks tactile feedback and can easily cause fatigue. Richardson et al. [41] implemented a vision-based text entry system relevant to this project, where hand motion captured by cameras was fed into a neural network decoder that then output the decoded text corrected by another neural language model. Although they did not perform a user study to evaluate their system, the offline CER was reported to be 2.22%. While promising, such computer vision-based systems still require fixed-position cameras overlooking the hands wearing arrays of markers, making such systems impractical for ubiquitous computing.

Another common barrier for off-desktop QWERTY-style 10-finger typing systems is the training usually required. All of the projects mentioned above need to collect typing data from users to train spatial models [41]. This need for training data can hinder the generalizability of the approaches, as people type with different finger motion characteristics and even different finger-to-key mappings [13]. In contrast, TypeAnywhere only utilizes the finger-tap sequence for decoding, eliminating the need for motion data collection for model training. The flexibility of TypeAnywhere's minimal training process makes it easy to generalize through different finger-to-key mapping strategies.

2.3 Statistical Language Decoders

Traditional statistical decoding methods for soft keyboards were proposed by Goodman et al. [18], who applied a bivariate Gaussian distribution to model the touch positions of each key and a character *n*-gram language model for auto-correction. Subsequent work generally has focused on improving the spatial model, such as adding constraints on the key regions [22]. The main language modeling method for text entry remains *n*-gram models [4, 19, 53], although some projects have explored phrase-level decoding where the model decodes the input not only by its preceding text, but also by the text that follows [53, 73].

Recent advances in Natural Language Processing (NLP) have demonstrated the power of neural networks. Deep neural network language models (NNLM) trained on large amounts of text have achieved impressively low perplexity [3, 10, 50], and state-of-the-art performance on many language understanding tasks. Unlike traditional *n*-gram models, neural networks can take more textual context into account during the modeling process. In this work, we applied the BERT [10] language model for input sequence decoding. The model was pre-trained on 3.3 billion words and fine-tuned for the finger-to-text task with extra training data, which is explained in Section 4.3, below.

3 ENABLING THE TYPEANYWHERE EXPERIENCE

TypeAnywhere aims to provide users with a QWERTY-style typing experience similar to typing on a physical QWERTY keyboard, except that typing occurs on any surface without the benefit of mechanical keys. The TypeAnywhere system contains the following building blocks: the hardware that detects finger taps, the neural language model that decodes the input sequence into text, and the interface that allows text composition and editing. In this section, we describe TypeAnywhere's interaction and user interface. In Section 4, we describe TypeAnywhere's neural language decoder.

3.1 Typing Interaction

TypeAnywhere's hardware interface comprises two wearable devices called *Tap Straps*,[4] which each have five connected rings that can be worn on each finger (Figure 3). Each ring contains a three-axis accelerometer that detects when a finger performs a tap action, the detection of which it is reported to have over 98% accuracy.[5] The intended purpose of Tap Strap is for one-handed text entry with a proprietary chording system.[6] https://www.tapwithus.com/wp-content/uploads/2018/09/Tap-Alphabet-Glossary-2.pdf As prior chording-based text entry systems have shown (e.g., [33]), chords can take considerable time to learn and master. The reported best performance using the default Tap Strap chording system is 62 WPM, which was achieved after three weeks' practice.[7] To the best of our knowledge, Tap Strap is the most affordable and accurate commodity device for finger-tap detection on the market. Because of Tap Strap's small form factor, users can wear the device comfortably. However, due to the requirement that users learn a proprietary one-handed chording scheme, Tap Strap's widespread adoption seems extremely unlikely absent efforts like ours to enable familiar QWERTY-style typing.

TypeAnywhere's typing interaction resembles the same typing interaction as one would use on a physical QWERTY keyboard. Specifically, users perform finger taps based on their own personal finger-to-key mappings, and the most likely character is produced. Users can perform finger taps on hard surfaces such as tabletops or walls, or on soft surfaces such as their stomachs or laps. The tap of the two thumbs is mapped to the spacebar, as most people use a thumb for spacebar pressing [15]. Essential text editing functions are also mapped to certain multi-finger chords, as shown in Figure 4.

[4]https://www.tapwithus.com/
[5]https://www.tapwithus.com/how-to-the-tap-strap-works/
[6]The default chording scheme of TapStrap is provided at
[7]See https://www.tapwithus.com/wpm-contest. Note that this is advertised as a commercial contest and the reported performance is therefore the best achieved from among all customers.

The backspace gesture was the same as the default Tap Strap gesture, and we assigned symmetrical gestures to both hands to improve learnability [1]. We also assigned gestures only to neighboring fingers (for example, index + middle + ring) to make them easy to perform. We did not assign backspace to any specific finger because (1) each finger was already assigned with a set of letters, and (2) people were inconsistent with which finger they used to press backspace [13]. Assigning a backspace finger would add too much ambiguity to the typing process, as there was no clear indication of whether the user wanted to type a letter or backspace text.

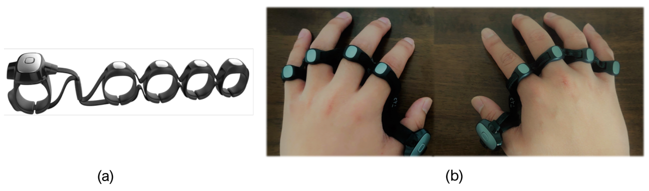

Figure 3: The hardware used in TypeAnywhere. (a) The Tap Strap device. (b) The user wearing two Tap Strap devices.

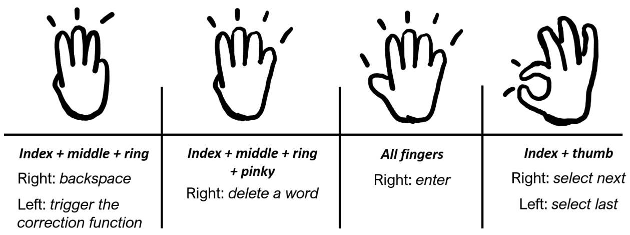

Figure 4: The tap-chords for text editing in TypeAnywhere. For the rightmost gesture, which performs candidate selection from a word list, the user can either tap the thumb and index finger, or perform a pinch gesture.

3.2 Typing Interface

To evaluate TypeAnywhere's interaction, we designed a web app implemented in JavaScript as an evaluative user interface (Figure 5). There are four components of this interface: (1) the *test area*, which displays the target string for evaluation, used in lab studies only; (2) the *commit area*, which displays the committed text corresponding to the text view in apps; (3) the *compose area*, which contains the text that is being decoded; and (4) the *candidate list*, which displays the word candidates of the current typing sequence. Alternatively, the *compose area* could be integrated into the *commit area* by differentiating its content with underlines or different colors [73].

When the user starts typing, the finger-tap sequence is decoded in real-time and the output text is displayed in the *compose area*. Text in the *compose area* might change as the user taps more letters because the decoder can utilize the full tapping sequence as context to improve decoding quality. For example, if the user taps the right index finger (YHNUJM) and then the left middle finger (EDC) under the standard finger-to-key mapping, the decoded text will be "me".

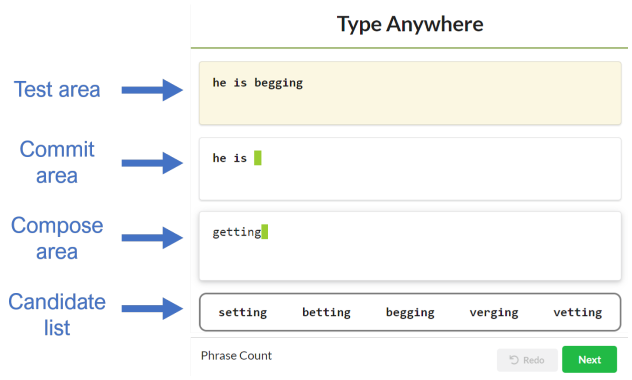

Figure 5: The TypeAnywhere web app. There are four main components in the interface. Note that the test area, redo button, and next button are for user study purposes only, and would not be present when just using TypeAnywhere.

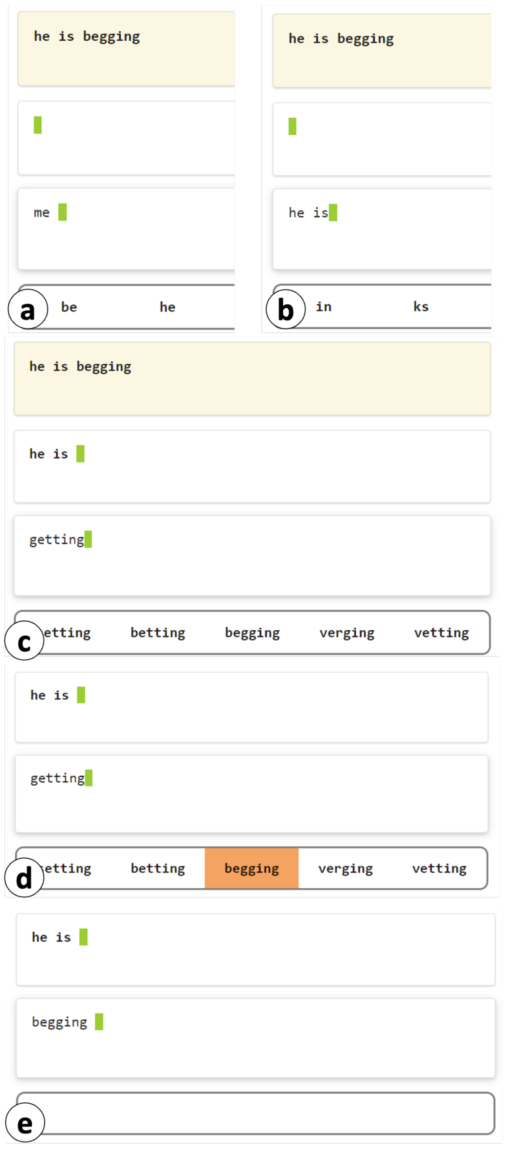

Figure 6: The typing flow of TypeAnywhere. The user is typing the sentence "he is begging" with the standard finger-to-key mapping. (a) After typing the first word, the decoder outputs "me" as the most probable text. (b) With more input context, the decoder is able to decode the tap sequence to "he is". (c) The user first double-taps space to commit "he is" to the *commit area*, then types the last word decoded as "getting". (d) The user then performs a selection gesture to trigger candidate selection and navigate to "begging". (e) The word "begging" is now entered in the *compose area* after the user presses space.

If the user continues typing with a thumb, right middle finger, and then left ring finger, the decoded text becomes "he is".

When the user is ready to "commit" a string of decoded text, she can tap the space key twice to output the text from the *compose area* to the *commit area*. This type of design is commonly seen in typing interfaces for Asian languages, such as Chinese and Japanese, which usually contain a temporary textbox holding intermediate input before it is committed. The user can also delete the text in the *commit area* if the *compose area* is empty. An example of the user's typing flow is depicted in Figure 6.

The user can also select a candidate for the currently typed word from a candidate selection list as shown in Figure 6d. If the user performs the selection gesture shown at the far right of Figure 4, the candidate selection function will be triggered and the candidate word will be highlighted in orange. The user can use the selection gesture to navigate through the candidates and then tap space to select the candidate, which copies it into the *compose area* (Figure 6e); or, the user can tap the backspace chord to cancel the candidate selection. (Candidate word lists raise the specter of out-of-vocabulary words. We describe how the user can enter such words in Section 3.4, below.)

3.3 Text Editing with "Type, then Correct"

As the user finger-taps with TypeAnywhere, the text in the *compose area* often changes, as the decoder continually processes entered finger-taps in light of surrounding textual context. At times, the text in the *compose area* will not be what the user intends, but by providing more textual context, the resolved text might conform to the user's desires. Therefore, the user is encouraged to continue typing in TypeAnywhere, even when the currently decoded text is not (yet) what the user wants. However, in cases where further context does *not* resolve the decoded text to match the user's intentions, the user can edit the text; however, at this point, they will be well ahead of the error position. This situation can result in user uncertainty and editing inefficiencies that are not desirable in TypeAnywhere.

To address this issue—that is, letting users continue to type, which provides the decoder with valuable additional context, but also avoids costly error correction situations—we employ the text correction scheme from *Type, then Correct* [7, 71], which lets the user type a correction at the *end* of their current text stream and then apply it directly and retroactively to a previous error without

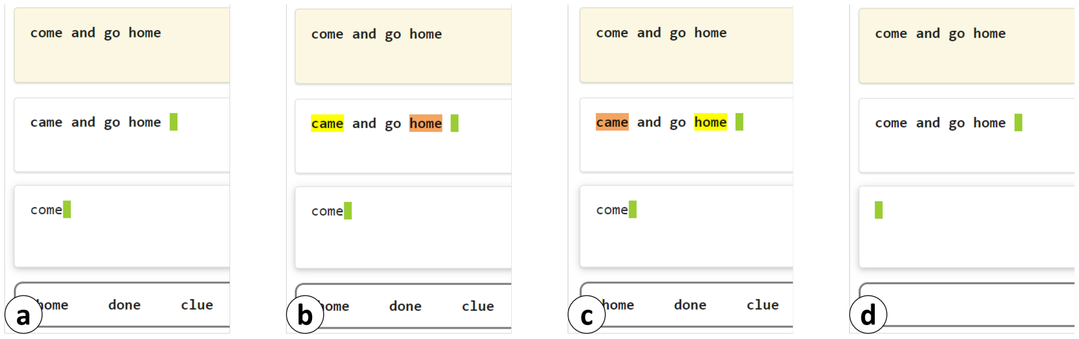

Figure 7: The correction interaction flow: (a) After committing the text containing the error word "came", the user types the correction "come" in the *compose area.* (b) The user then triggers the correction function. Possible errors are highlighted: both "came" and "home" have one finger-sequence edit distance with "come", hence the last word, "home", is highlighted in orange. (c) The user then employs the selection gesture to navigate between the candidates. (d) After the user presses the (virtual) spacebar, "came" is corrected to "come".

any cursor navigation. In the context of TypeAnywhere, if there are errors perhaps multiple words behind the current typing position, the user can commit all current *compose area* text to the *commit area*, and then type the corresponding correction in the *compose area*. The user can then tap the chord to trigger the correction function (Figure 4, left). The function compares the similarity of the correction word and the words in the *commit area* based on their finger-sequence edit distances, as described in the next paragraph. All words that are less than or equal to two edit-distance will be highlighted in yellow, with the shortest distance word being highlighted in orange. The user can then navigate through the possible errors and tap their virtual spacebar to commit the correction. To cancel correction mode, the user simply taps their virtual backspace. The whole correction interaction flow is demonstrated in Figure 7.

Edit distance refers to the minimum number of character edit operations (substitute, insert, delete) to change one string into another [30]. In TypeAnywhere, because the text is decoded based on finger-tap sequences, we compare the finger-sequence edit distance to implement our correction function. For example, in the standard finger-to-key mapping, the word "far" (left index - left pinky - left index) has the same finger sequence as the word "rat", and has a finger-sequence edit distance of one with the word "get" (left index - left middle - left index). Note that in our finger-sequence edit distance calculation, along with substitutions, insertions, and deletions, we also added transpositions, thereby handling cases when finger-taps are interchanged, such as typing the sequence "information" as "infromation").

3.4 Out of Vocabulary Words

As the provided word candidates are all in-dictionary words, typing out-of-vocabulary (OOV) words takes a little more effort. With TypeAnywhere, to enter an OOV word, the user needs to type the word character by character, and select the correct character in the candidate list each time. For example, to type the string "zmr" with the standard finger-to-key mapping, the user presses his left pinky first and selects "z"; then presses his right index and selects "m"; and finally presses the left index and selects "r".

4 THE NEURAL LANGUAGE DECODER

We now present the neural language decoder that converts a finger-tap sequence to text output in TypeAnywhere. Below, we explain the data processing and model training procedures, and report the model evaluation results.

4.1 Converting Text into Finger Sequences

Preparing the training data for the TypeAnywhere model is straightforward. As it only requires finger-tap sequences, we can reverse the decoding step to generate the training data: take the finger ID sequences corresponding to the character sequences in a corpus as the network input data, and the character sequence itself as the label. We assigned a unique ID to each finger: $2-5$ for pinky to index finger of the left hand, and $6-9$ for index to pinky finger of the right, and 1 for the two thumbs. For example, if we are creating the training data for the standard finger-to-key mapping, the phrase "sunny day" would be converted to "366661426". Because the model does not need finger motion information, generalizing the data to other finger-to-key mappings is easy, as we just need to change the finger ID for certain characters. For people who type the same letter with multiple fingers (such as typing the letter "u" with their index or middle fingers [13]), we can generate multiple training samples for each combination of the mapping. (Usually, people type one key with no more than two fingers [13], meaning the number of combinations will not be too large.)

4.2 BERT-Based Neural Language Model

The neural language model used for decoding finger-tap sequences in TypeAnywhere is based on the BERT model [10]. To fine-tune the model for our task, we used the finger-tap sequence as the input to make the BERT model output the corresponding characters [8]. Specifically, we added a linear classification layer on top of the BERT base model to generate the character for each input code. We

[8]We needed to finetune the BERT model for the finger mapping purpose as the language model itself could not be directly used for the mapping purpose, otherwise the system would query the model for every finger tap, which was very computational expensive

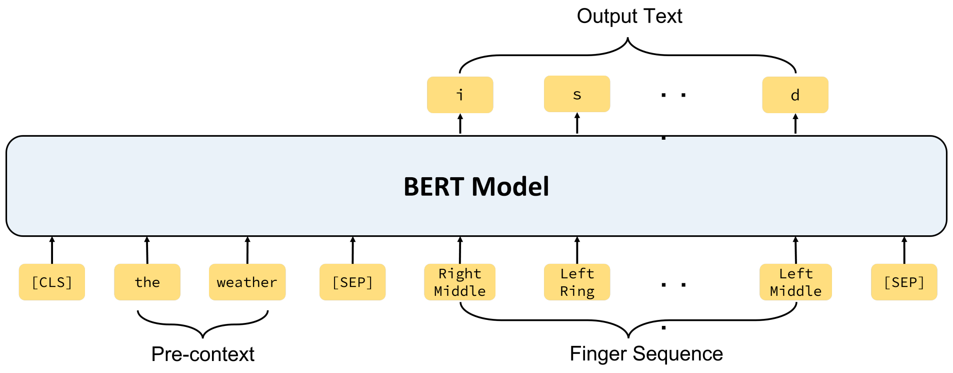

Figure 8: Input and output of the BERT decoder model.

decided to decode on the character level to provide real-time feedback for each tap action, even if the user has not yet finished typing the entire word. Another important motivation for decoding on the character level is that such a model can be used to type out-of-vocabulary (OOV) words not in the training data (see Section 3.4). Our implementation is based on the open-sourced transformers library [57]. Specifically, we used the *bert-base-uncased* pre-trained model. The model employed the multi-layer bidirectional transformer structure [50] with 12 transformer blocks, 769 hidden layers and 12 self-attention heads. It was pre-trained with a masked language model (i.e., predicting a hidden word with its surrounding context) and next-sentence prediction tasks on 3.3 billion words [10]. BERT was reported to achieve state-of-the-art performance on many language understanding tasks, and became the basis for many subsequent language models [27, 31, 62].

The flow of the model input and output is illustrated in Figure 8. Given a finger-tap sequence, our model will classify each tap in the sequence as a corresponding character. Because the committed text also provides useful contextual information for the decoder, we also include the text in the *commit area* (we call this *pre-text*) in the typing sequence. For example, if the user taps "*right middle – left ring – thumb – left index – right ring – right ring – left middle*" after committing the text "the weather", then the input to the model will be:

```
[CLS] the weather [SEP] 7 3 1 5 8 8 4 [SEP]
```

Here "[CLS]" and "[SEP]" are special tokens representing the start of the input and the separator of different parts of the input. The *pre-text* is then tokenized by the model at a sub-word level [59] while the finger IDs are all special tokens, and the decoded text will be "the weather is good". In the TypeAnywhere interface (Figure 5), the text in the *commit area* is treated as the pre-text.

4.3 Data Collection

We used the Yelp and Amazon review dataset generated by Zhang et al. [75] containing over 3.6 million reviews. For each review in the dataset, we cropped consecutive sequences containing 5 – 25 words, and removed all numbers and special characters so that sequences only contained the 26 Roman letters and the space character. For each sequence with n words, we then randomly selected m words as the *pre-text* ($0 \le m < n$), with the rest of the text converted to a finger sequence. For the test dataset, we used the Cornell Movie-Dialogs Corpus [8], which contained conversational exchanges from movie scripts. The purpose was to test the performance in a conversational setting so that the evaluation could reflect real usage scenarios for ubiquitous text entry. We randomly selected 10,000 phrases containing fewer than 25 words from the corpus. In all, we gained 44 million training samples and 10,000 test samples.

4.4 Training Process

We fine-tuned the BERT-based model with a linear classification layer (similar to a token classification task in NLP [10]). We calculated the cross-entropy loss of the output characters versus the ground truth text from the finger sequence. We used the AdamW optimizer [32] with a learning rate of 3e-5, and trained the model for one epoch. We applied weight clipping of $[-1.0, +1.0]$, and the batch size was set to 12 due to hardware limitations.

4.5 Model Evaluation

The best performance of the model on the test dataset (Cornell Movie-Dialogs Corpus[8]) reached 1.6% character error rate (CER) for an average sentence. To compare its performance with traditional language models, we also trained a 5-gram word-level language model using the KenLM library [23] on the same training dataset, and we excluded the punctuation and special characters other than Roman letters and space. We also applied Kneser-Ney smoothing [38]. The final binary model was 12 GB, which was large compared to the 440 MB BERT model. For evaluation purposes, we pruned the KenLM model by filtering out the words that were not in the test dataset (the movie corpus) to speed computation. The pruned version contains 27,000 1-grams, 0.8 million 2-grams, 2.7 million 3-grams, 4.2 million 4-grams and 4.6 million 5-grams.

To test the 5-gram model, for each word in a test phrase, we first found all possible candidate words based on the word's finger sequence. Candidate words were searched from the Open American National Corpus (OANC) word frequency dictionary [36], which contained over 22 million words of written American English. We then selected the best word candidate based on probabilities from the language model using the back-off strategy. Both neural and 5-gram decoders were given whole finger-tap sequences, and they both had access to all contextual information. The 5-gram model

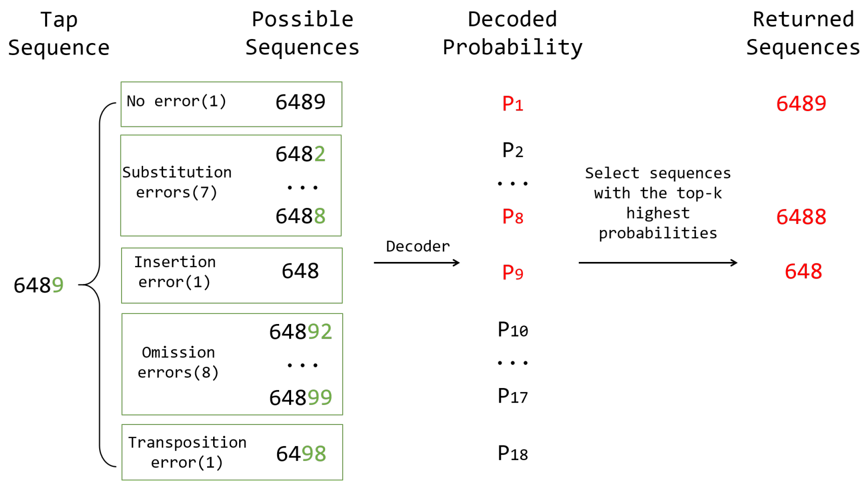
Tap Sequence
Possible Sequences
Decoded Probability
Returned Sequences
6489
No error(1) 6489
Substitution errors(7) 6482 ... 6488
Insertion error(1) 648
Omission errors(8) 64892 ... 64899
Transposition error(1) 6498
Decoder
P1
P2
...
P8
P9
P10
...
P17
P18
Select sequences with the top-k highest probabilities
6489
6488
648

Figure 9: Finger-tap sequence beam search example. The original sequence is “6489” with the last finger being ‘9’. Seventeen extra combinations are then added to the sequence list, and each possible sequence is processed by the decoder. The decoder outputs the probability for each sequence, and beam search then selects the top-*K* (here, *K*=3) sequences. If the user taps a new finger, for example, finger ‘4’, the new tap sequences would be “64894”, “64884”, and “6484”. Each of the three sequences then go through the procedure again; this time, there will be 3 × 18 = 54 total possible sequences, and beam search will select the top-*K* sequences from among all candidates.

achieved 5.3% CER. The results showed that the neural language model was both lighter and more accurate than a traditional *n*-gram model.

4.6 Candidate Generation and Auto-Correction with Beam Search

Of course, TypeAnywhere's neural decoder is not 100% accurate, and users may also type an incorrect finger-tap sequence for their intended word. We devised and implemented a few simple algorithms to complete TypeAnyWhere in order to address these challenges. To provide alternative word candidates for a finger-tap sequence, we searched the sequence in the OANC word frequency dictionary [36] (e.g., search the words whose character sequence is the same as the finger sequence), and collected the five most frequent of these words as alternate candidates. We did not use the BERT model for candidates, as it was designed to output character-level results, which did not work well for generating alternate word candidates.

We also needed to deal with situations in which the user typed the word incorrectly, such as missing a tap or interchanging the tap sequence. We developed an auto-correction feature based on beam search for the neural language model. Each time the user performed a tap, we searched through all possible hypotheses for different error types [56]. Besides decoding the detected finger-tap sequence, we also substituted the last finger ID with other finger IDs to detect substitution errors, omitted the last finger ID to detect insertion errors, interchanged the last two finger IDs for transposition errors, and appended one extra finger ID for omission errors. In total, 17 extra sequences were decoded alongside the original sequence.

As an example (Figure 9), consider when the user taps the finger sequence “6489”. (Recall that finger IDs ranged from ‘2’ to ‘9’ in the model, and we here excluded the thumb IDs of ‘1’ for spacebar in beam search.) The possible combinations then include seven sequence substitution errors (“648x” where ‘x’ is ‘2’ to ‘8’), one sequence for insertion errors (“648”), eight sequences for omission errors (“6489x” where ‘x’ is ‘2’ to ‘9’), and one sequence for transposition errors (“6498”).

If we maintain all possible sequences each time a new finger-tap is detected, the number of the sequences will explode as tapping continues. We therefore applied beam search to reduce the number of sequences, similar to the surface touch-decoding algorithm [41], where we only kept the top *K* sequences with the highest probabilities for each step. When a new tap was detected, we generated new sequences for error corrections based on the *K* sequences. In this project, *K* was set to 3, meaning for each tap, TypeAnywhere takes the top-3 sequences thus far. In this way, the model was able to detect most user errors caused by finger mis-taps. The best candidate of the current tap sequence was then displayed in the compose area of the interface in Figure 5.

5 EVALUATION OF TYPEANYWHERE

To evaluate the performance of TypeAnywhere in realistic settings, we conducted a longitudinal user study. Although TypeAnywhere utilized a QWERTY style of typing, we anticipated that it still would take time for users to acclimate to the finger-worn devices and the feeling of typing without a keyboard.

5.1 Participants

We recruited 10 participants (aged 23 – 28, all men[9]) for the study via word-of-mouth and online forums. The study was longitudinal over five days, resulting in 50 participant sessions and the ability

[9]We recognize the limitation of only having men in this study. We had all men because of the size of Tap strap, which fit men's hands better than women's.

to observe learning. We required that participants were able to perform desktop QWERTY typing without looking at the keyboard, and that they were consistent with their own personal finger-to-key mappings, i.e., they always used the same finger to type a given letter. Due to the COVID-19 pandemic, it was extremely difficult to recruit participants; we therefore included one participant ("P1") who was not consistent with his finger-to-key mappings, but was able to commit his time to the study. P1 could type on a physical keyboard without looking, but he used multiple fingers to press certain letters, such as "uio", for different words. Therefore, we can regard P1 as a novice learner for TypeAnywhere, as he needed to learn fixed finger-to-key mappings for the study.

Among the other nine participants, six typed with standard QWERTY finger-to-key mappings, while the other three typed the letter "b" with the right index finger and "c" with the left index finger. We therefore trained two decoding models for the study; the models' performance was similar for the offline data.

Our study was approved by our university's Institutional Review Board, and all participants provided consent to participate.

5.2 Apparatus

The typing device used was the Tap Strap mentioned in Section 3.1. We built a server written in Python to handle the HTTP communication between the web interface, the hardware, and the decoder. The model ran on a machine with a GTX 1080 graphics card. The testing software was from the TextTest++ open-source text entry evaluation project [72, 74].

5.3 Procedure

The study was a five-day longitudinal study. After a participant signed an online consent form, the experimenter sent the hardware apparatus to the participant's home, and instructed the participant on the use of the device and the web interface via Zoom videoconferencing.[10] Specifically, the typing interaction, the text editing gestures, and the correction function were demonstrated. Participants then started to practice typing with the device in the TextTest++ web application. The phrase sets for both practicing and testing were from the MacKenzie and Soukoreff phrase set [34] and the Enron email phrase set [52]. We randomized and divided the phrases so that they were different for practicing and testing in any given session, and also different from one session to the next.

Separate sessions occurred over five consecutive days. In each session, participants would practice typing with TypeAnywhere for 30 minutes on their desk surfaces, and then start testing. During practice, the web interface also displayed a QWERTY layout for the participants to refer to. For testing, 20 different phrases were used each day. Participants were instructed to type as fast and accurately as possible. The "enter" gesture was used to submit a completed phrase. During the evaluation, the participants were told that they could rest before typing each phrase, but needed to finish without stopping a phrase once they began typing it. We logged participants' typing using the transcription sequence model [72], which recorded all the intermediate text during the typing procedure to enable a thorough analysis of the input. After the evaluation, participants sent their log file to the experimenter.

[10]Due to the COVID-19 pandemic, we conducted the study remotely via Zoom. Our hardware apparatus was shipped to each participant and sanitized between each use.

On Day 1, the first day, participants performed the same typing test on their desktop keyboards to provide a baseline of their typing performance on a mechanical QWERTY keyboard. On Day 5, the last day, participants also performed a typing-on-their-lap evaluation with TypeAnywhere. The evaluation for the desktop keyboard session contained 30 phrases, while the evaluation for the typing-on-lap session contained 20 phrases, the same as the tabletop condition. At the end of the study on day 5, we conducted a debriefing session with participants, gathering their feedback from using TypeAnywhere. They also took a NASA Task Load Index (TLX) questionnaire on TypeAnywhere.

6 RESULTS

We gathered 1500 phrases in total ($5 \times 20 \times 10 = 1000$ phrases for the tabletop condition, $10 \times 30 = 300$ phrases for the desktop keyboard condition, $10 \times 20 = 200$ phrases for the typing-on-lap condition).

The average performance in each condition is shown in Table 1. We also report each participant's performance individually to reveal more insights. We refer to the three conditions, respectively, as *tabletop*, *on-lap*, and *keyboard* (i.e., typing on a mechanical desktop keyboard).

6.1 Text Entry Speed

Overall performance. Participants' average typing speed is shown in Table 1, with the corresponding learning curve shown in Figure 10. Their speed on Day 1 was 41.6 WPM, already comparable to mobile phone touch or gesture typing after five days of practice [40]. Their speeds increased each day with this trend still continuing by the end of the study. Compared to the mechanical desktop QWERTY keyboard (87.8 WPM), participants reached 70.6 WPM with TypeAnywhere on the last day. The 19.6% (17.2 WPM) difference between TypeAnyWhere and the desktop keyboard was smaller than the previously reported difference between touchscreen tabletop keyboards [15] and desktop keyboards.

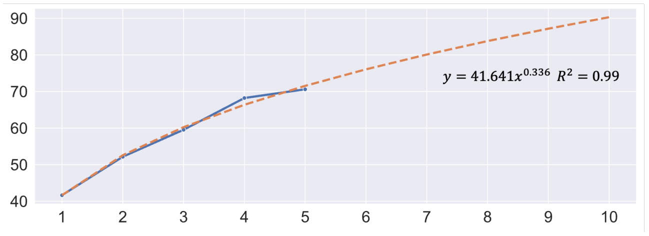

Figure 10: The average words per minute and its fitted learning curve for TypeAnywhere being used by participants finger-tapping on a desk. The extrapolated curve is limited to twice the number of sessions [35]

Typing-on-lap performance. The 43.9 WPM speed of the typing-on-lap condition also demonstrated the feasibility of TypeAnyWhere to be used in off-desktop conditions: As long as there is a surface, the user can perform QWERTY-style text entry with reasonable speed. However, as people's laps are softer than table surfaces,

Table 1: The average words per minute (WPM) and character error rate (CER) and standard deviations (*SD*) on each day for all participants with TypeAnywhere. Performance with the mechanical desktop QWERTY keyboard and in the typing-on-lap conditions are also shown.

	Day 1	Day 2	Day 3	Day 4	Day 5	Keyboard	On-lap
WPM	41.6 (16.3)	52.6 (19.5)	59.5 (21.4)	68.2 (18.9)	70.6 (19.9)	87.8 (27.8)	43.9 (17.2)
CER (%)	1.05 (7.34)	1.71 (3.75)	0.95 (2.67)	1.54 (4.06)	1.50 (3.42)	1.77 (4.31)	1.37 (3.52)

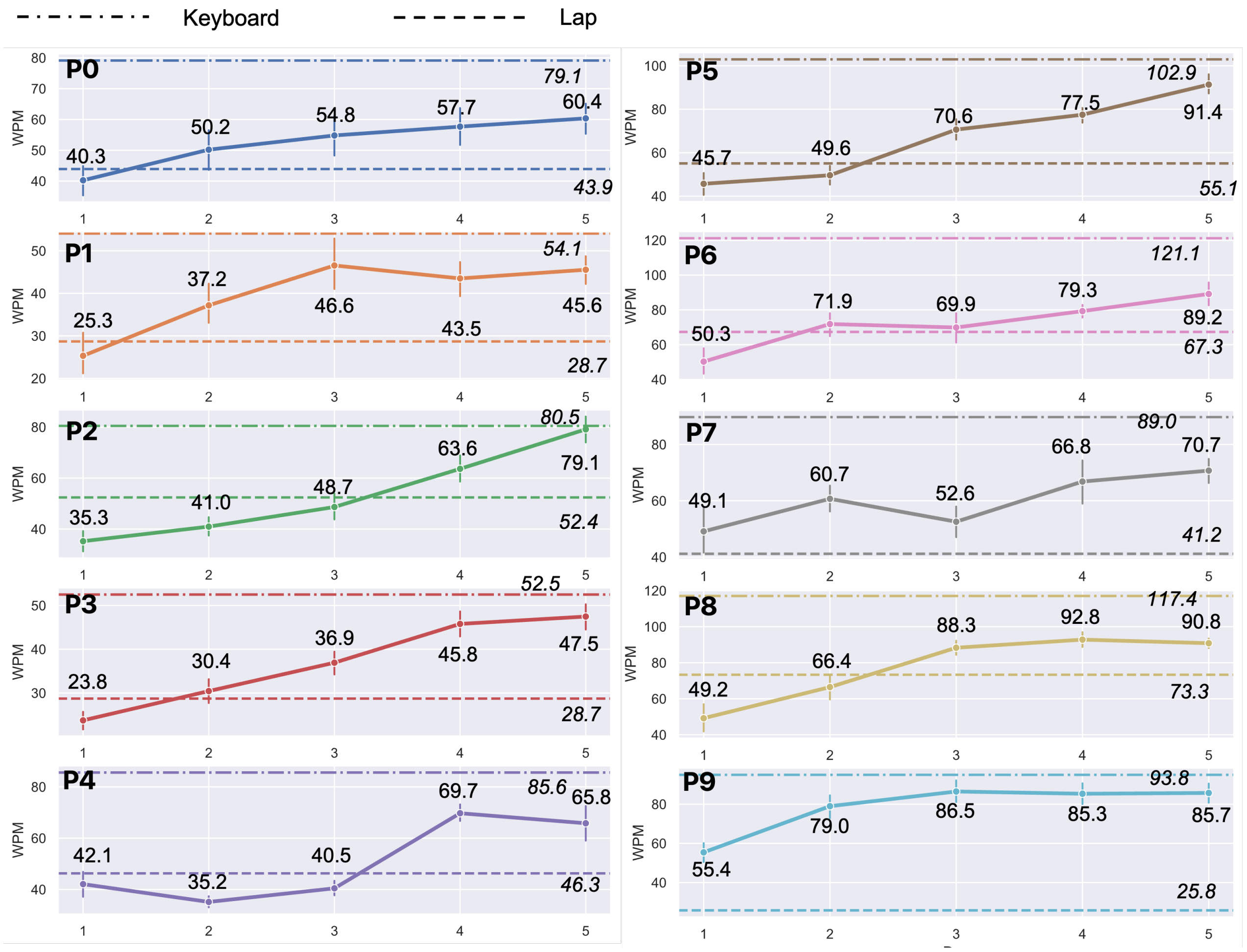

Figure 11: Words per minute (WPM) over five days for each participant. Higher is better. The upper horizontal line in each graph represents the desktop keyboard speed; the lower line in each graph represents the typing-on-lap speed. The vertical bar on each data point represents the 95% confidence interval.

TypeAnywhere's finger-tap detection performed less accurately on laps, causing some false-negative non-detections. That said, typing on laps is an odd but potentially useful scenario compared to typing on firm table surfaces, so some degradation of performance is to be expected.

Individuals' performance. Figure 11 shows each participant's performance across the five days. All participants increased their typing speeds over time, although there were some small drops due to the vagaries of the test phrases and human performance. With TypeAnywhere, participants reached between 73.7% and 98.3% (mean 80.4%) of their desktop QWERTY keyboard typing speeds on the last day. The fastest typist (P5) reached 91.4 WPM, while P2 even reached a typing speed with TypeAnywhere very close to his desktop keyboard typing speed on the fifth day, about 80 WPM. The novice learner P1 reached 84.2% of his desktop keyboard typing speed, although he had to learn both the interaction and the finger-to-key mappings in the study. The results suggest that people *can* learn TypeAnywhere in a relatively short period of time, even if they do not follow fixed finger-to-key mappings.

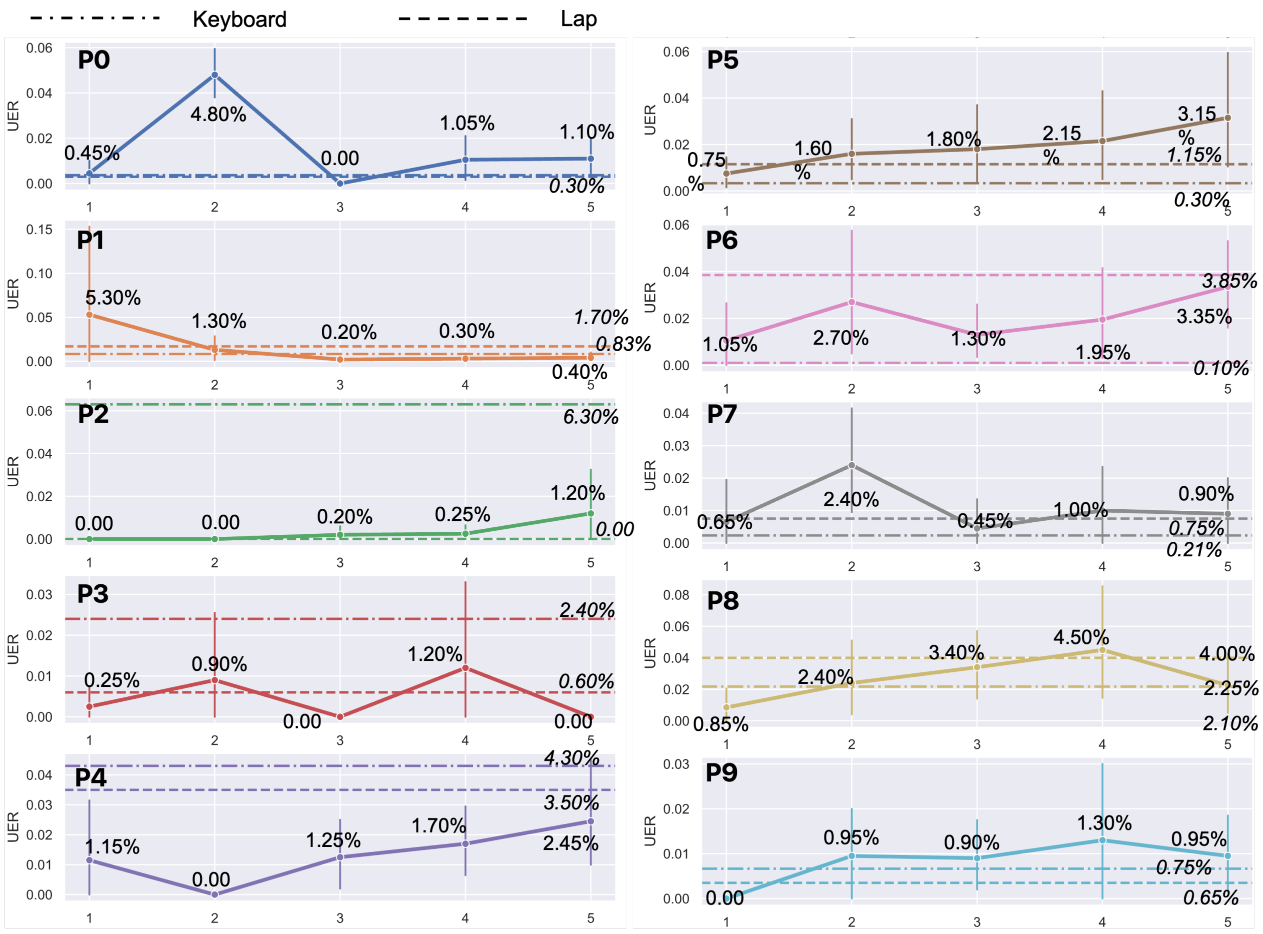

Figure 12: Character error rates (CER) over the five day study for each participant. Lower is better. Here we use Uncorrected Error Rate (UER) [47, 72] as CER. The upper horizontal lines in each graph represent the desktop keyboard CER; the lower lines represent the typing-on-lap CER. The vertical bars on each data point represent 95% confidence intervals.

6.2 Error Rate

Overall performance. Participants' average character error rates (CER) are shown in Table 1. These error rates were operationalized as *uncorrected* errors in the final transcribed text, as such errors trade-off against speed [74]; in contrast, *corrected* errors were not counted in CER, since any errors made but fixed during entry take time and are thus subsumed by speed. There is no obvious trend for the error rates, but the desktop keyboard condition had the biggest CER on average. From our log files, we observed that participants generally made more corrections (i.e., they pressed more backspaces) when typing with TypeAnywhere than in the desktop keyboard condition. However, since they left fewer errors in their final transcriptions, their resultant CER was lower.

Individual performance. Figure 12 shows each participant's error rate across the five days. The CER of all participants remained low for all five days. P2, P3, and P4 were even able to achieve lower error rates than the desktop keyboard condition for all sessions, despite the lack of physical keys for TypeAnywhere. Because of auto-correction, participants tended to leave few typos with TypeAnywhere.

Utilization of "Type, Then Correct". We logged the frequency of participants using the Type, Then Correct (TTC) [71] correction feature in the study, finding that only four participants (P0, P2, P4 and P8) used this interaction method for making corrections. Among the 1000 phrases entered, only 16 phrases were completed with this method. As the interaction was a new concept to the participants, it was not surprising that participants still heavily relied on backspace and word-delete functions for text correction. Although the TTC method can be cognitively demanding for new users, with more practice, we expect that participants might make greater use of TTC style corrections, since it is more efficient.

6.3 Error Behavior Analysis

Although error rates were small for each participant, we observed several common error types such as omitting characters and typing with the wrong fingers, which could be caused by the device itself (not recognizing the taps), or the user while trying to adapt to the new interface. We therefore performed a more detailed behavior analysis for errors made during the study.

From the text logs, we located 513 total typing errors made by all participants. We identified those typing errors by finding the mismatches between finger sequences, corresponding text sequences (the transcribed text), and the ground truth text sequences (the presented text). Among the 513 errors that were typed incorrectly by participants, 224 errors were auto-corrected, with 112 (50%) of those fixed to become the correct text, indicating the usefulness of the auto-correction algorithm.

Table 2: Count of different error types each day for all participants.

Error Type	Day 1	Day 2	Day 3	Day 4	Day 5	Total
Substitution	80	69	54	48	48	299
Omission	48	25	19	19	16	127
Insertion	20	10	18	8	7	63
Transposition	10	4	2	5	3	24

The distribution of errors is shown in Table 2. The most common error was typing with the wrong finger (i.e., 299 substitution errors[11]), followed by omitting a finger tap (i.e., 127 omission errors). Although the latter error type was mostly caused by the device not recognizing finger-taps, omissions accounted for less than 50% of wrong-finger errors, which were caused by the user.

The decrease in omission errors also indicated that the participants learned how to make a recognizable tap during practice. Although Tap Strap was reported to have over 98% tap recognition accuracy, not all taps were recognized well during the initial practice session. The Tap Strap company also has a video tutorial on how to perform a proper tap,[12] where it mentions, "*tap gently, and you don't need a lot of force to tap.*" In our study, it took some time for the participants to learn how to tap properly. Specifically, in the first practice session, some participants tended to move their whole hands instead of their fingers to "tap", which caused non-recognitions, as the straps sense the movement of fingers, not the whole hands. Another failure case was when our participants performed "roll-over" actions when typing [13] (*e.g.*, pressing one finger before another finger lifts; this action was common among fast typists). The roll-over finger usually exerted a light touch on the surface instead of a tap. Over the five days of learning, participants successfully adjusted their typing style to make sure that taps were recognized by the straps.

We also examined the error dynamics of P1, who was a novice learner of the QWERTY keyboard layout. Wrong-finger errors decreased from Day 1 to Day 5 (i.e., 11, 9, 3, 2, 1), indicating that P1 was adapting to the TypeAnywhere interaction gradually.

6.4 Subjective Feedback

The NASA TLX workload results are shown in Figure 13. The mental load and the overall effort were perceived as high, as participants had to explicitly think about the keyboard layout during typing. (By contrast, when typing on a physical desktop QWERTY keyboard,

[11]The error type definitions can be found in [56].
[12]https://www.youtube.com/watch?v=XCI7D3IkA6E

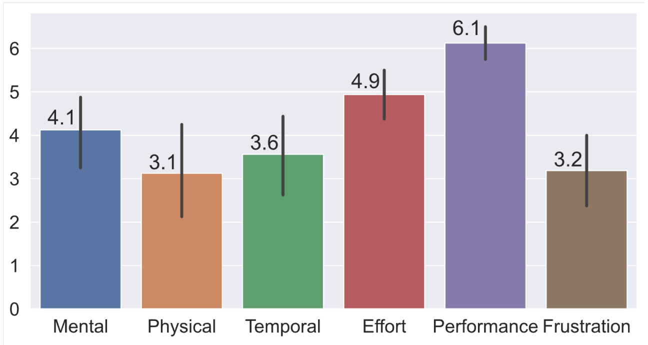

Figure 13: Average NASA TLX ratings of the participants (scale 1 - 7). For mental / physical / temporal / effort / frustration, high score means high task load; for performance, high score means participants are highly satisfied with their performance. The vertical bars on each data point represent 95% confidence intervals.

participants can rely upon muscle memory without thinking about which finger is striking each key [6].) The temporal taskload (*i.e.*, time pressure) was also perceived as a bit high, as our participants tried to type as fast and accurately as possible during our study. However, their perceived performance was rated as very successful, indicating that participants were quite satisfied with their typing performance by the end of the study.

We also collected feedback from participants during a debriefing session at the end of Day 5. Specifically, we asked how well they liked (or disliked) TypeAnywhere, and what they thought could be improved. Overall, participants were all enthusiastic about the system and thought that it could be practical in actual use. The most touted benefit of TypeAnywhere by participants was that they no longer needed to concern themselves with (x, y) locations when typing: *"I can just tap the finger without caring about whether the position was right or not"* (P3), which "*actually enabled me to spread my fingers a bit so that when I tap my pinky, I don need to worry about moving my ring finger together. And I do not need to keep my both hands together, which is nice.*" (P5). The other benefit was that the user could perform typing on any surface similar to typing on a desktop QWERTY keyboard, without the need to *"learn a new coding system"* (P2). Three participants also appreciated the simple gesture designs, such as the backspace gesture, commenting that the gestures were "*easy to perform and intuitive*" (P6).

Five participants mentioned that they needed to adjust their typing style to make the device recognize their taps. For example, P5 said, "*Initially I was imagining there was a keyboard and my fingers would follow the positions, but gradually I felt that I only need to listen to my fingers and not worry about the positions. I also learned how to tap to make the device work smoothly.*" P8 struggled a bit when he was using the device for the first two days, as he moved the whole hand rather than the finger to perform his taps. After he adjusted his typing style, his performance was much better: "*I'm used to the [physical] keyboard for a long time and I need a bit [of] mental training to adjust my taps.*"

As for aspects of TypeAnywhere that participants wanted to improve, all participants felt they needed to think about their finger-to-key mappings explicitly while typing, which increased their cognitive load. Our novice learner, P1, commented that, *"I used to type without even knowing what my finger-to-key mapping was. It just happened naturally. Now I need to have a keyboard map in my mind during typing."* Participants also complained that the Tap Strap hardware device failed to detect taps sometimes, especially in the typing-on-lap condition. Finally, as the text dynamically changed in the composition area during typing, participants mentioned feeling unsure when they were in the middle of a word, as the intermediate results were not always as they expected. This uncertainty added to participants' occasional confusion, so sometimes they would just delete a word and retype it.

7 DISCUSSION

Combining off-the-shelf wearable finger-tap sensors, the pre-trained deep learning language model BERT, and custom software to enable a novel interaction design, TypeAnywhere has achieved the highest reported performance among manual text input methods other than the desktop physical keyboard 1. Although TypeAnywhere does not have the benefit of tactile feedback from mechanical desktop QWERTY keyboards, the design of TypeAnywhere has the benefit of several factors: (1) Its QWERTY-style typing interaction provided for a relatively easy-to-learn system, enabling users to transfer their years of keyboard typing skills to a new system within a short period; (2) Unlike previous work [46, 61, 65, 76], which also required position information, only finger-tap sequences are required in TypeAnywhere, enabling us to utilize an existing large corpus as the training data without the need of collecting real typing data; (3) The neural language decoder took advantage of current advances in natural language processing by incorporating more context during the decoding process and achieving better decoding accuracy compared to a traditional n-gram model, for $n = 5$; (4) To improve the usability of TypeAnywhere, we also implemented auto-correction and alternative candidates based on beam search and Type, then Correct [71] correction interactions, both of which reduce the penalties incurred with making mistakes.

All participants were able to adapt their typing to the device without a real keyboard. In fact, on the first day, most participants were able to successfully transfer their typing style from the mechanical QWERTY keyboard, moving their fingers to imaginary key positions on a QWERTY keyboard layout. Participants also tried to intentionally make heavy taps in order to get them recognized by the device. However, after one or two sessions practicing with the device, participants learned how to tap their fingers in a relaxed manner without moving the finger very far to hit the intended imaginary "key". To be clear, this is not to say that participants only moved their fingers up and down; they still moved them to the relative key positions, but they did so with less effort and motion. Participants' rapid adaptation to minimize their necessary movements was actually surprising to the authors, as participants not only transferred their existing typing skills, but also learned a more efficient way to operate their fingers for the interaction.

Participants also learned our specific gestures quickly. They were able to perform delete and selection gestures in the first test session, and four participants went on to use the correction gesture quite frequently. On average, participants performed 11.7 (SD = 12.8) backspace gestures, and 17.7 (SD = 10.9) word-delete gestures for each section (20 phrases). This use of gestures indicated that both delete gestures were useful for text editing. For candidate selection gestures (select next/last), participants, on average, performed 5.4 (SD = 2.4) candidate selections over the five days of the study.

Another noticeable result was the performance degradation in the typing-on-lap condition. A reduction in performance was expected, as participants only performed on-lap typing on the last day, and the surface was softer and smaller compared to the tabletop. Participants also adjusted their sitting postures several times during the test and changed the part of the lap on which they tapped,. P3 observed a "lack of the feeling of taps as the laps and pants are soft." Due to the softness of the lap, it was also harder for the device to detect a "tap" action. Nevertheless, all participants were able to finish the on-the-lap typing and reached an average 43.9 WPM, which is still a considerable entry rate compared to most off-desktop text entry solutions [68]. The on-lap condition was included to demonstrate TypeAnywhere's ability to enable the user to type on most any surface, such as on a wall, a bicycle handlebar, or a user's leg or lap.

The era of ubiquitous computing is here, and we no longer only interact with computers in traditional desktop settings. TypeAnywhere enables users to type without a physical keyboard, while nonetheless resembling a QWERTY keyboard typing experience. It can be potentially used on-the-go with smart glasses or earbuds, with AR/VR devices, or as a unified input solution for multiple devices such as televisions, tablets, smartwatches, or mobile phones.

7.1 Limitations

This paper describes what we might call the first attempt at achieving the "type anywhere hypothesis," namely testing whether using neural decoding to translate finger-taps on an imaginary QWERTY keyboard can achieve efficient typing on (almost) any surface. The answer appears to be "yes" thus far, but this research project still has several limitations:

There is still room for the hardware devices we used to improve tap-detection accuracy. In our study, participants performed tapping with different gestures, even when using the same finger on the same key. For example, we observed that some participants preferred moving and tapping their whole hand instead of moving the finger, which added difficulty to tap detection.

We only conducted the study with ten participants, albeit longitudinally over five sessions. Although having multiple sessions helps to compensate for the small number of participants, our generalizability could be improved further by a larger study. Also, participants were still improving after five sessions; more sessions would illuminate their performance further. On the other hand, a five-day longitudinal study with five participants produces 50 study sessions, which for most text entry evaluations is a reasonable size. In fact, many text entry studies (e.g., [26, 39, 41, 42]) have had smaller participant numbers, and most were not longitudinal.

The phrase sets we used in the study were extracted from standard text entry test sets. The phrase sets did not include any out-of-vocabulary (OOV) words, which might not align well with actual usage scenarios for TypeAnywhere. It would be worthwhile to conduct composition tasks instead of transcription tasks to evaluate the interaction in a comprehensive manner.

We did not conduct a formal study of the default commerical text entry interaction of the Tap Strap, or any other chording keyboards. This is because the experience of chording keyboards is very different from that of typing, including on TypeAnywhere. Chording systems are unfamiliar to novice users and are often designed for single-hand use. In contrast, TypeAnywhere focuses on utilizing users' existing two-hand typing skills, where they can pick up and type without days or weeks of learning.

We also did not explicitly measure the accuracy of the tap actions or the editing gestures. As we show in section 6.3, the accuracy of the gestures relies not only on how well the device was, but also on how participants performed their taps. And throughout the study, all participants adjusted their typing style for better performance.

Finally, the BERT-based neural decoding implementation was not optimized. Possible optimizations include adding weights to different errors during the beam-search auto-correction procedure; outputting words instead of characters so that the model could also provide alternative candidates; condensing the model size so that it could incorporate larger context and run faster.

8 FUTURE WORK

Along with addressing the limitations mentioned above, we see several possible directions for future work. We could add comprehensive support for special characters, since currently, only letters and the space character are supported in TypeAnywhere. Numbers, punctuation, symbols, and modifier keys must also be included in a full solution capable of supporting everyday use. One possible solution could be assigning different symbols to different chords and users would perform a mode gesture to switch between typing and symbol-entry modes. Another possible solution could be query-style entry, such as for emojis: the user simply performs a chord to trigger the search mode, and then types the description of the symbol or emoji to search and enter it [70]. For example, typing a "q" in search mode could make a question mark ("?") immediately available for entry. We also could utilize gestures beyond just finger-tapping. TypeAnywhere's current design only involves tapping, whether by a single finger or chorded for editing operations. Motion gestures, such as swipe, pinch, snap, or open/close hand could be used for various purposes in future iterations. We also might imagine improving the form factor of the hardware device. The devices we used for the project were two finger-worn ring straps. In the future, we expect smartwatches and other wrist-worn devices, or a computer vision-based system, to detect finger taps to avoid the need to wear specialized devices on fingers for text entry. Even with alternate hardware, our finger-tap encoding scheme, neural language decoder, and interaction design all could be reused.

9 CONCLUSION

We have presented *TypeAnywhere*, a system that enables QWERTY-style typing on just about any surface as a general text entry solution for ubiquitous computing. TypeAnywhere applies a neural language decoder that accepts only finger-tap sequences as the input, which eases both the process of training and the generalization to different finger-to-key mappings. We trained the decoder with a large corpus and achieved 1.6% character error rate (CER) in an offline evaluation. To enable interactive use, we also designed auto-correction and *Type, then Correct* [71] style interaction to lower the risk and cost of making and correcting errors. Our longitudinal study showed that participants were able to learn TypeAnywhere with relative ease, reaching 70.6 WPM on average after five sessions or about 2.5 hours of practice, with the fastest person reaching 91.4 WPM. We hope that TypeAnywhere can serve as a first step towards a usable text entry solution for wearable and ubiquitous computing devices, and will inspire future research to design faster yet intuitive text entry methods for the ubiquitous computing era.

ACKNOWLEDGMENTS

Anonymized for review. We thank Moti Kdoshim from TapWithUs for providing device related assistance; Tianyu Gao for helping designing the language model training methods; Mingyuan Zhong for helping training neural network models; Google for providing TAP devices; and we thank all the participants. This work was supported in part by Baidu. Any opinions, findings, conclusions or recommendations expressed in our work are those of the authors and do not necessarily reflect those of any supporter.

REFERENCES

[1] Michelle Annett and Walter F. Bischof. 2013. Your Left Hand Can Do It Too! Investigating Intermanual, Symmetric Gesture Transfer on Touchscreens. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Paris, France) *(CHI '13)*. Association for Computing Machinery, New York, NY, USA, 1119–1128.

[2] Xiaojun Bi, Barton A. Smith, and Shumin Zhai. 2012. Multilingual Touchscreen Keyboard Design and Optimization. *Human–Computer Interaction* 27, 4 (2012), 352–382.

[3] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. arXiv:2005.14165 [cs.CL]

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

[5] Xiang 'Anthony' Chen, Tovi Grossman, and George Fitzmaurice. 2014. Swipeboard: A Text Entry Technique for Ultra-Small Interfaces That Supports Novice to Expert Transitions. In *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology* (Honolulu, Hawaii, USA) *(UIST '14)*. Association for Computing Machinery, New York, NY, USA, 615–620.

[6] Matthew J. C. Crump and Gordon D. Logan. 2010. Warning: This keyboard will deconstruct— The role of the keyboard in skilled typewriting. *Psychonomic Bulletin & Review* 17 (2010), 394–399.

[7] Wenzhe Cui, Suwen Zhu, Mingrui Ray Zhang, H. Andrew Schwartz, Jacob O. Wobbrock, and Xiaojun Bi. 2020. JustCorrect: Intelligent Post Hoc Text Correction Techniques on Smartphones. In *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology*. Association for Computing Machinery, New York, NY, USA, 487–499.

[8] Cristian Danescu-Niculescu-Mizil, Justin Cheng, Jon Kleinberg, and Lillian Lee. 2012. You Had Me at Hello: How Phrasing Affects Memorability. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, Jeju Island, Korea, 892–901.

[9] Paul A David. 1985. Clio and the Economics of QWERTY. *American Economic Review* 75, 2 (May 1985), 332–337. https://ideas.repec.org/a/aea/aecrev/v75y1985i2p332-37.html

[10] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv:1810.04805 [cs.CL]

[11] Vivek Dhakal, Anna Maria Feit, Per Ola Kristensson, and Antti Oulasvirta. 2018. *Observations on Typing from 136 Million Keystrokes*. Association for Computing Machinery, New York, NY, USA, 1–12.

[12] Jacqui Fashimpaur, Kenrick Kin, and Matt Longest. 2020. PinchType: Text Entry for Virtual and Augmented Reality Using Comfortable Thumb to Fingertip

Pinches. In *Extended Abstracts of the 2020 CHI Conference on Human Factors in Computing Systems* (Honolulu, HI, USA) *(CHI EA '20)*. Association for Computing Machinery, New York, NY, USA, 1–7.

[13] Anna Maria Feit, Daryl Weir, and Antti Oulasvirta. 2016. How We Type: Movement Strategies and Performance in Everyday Typing. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems* (San Jose, California, USA) *(CHI '16)*. Association for Computing Machinery, New York, NY, USA, 4262–4273.

[14] Leah Findlater and Jacob Wobbrock. 2012. Personalized Input: Improving Ten-Finger Touchscreen Typing through Automatic Adaptation. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Austin, Texas, USA) *(CHI '12)*. Association for Computing Machinery, New York, NY, USA, 815–824.

[15] Leah Findlater, Jacob O. Wobbrock, and Daniel Wigdor. 2011. Typing on Flat Glass: Examining Ten-Finger Expert Typing Patterns on Touch Surfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Vancouver, BC, Canada) *(CHI '11)*. Association for Computing Machinery, New York, NY, USA, 2453–2462.

[16] Mikael Goldstein, Robert Book, Gunilla Alsiö, and Silvia Tessa. 1999. Non-Keyboard QWERTY Touch Typing: A Portable Input Interface for the Mobile User. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Pittsburgh, Pennsylvania, USA) *(CHI '99)*. Association for Computing Machinery, New York, NY, USA, 32–39.

[17] Jun Gong, Zheer Xu, Qifan Guo, Teddy Seyed, Xiang 'Anthony' Chen, Xiaojun Bi, and Xing-Dong Yang. 2018. WrisText: One-Handed Text Entry on Smartwatch Using Wrist Gestures. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems* (Montreal QC, Canada) *(CHI '18)*. Association for Computing Machinery, New York, NY, USA, 1–14.

[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* (San Francisco, California, USA) *(IUI '02)*. Association for Computing Machinery, New York, NY, USA, 194–195.

[19] Mitchell Gordon, Tom Ouyang, and Shumin Zhai. 2016. WatchWriter: Tap and Gesture Typing on a Smartwatch Miniature Keyboard with Statistical Decoding. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems* (San Jose, California, USA) *(CHI '16)*. Association for Computing Machinery, New York, NY, USA, 3817–3821.

[20] Jonathan T. Grudin. 1983. *Error Patterns in Novice and Skilled Transcription Typing.* Springer New York, New York, NY, 121–143.

[21] Yizheng Gu, Chun Yu, Zhipeng Li, Zhaoheng Li, Xiaoying Wei, and Yuanchun Shi. 2021. QwertyRing: Text Entry on Physical Surfaces Using a Ring. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 2 (March 2021), 1.

[22] 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* (Hong Kong, China) *(IUI '10)*. Association for Computing Machinery, New York, NY, USA, 111–118.

[23] Kenneth Heafield. 2011. KenLM: Faster and Smaller Language Model Queries. In *Proceedings of the Sixth Workshop on Statistical Machine Translation.* Association for Computational Linguistics, Edinburgh, Scotland, 187–197.

[24] Jonggi Hong, Seongkook Heo, Poika Isokoski, and Geehyuk Lee. 2015. SplitBoard: A Simple Split Soft Keyboard for Wristwatch-Sized Touch Screens. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems* (Seoul, Republic of Korea) *(CHI '15)*. Association for Computing Machinery, New York, NY, USA, 1233–1236.

[25] Yoon Sang Kim, Byung Seok Soh, and Sang-Goog Lee. 2005. A new wearable input device: SCURRY. *IEEE Transactions on Industrial Electronics* 52, 6 (2005), 1490–1499. https://doi.org/10.1109/TIE.2005.858736

[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* (Santa Fe, NM, USA) *(UIST '04)*. Association for Computing Machinery, New York, NY, USA, 43–52.

[27] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2019. ALBERT: A Lite BERT for Self-supervised Learning of Language Representations. *CoRR* abs/1909.11942 (2019), 1. arXiv:1909.11942

[28] Minkyung Lee and Woontack Woo. 2003. ARKB: 3D vision-based Augmented Reality Keyboard. In *Online Proceeding of the 13th International Conference on Artificial Reality and Telexistence.* IEEE, Keio University, Tokyo, Japan, 1.

[29] Luis A. Leiva, Alireza Sahami, Alejandro Catala, Niels Henze, and Albrecht Schmidt. 2015. Text Entry on Tiny QWERTY Soft Keyboards. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems* (Seoul, Republic of Korea) *(CHI '15)*. Association for Computing Machinery, New York, NY, USA, 669–678.

[30] V. I. Levenshtein. 1965. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet physics. Doklady* 10 (1965), 707–710.

[31] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A Robustly Optimized BERT Pretraining Approach. *CoRR* abs/1907.11692 (2019), 1. arXiv:1907.11692

[32] Ilya Loshchilov and Frank Hutter. 2019. Decoupled Weight Decay Regularization. arXiv:1711.05101 [cs.LG]

[33] Kent Lyons, Thad Starner, Daniel Plaisted, James Fusia, Amanda Lyons, Aaron Drew, and E. W. Looney. 2004. Twiddler Typing: One-Handed Chording Text Entry for Mobile Phones. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Vienna, Austria) *(CHI '04)*. Association for Computing Machinery, New York, NY, USA, 671–678.

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

[35] 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* (Pittsburgh, Pennsylvania, USA) *(CHI '99)*. Association for Computing Machinery, New York, NY, USA, 25–31.

[36] Catherine Macleod, Nancy Ide, and Ralph Grishman. 2000. The American National Corpus: A Standardized Resource for American English. In *Proceedings of the Second International Conference on Language Resources and Evaluation (LREC'00)*. European Language Resources Association (ELRA), Athens, Greece, 1.

[37] Aarthi Easwara Moorthy and Kim-Phuong L. Vu. 2015. Privacy Concerns for Use of Voice Activated Personal Assistant in the Public Space. *International Journal of Human–Computer Interaction* 31, 4 (2015), 307–335.

[38] H. Ney, U. Essen, and Reinhard Kneser. 1994. On structuring probabilistic dependences in stochastic language modelling. *Comput. Speech Lang.* 8 (1994), 1–38.

[39] Stephen Oney, Chris Harrison, Amy Ogan, and Jason Wiese. 2013. ZoomBoard: A Diminutive Qwerty Soft Keyboard Using Iterative Zooming for Ultra-Small Devices. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Paris, France) *(CHI '13)*. Association for Computing Machinery, New York, NY, USA, 2799–2802.

[40] Shyam Reyal, Shumin Zhai, and Per Ola Kristensson. 2015. Performance and User Experience of Touchscreen and Gesture Keyboards in a Lab Setting and in the Wild. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems* (Seoul, Republic of Korea) *(CHI '15)*. Association for Computing Machinery, New York, NY, USA, 679–688.

[41] Mark Richardson, Matt Durasoff, and Robert Wang. 2020. Decoding Surface Touch Typing from Hand-Tracking. In *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology* (Virtual Event, USA) *(UIST '20)*. Association for Computing Machinery, New York, NY, USA, 686–696.

[42] Helena Roeber, John Bacus, and Carlo Tomasi. 2003. Typing in Thin Air: The Canesta Projection Keyboard - a New Method of Interaction with Electronic Devices. In *CHI '03 Extended Abstracts on Human Factors in Computing Systems* (Ft. Lauderdale, Florida, USA) *(CHI EA '03)*. Association for Computing Machinery, New York, NY, USA, 712–713.

[43] T. Salthouse. 1984. Effects of age and skill in typing. *Journal of experimental psychology. General* 113 3 (1984), 345–71.

[44] T. Salthouse. 1986. Perceptual, cognitive, and motoric aspects of transcription typing. *Psychological bulletin* 99 3 (1986), 303–19.

[45] L. H. Shaffer. 1978. Timing in the Motor Programming of Typing. *Quarterly Journal of Experimental Psychology* 30 (1978), 333 – 345.

[46] Weinan Shi, Chun Yu, Xin Yi, Zhen Li, and Yuanchun Shi. 2018. TOAST: Ten-Finger Eyes-Free Typing on Touchable Surfaces. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 2, 1, Article 33 (March 2018), 23 pages.

[47] R. William Soukoreff and I. Scott MacKenzie. 2003. Metrics for Text Entry Research: An Evaluation of MSD and KSPC, and a New Unified Error Metric. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Ft. Lauderdale, Florida, USA) *(CHI '03)*. Association for Computing Machinery, New York, NY, USA, 113–120.

[48] Srinath Sridhar, Anna Maria Feit, Christian Theobalt, and Antti Oulasvirta. 2015. Investigating the Dexterity of Multi-Finger Input for Mid-Air Text Entry. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems* (Seoul, Republic of Korea) *(CHI '15)*. Association for Computing Machinery, New York, NY, USA, 3643–3652.

[49] Ryo Takahashi, Masaaki Fukumoto, Changyo Han, Takuya Sasatani, Yoshiaki Narusue, and Yoshihiro Kawahara. 2020. TelemetRing: A Batteryless and Wireless Ring-Shaped Keyboard Using Passive Inductive Telemetry. In *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology* (Virtual Event, USA) *(UIST '20)*. Association for Computing Machinery, New York, NY, USA, 1161–1168.

[50] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention Is All You Need. arXiv:1706.03762 [cs.CL]

[51] 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.* Association for Computing Machinery, New York, NY, USA, 1–12.

[52] Keith Vertanen and Per Ola Kristensson. 2011. A Versatile Dataset for Text Entry Evaluations Based on Genuine Mobile Emails. In *Proceedings of the 13th International Conference on Human Computer Interaction with Mobile Devices and Services* (Stockholm, Sweden) *(MobileHCI '11)*. Association for Computing Machinery, New York, NY, USA, 295–298.

[53] Keith Vertanen, Haythem Memmi, Justin Emge, Shyam Reyal, and Per Ola Kristensson. 2015. VelociTap: Investigating Fast Mobile Text Entry Using Sentence-Based Decoding of Touchscreen Keyboard Input. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems* (Seoul, Republic of Korea) *(CHI '15)*. Association for Computing Machinery, New York, NY, USA, 659–668.

[54] Mark Weiser. 1999. The Computer for the 21<sup>st</sup> Century. *SIGMOBILE Mob. Comput. Commun. Rev.* 3, 3 (July 1999), 3–11.

[55] Jacob Wobbrock, Brad Myers, and Brandon Rothrock. 2006. Few-Key Text Entry Revisited: Mnemonic Gestures on Four Keys. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Montréal, Québec, Canada) *(CHI '06)*. Association for Computing Machinery, New York, NY, USA, 489–492.

[56] Jacob O. Wobbrock and Brad A. Myers. 2006. Analyzing the Input Stream for Character- Level Errors in Unconstrained Text Entry Evaluations. *ACM Trans. Comput.-Hum. Interact.* 13, 4 (Dec. 2006), 458–489.

[57] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2019. HuggingFace's Transformers: State-of-the-art Natural Language Processing. *ArXiv* abs/1910.03771 (2019), 1.

[58] Pui Chung Wong, Kening Zhu, and Hongbo Fu. 2018. *FingerT9: Leveraging Thumb-to-Finger Interaction for Same-Side-Hand Text Entry on Smartwatches.* Association for Computing Machinery, New York, NY, USA, 1–10.

[59] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. 2016. Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. arXiv:1609.08144 [cs.CL]

[60] Zheer Xu, Weihao Chen, Dongyang Zhao, Jiehui Luo, Te-Yen Wu, Jun Gong, Sicheng Yin, Jialun Zhai, and Xing-Dong Yang. 2020. BiTipText: Bimanual Eyes-Free Text Entry on a Fingertip Keyboard. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems* (Honolulu, HI, USA) *(CHI '20)*. Association for Computing Machinery, New York, NY, USA, 1–13.

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

[62] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. 2019. XLNet: Generalized Autoregressive Pretraining for Language Understanding. In *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (Eds.), Vol. 32. Curran Associates, Inc., ., 1.

[63] Xin Yi, Chen Wang, Xiaojun Bi, and Yuanchun Shi. 2020. PalmBoard: Leveraging Implicit Touch Pressure in Statistical Decoding for Indirect Text Entry. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems* (Honolulu, HI, USA) *(CHI '20)*. Association for Computing Machinery, New York, NY, USA, 1–13.

[64] Xin Yi, Chun Yu, Weijie Xu, Xiaojun Bi, and Yuanchun Shi. 2017. COMPASS: Rotational Keyboard on Non-Touch Smartwatches. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems* (Denver, Colorado, USA) *(CHI '17)*. Association for Computing Machinery, New York, NY, USA, 705–715.

[65] Xin Yi, Chun Yu, Mingrui Zhang, Sida Gao, Ke Sun, and Yuanchun Shi. 2015. ATK: Enabling Ten-Finger Freehand Typing in Air Based on 3D Hand Tracking Data. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology* (Charlotte, NC, USA) *(UIST '15)*. Association for Computing Machinery, New York, NY, USA, 539–548.

[66] Chun Yu, Ke Sun, Mingyuan Zhong, Xincheng Li, Peijun Zhao, and Yuanchun Shi. 2016. One-Dimensional Handwriting: Inputting Letters and Words on Smart Glasses. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems* (San Jose, California, USA) *(CHI '16)*. Association for Computing Machinery, New York, NY, USA, 71–82.

[67] Shumin Zhai, Per-Ola Kristensson, and Barton A. Smith. 2004. In search of effective text input interfaces for off the desktop computing. *Interacting with Computers* 17, 3 (02 2004), 229–250.

[68] Shumin Zhai, Per-Ola Kristensson, and Barton A. Smith. 2004. In search of effective text input interfaces for off the desktop computing. *Interacting with Computers* 17, 3 (02 2004), 229–250.

[69] Shumin Zhai, Per-Ola Kristensson, and Barton A. Smith. 2005. In search of effective text input interfaces for off the desktop computing. *Interacting with Computers* 17, 3 (2005), 229–250. Special Theme - Papers from Members of the Editorial Boards.

[70] Mingrui Ray Zhang, Ruolin Wang, Xuhai Xu, Qisheng Li, Ather Sharif, and Jacob O. Wobbrock. 2021. Voicemoji: Emoji Entry Using Voice for Visually Impaired People. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. Association for Computing Machinery, New York, NY, USA, Article 37, 18 pages.

[71] Mingrui Ray Zhang, He Wen, and Jacob O. Wobbrock. 2019. Type, Then Correct: Intelligent Text Correction Techniques for Mobile Text Entry Using Neural Networks. In *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology* (New Orleans, LA, USA) *(UIST '19)*. Association for Computing Machinery, New York, NY, USA, 843–855.

[72] Mingrui Ray Zhang and Jacob O. Wobbrock. 2019. Beyond the Input Stream: Making Text Entry Evaluations More Flexible with Transcription Sequences. In *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology* (New Orleans, LA, USA) *(UIST '19)*. Association for Computing Machinery, New York, NY, USA, 831–842.

[73] 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* (Yokohama, Japan) *(CHI '21)*. Association for Computing Machinery, New York, NY, USA, 32–39.

[74] Mingrui Ray Zhang, Shumin Zhai, and Jacob O. Wobbrock. 2019. Text Entry Throughput: Towards Unifying Speed and Accuracy in a Single Performance Metric. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems* (Glasgow, Scotland Uk) *(CHI '19)*. Association for Computing Machinery, New York, NY, USA, 1–13.

[75] 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* (Montreal, Canada) *(NIPS'15)*. MIT Press, Cambridge, MA, USA, 649–657.

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