Intelligent Text Input Methods and Metrics

Ph.D. Thesis Proposal

Mingrui "Ray" Zhang
The Information School
University of Washington

mingrui@uw.edu
https://drustz.com/

March 17, 2021

Supervisory Committee:

Jacob O. Wobbrock (Chair), University of Washington
Alexis Hiniker, University of Washington
Leah Findlater, University of Washington
Shumin Zhai, Google
James Fogarty (GSR), University of Washington

Abstract

Text input serves as a fundamental interactive part of almost any human-computer system, enabling expression, communication, and information capture. Since the cuneiform was invented as a writing system back in 3500 BCE, text as the medium of information has been created, recorded and communicated throughout the human civilization. The way we enter text significantly affects the efficiency we communicate with the device. Traditional desktop settings employ hardware keyboards for text input, which worked well for skilled typists. However, the devices in the current era are becoming mobile and ubiquitous, and have various form factors: from watch-sized wearable devices to handheld smartphones and wall-size large displays, most of which do not offer a physical keyboard. As a result, we need new text input interactions to communicate with those new devices. On the other hand, devices are becoming "intelligent", with powerful hardware processors to run advanced algorithms. Text input can take advantage of these capabilities but only if new algorithms and interaction techniques are developed.

To address the need, my thesis aims to develop intelligent text input systems utilizing the state-of-the-art machine learning techniques, in order to support a comprehensive spectrum of text interactions, including entry, editing, and entry of special symbols such as emojis. To encourage fast typing and unleash the power of the auto-correction, I design *PhraseFlow* which decodes the keystrokes on phrase-level and uses future input sequences to correct previous text. To improve the text editing process on touch-screen devices, I design *Gedit*, a set of on-keyboard gestures for mobile text editing. I also invent a novel concept, *Type, Then Correct*, together with three accompanying interaction techniques to improve the text correction on touch screens.

People are not only communicating in text: they use non-textual symbols such as emojis to enrich the expression. Another part of my work includes understanding how people use lexical and semantic emoji suggestion systems, and designing a voice-based interaction *Voicemoji* to facilitate blind and low vision users in entering and exploring emojis.

As advanced algorithms enabled more intelligent text input interactions, we need new metrics and evaluation methods to quantify their performance. I present a method-independent model, *transcription sequences*, for intelligent text input evaluations. Furthermore, to combine the speed and accuracy into a unified performance metric, I derive the *text entry throughput* based on the information theory, which is proved the most stable metric across various speed-accuracy conditions.

Based on my prior work, I propose three text input interactions for daily typing in the ubiquitous computing environment: 1) *TypeAnywhere*: a QWERTY-based text input interaction that detects finger taps with a wearable device and decodes the taps into text. 2) *TypeAnywhere One*: a single-hand extension for *TypeAnywhere*, and a friendly version for situational impairments when one hand is occupied. 3) *AnchorKeyboard*: a non-visual QWERTY keyboard for large touch screens. The user puts all fingers on the screen and only lift one finger when touching a key, making the typing finger identifiable with other anchor fingers. My thesis statement is:

> *Artificial intelligence can enable and improve advanced text input interactions, including the entering and editing of text, and entering of emojis; intelligent text interactions, in turn, warrant new text entry metrics for their evaluation.*

Contents

Chapter 1

Introduction

Text, as the medium of information, is an essential part of the human history: we think, talk and write through text. Text input, the interaction of entering text to devices, thus serves as a fundamental part of most human-computer systems. Since the commercial successful "Sholes–Glidden Type Writer" was produced in 1874 [67], tons of research on text input has been conducted, including improving the keyboard layout [7, 15, 86], proposing alternative hardware for other languages [63], and modeling the human performance in text entry tasks [11, 18, 84]. For traditional desktop settings, the physical QWERTY keyboard works well, with expert typists reaching 100 or more words pre minute (WPM) using multiple fingers [106]. However, as a result of the technology advancement, the devices today evolve rapidly into various form-factors, and the physical keyboard is no longer an optimal text entry solution for most of them. For example, wearable devices such as watches and earphones has small physical input spaces; smartphones and tablets employ touch screens rather than physical buttons as the primary interaction modality. Although those devices still largely employ the traditional keyboard paradigm in their text entry systems, such as using cursor for position control and manipulating text at character level, there are definitely new opportunities to design text input interactions better suited to the new paradigms.

The HCI community has focused on improving the text input interaction beyond the desktop-settings for a long time. For example, designing gestures and sensors to support text input on small-size devices [21, 42, 113, 114], optimizing and personalizing keyboard layout for touch screen interfaces [16, 86, 119], and leveraging multi-modal feedback for accessible text entry interactions [8, 89]. However, most of them exhibit three limitations: 1) they emphasize too much on inventing "novel interactions", instead of making the interaction practical. Novel and fancy interactions do not equal to good interactions, and interactions that are theoretically optimal do not necessarily perform well in real settings. For example, complex typing gestures can be tiring and hard to perform, and alternative keyboard layouts usually have a steep learning curve; 2) they only consider the text entering part, ignoring other parts such as text editing, and non-textual symbol entering, which are almost as important as entering text. 3) they focus on only the literal-level input, such as deterministic character-level input, or word-level input backed by word frequencies. In this way, they are not able to understand the meaning of users' intention, which operates on the semantic level. The end goal of text entry interactions is to improve the communication between the human

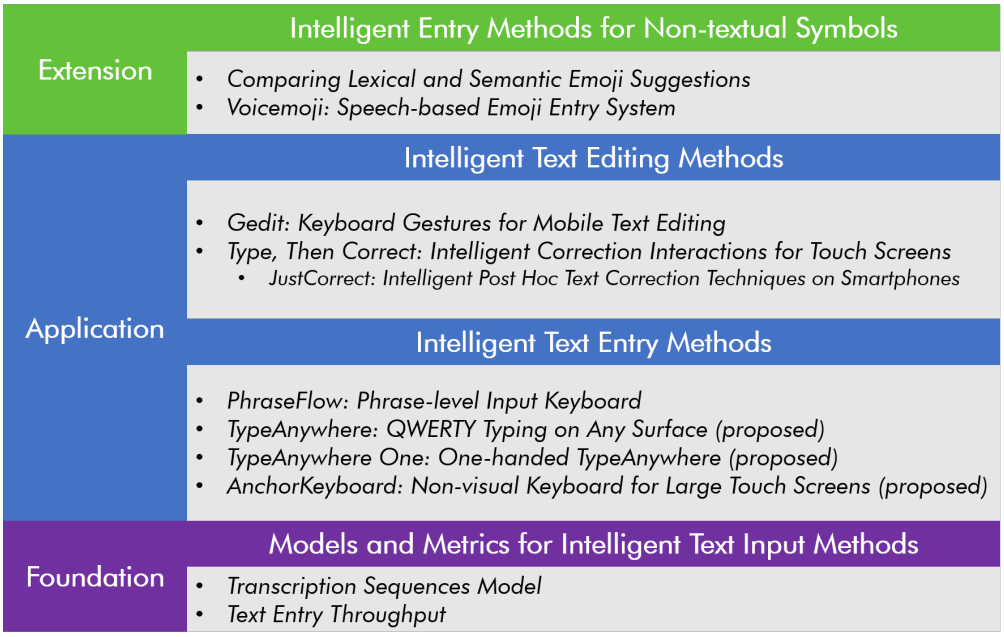

Figure 1.1: An overview of my research. Over the past three years, I have done foundational work on deriving text input models and metrics, inventing applications for intelligent text entry and editing, as well as designing intelligent entry methods for non-textual symbols

and the machine, rather than to solely improve the words-per-minute entry speed.

To address the limitations, one promising approach is to combine the power of artificial intelligence (AI). The advancement of machine learning (ML) algorithms, especially deep learning algorithms, has enabled the machine to achieve and even beat human performances on many understanding tasks that were impossible before, such as visual object classification [48] and voice recognition [3]. The natural language processing (NLP) community has also benefited from such advancement, making language models able to incorporate larger context and achieve high performance on tasks including summary, question-answering and conversation [17, 79, 120]. Those algorithms can potentially empower the text input methods to understand the user intentions, and to provide interactions that are simple and practical without introducing many barriers of learning. On the other hand, most ML algorithms are designed without human-in-the-loop, while text entry is a highly interactive procedure with the corporation of human and machine. There is for sure a gap between ML algorithms and text input interactions.

The goal of my dissertation work is to fill in the gap — designing and evaluating novel text entry interactions powered by artificial intelligence. Because of the limitation of traditional metrics, I first invented a method-independent model, *T-seq* [127], and a performance-level metric *throughput* [130], to evaluate intelligent text input methods. I then designed and evaluated a novel keyboard, *PhraseFlow* [129], that decodes the input sequences on the phrase-level. To improve the text editing experience on touch screens, I implemented *Gedit* [128], a set of on-keyboard gestures for convenient text editing, and *Type, Then Text (TTC)* [126], the concept of typing corrections first and applying it to the errors without the need of cursor placement. TTC is further extended by the correction interaction *JustCorrect* [25]. My work also focuses on non-textual symbol entry, *i.e.*

emojis, to support text input as a holistic information input system: I conducted empirical studies to understand the effect of lexical and semantic emoji suggestions on online conversations [124], and invented a speech-based emoji input system *VoicEmoji* for blind and low vision users [125].

Building upon my previous work, I would like to propose three potential projects that further deepen the combination of artificial intelligence and text input systems, deriving practical input systems for the ubiquitous computing environment:

- *TypeAnywhere*, an input interaction enabling the user to perform QWERTY-like keyboard typing on any surface with two wearable devices. *TypeAnywhere* utilizes the finger taps detected by the devices and decodes the tap sequences into text with neural network language models. There is no predefined finger-to-key mappings in *TypeAnywhere*, and the user can leverage their touch-typing skills on the physical keyboard, which lowers the bar of adoption. *TypeAnywhere* will also employ the phrase-level decoding and the TTC-style correction to improve the system usability.
- *TypeAnywhere One*, a single-hand version of *TypeAnywhere*. *TypeAnywhere* requires the user to use both hands to type, which is not feasible for motor-impaired users or when one hand is occupied. *TypeAnywhere One* has a different design of the finger-to-key mapping with *TypeAnywhere*, as it only requires one hand to type the characters on the whole keyboard. It utilizes a "mirrored" version of the layout when typing the characters on the other half of the keyboard. For example, if one types 'A' with left pinky, then one could type it with the right pinky as well. I will explore efficient candidate selection and disambiguation interactions for this project.
- *AnchorKeyboard*, a non-visual QWERTY keyboard for large touch screens. The keyboard does not have a fixed layout on the screen, rather it regards all on-screen touches as "anchors", and decodes the tapped finger point according to the anchor positions. Hence the user need to place all the fingers on screen and only tap one finger when typing. *AnchorKeyboard* could help blind and low vision users to type on large touch screens, as well as sighted users for eyes-free text entry.

The overall research diagram, including my proposed work, is illustrated in Figure 1.1. The statement of my dissertation is:

> *Artificial intelligence can enable and improve advanced text input interactions, including the entering and editing of text, and entering of emojis; intelligent text interactions, in turn, warrant new text entry metrics for their evaluation.*

The following Chapter 2 describes the background and related work. Remaining chapters present my research that is illustrated in Figure 1.1: in Chapter 3, I present my work on models and metrics for intelligent text entry systems. In Chapter 4, I introduce my work on improving text entry and editing process with decoders and interactions powered by machine learning. In Chapter 5, I extend my work on intelligent emoji entry. In Chapter 6, the proposed work for ubiquitous intelligent text entry interactions is presented. Chapter 7 concludes the proposal with anticipated contributions and a schedule.

Chapter 2

Related Work

This chapter discusses prior work on *(i)* text input models and metrics, *(ii)* text input methods beyond the desktop environment, *(iii)* the machine learning advancement in Natural Language Processing (NLP) and *(iv)* design principles for human-AI interaction.

2.1 Text Input Models and Metrics

The backbone of text input research is the evaluation models and metrics: when a new input method is invented, we need to quantify its performance, mostly in speed and accuracy. For speed, the metric Words-per-Minute (WPM) is used, and for accuracy, different error rates are calculated, which will be specified later. A robust and reliable metric is even more important when evaluating intelligent text input systems, as features such as auto-correction and predictions are changing the way we type, yet their performance is difficult to measure given the complexity of the algorithms. While calculating the speed is straightforward, measuring text entry error rates presents a particular challenge. First, it is difficult to identify the error text composed by a user without a reference text. Hence modern text input research mainly utilizes the transcription tasks during the evaluation. Second, there are different kinds of errors, *i.e.* errors that are fixed during the typing process, and errors that remained in the final text. Early experiments in the 1990s did not use the error metrics in their evaluations: some studies simply ignored errors [62], while other prohibited erroneous characters from appearing [94], or disabled all error correction [68]. Those evaluations were both unnatural and constrained, and could lead to further errors in the transcription [69].

In 2001, seminal work by Soukoreff and MacKenzie [87] began to loosen these constraints by using the Levenshtein minimum string distance algorithm [61] to calculate errors based on the edit distance between two strings[1]. BACKSPACE was now allowable as the sole means of error correction, and participants could enter text freely without having to resynchronize after an inserted or omitted character. Later on, Soukoreff and MacKenzie's influential 2003 paper [88] showed how to

[1]The edit distance is the minimum number of character insertions, deletions, or substitutions required to turn one string into another.

calculate error rates with the Input Stream (IS) model, where characters entries and BACKSPACES (<) were recorded sequentially as a stream during typing, such as:

thr<<**e q**uck<<**ick br**wn<<**o**n<**wn**

The resulting transcribed string from the IS above is "the quick brown" with six BACKSPACEs encoded as error corrections during entry. The IS not only contains all information necessary to extract the final transcribed string, but also contains all dynamic information about the text entry process that created it. From this information, Soukoreff and MacKenzie [88] defined three separate error rates: 1) uncorrected errors, for those remaining in the final transcribed string; 2) corrected errors, for any characters backspaced during entry; and 3) total errors, for their sum. Character- and word-level analyses and metrics based on the IS model were also proposed in the later work [108, 118]. However, because the IS model is strictly serial and only able to append edits to its right-hand side, numerous editing restrictions are imposed in this paradigm: (1) BACKSPACE is the only error correction mechanism allowed; (2) the text cursor must always remain at the end of the string entered thus far. No mouse or arrow keys can be used to move the text cursor; (3) selecting-and-replacing text is not allowed; and (4) autocorrection and other intelligent functions are not feasible. Character-level analyses were extended to the input stream by Wobbrock and Myers [108]. Other error-related metrics such as Cost per Correction [5, 41] and word error rate [58] have been introduced. However, as before, these metrics rely on the sequential input stream paradigm, and therefore are similarly constrained. Method-independent evaluations based on the IS model cannot accommodate many modern text entry behaviors. In section 3.1, I present the Transcription Sequence model [127], which remedies these limitations, and offers a method-independent evaluation paradigm and platform-independent evaluation testbed.

Besides the difficulty of error calculation, there is another challenge for text entry evaluation: as we use two metrics *speed* and *accuracy* to measure a text input method, we could not draw a firm conclusion on the overall performance. Furthermore, as with all human performance, speed and accuracy trade off against each other, complicating the assessment in the presence of such tradeoffs. Previous work attempted to derive simple unified metric for text entry performance. For example, Wobbrock [67] proposed Adjusted Words per Minute (AdjWPM) as

$$AdjWPM = WPM \times (1 - E) \tag{2.1}$$

where E refers to the uncorrected error rate [88]. However, the definition of AdjWPM lacks any theoretical basis. Related research on the speed-accuracy tradeoff has a long history, specifically arising with Fitts' law [35] in 1954 for aimed pointing movements. Fitts' law is an empirical model that combines the spread-of-hits and pointing time. Inspired by Shannon information theory [80], the law predicts the movement time of an aimed pointing task: it regards the human motor system as a communications channel, which transmits information during pointing. Based on Fitts' law, Crossman [24] provided a correction to normalize the speed-accuracy tradeoff with his corrected throughput measure, which was later popularized by Welford [105] (pp. 147-149). Inspired by the previous work, in section 3.2, I derived a text entry throughput metric [130] based on the information theory [80] as a performance metric unifying the speed and accuracy, which was validated to be the most stable metric under different speed-accuracy conditions.

2.2 Text Input Methods Beyond the Desktop Environment

There are plenty of projects aiming to provide alternative text input solutions beyond the traditional desktop settings. For example, text entry research with wearable devices has gained a lot of attention because of the always available environment. One category is to type on an external device: Twiddler [64] is a one-hand chording keyboard, where the user grabs the device and press multiple keys on a grid to enter a character. Smart watch text entry interactions also falls into this category [45, 50, 60, 74, 116, 117], where the user performs text entry through the interface of the watch device. For example, COMPASS [117] utilizes the rotor of the watch for character selection, and reached 12.5 WPM with a dynamically positioned cursor. WatchWriter [45] implemented both tap and swipe-based text entry on the watch interface. The other category is typing with bare hands with external sensors. For example, TipText [114] and BiTipText [113] project the QWERTY layout on the index finger and enable subtle text entry with thumb-to-index taps sensed by capacitive overlays. Finger-T9 [111] mapped the layout of a 9-key numpad to the finger segments for the thumb to tap. WrisText [42] detects the wrist motions by proximity sensors and enables joystick-like whirling input for text entry.

Alternative text input interactions with intelligent decoding algorithms was also proposed over the past decades: Kristensson and Zhai invented SHARK^2 [59], an on-screen swipe-based text input interaction that achieved fast speed with single hand. SHARK^2 uses template matching based algorithm to recognize the gestures performed by the user and returns word-level candidates. Findlater and Wobbrock proposed a personalized on-screen keyboard [33] that could adapt its layout to the users typing behavior gradually. A follow-up work TOAST [81] employed a Markov model in the keyboard decoding process and achieved 44.6 WPM on big touch screens. The Finger Fitts Law [14] proposed a dual-distribution to model finger's touch point distribution accurately; WalkType [40] incorporated the accelerometer data to improve the touch accuracy during walking conditions; Yin et al. [119] proposed a hierarchical spatial backoff model to make the touchscreen keyboards adaptive to individuals and postures. Weir et al. [103] utilized the touch pressure to "lock" the characters during decoding. Zhu et al. [132] showed that participants could type reasonably fast on an invisible keyboard with adjusted spatial models. Vertanen et al. [99] developed VelociTap, a phrase-level decoder for mobile text entry. Together with its follow-up projects [96, 97], various factors such as visual feedback on touched keys, keyboard size, word-delimiter actions (e.g. a right swipe), and decoding scopes were investigated for phrase-level input. However, they assumed that the user would input word deliminators perfectly without errors, which was not the case in real settings.

Beyond on-screen keyboards, alternative text input interactions are proposed in the air [118], in VR [122], for glasses [121], for accessibility purposes [8, 109], and even on music instruments [30]. However, many text input methods require long training time to adopt, and most of them operate on character or word level (except for the work from Vertanen et al. [96,97,99]), without taking the longer context and the higher-level information into the decoding process. To address those limitations, in section 4.1, I designed a phrase-level input keyboard *PhraseFlow* [129] and evaluated its adoption and performance in the wild; in Chapter 5, I investigated semantic-level emoji suggestion mechanism [124], and invented a speech-based emoji input system called Voicemoji for blind and low vision users [125].

While much previous work focused on user behaviors during mobile text entry, there have been a few projects that improved upon the text editing process. Most of the editing interactions are shipped on current commercial keyboards, such as the touchpad-style cursor moving interaction on the Apple iOS keyboard, and the swipe-based cursor moving interaction on Gboard [44]. Fuccella et al. [38,39] designed a gesture set that could be performed on the keyboard area for different editing operations, such as cursor movement and copy/paste. Their gestural method was shown to be faster and more favored than the *de facto* touch+widget method. For text correction interactions, the smart-restorable backspace [4] technique allows users to perform a swipe gesture on the backspace key to delete the text back to the position of an error, and restore that text by swiping again on the backspace key after correcting the error. To determine error positions, the technique used Myers' algorithm [70] to compare the edit distance of the text and the word in a dictionary. ReType [85] is a desktop keyboard based correction interaction that locates the the error positions guided by gaze information and correct them with the keyboard input. In section 4.2, I present *Gedit* [128], a set of on-keyboard gestures for mobile text editing, including cursor control, copy, cut, paste and undo commands. I then present a novel concept for text correction interaction: *Type, Then Correct* (TTC) [126] in section 4.3, which allows the user to type the correction first, and apply it to the error without the need of cursor reposition. I also realized the concept with three interaction techniques, utilizing the state-of-the-art deep neural language models.

2.3 The Machine Learning Advancement in Natural Language Processing

The decoding algorithms of text input intrinsically belong to the Natural Language Processing (NLP) research. In general, the decoder of a intelligent keyboard contains two essential models: the spatial model and the language model [19, 37, 43, 56]. The spatial model relates intended keys to the probability distributions of the input information (such as touch coordinates) and other features [9,33,119,132]. The distribution is then combined with a language model, such as an n-gram back-off model [56], to correctly decode noisy touch events into the intended text [37,43]. Borrowing the idea from speech recognition, the classic approach to combining the spatial model and language model estimations is through the Bayes' rule, as in Goodman et al. [43]. Practical keyboards may also model spelling errors by adding letter insertion and deletion probability estimates in its decoding algorithms [75].

Recent advance in Natural Language Processing (NLP) has demonstrated the power of neural networks. A noticeable innovation of deep learning in NLP is the invention of the attention mechanism [10], which tries to mimic the human brain actions as our brains tends to focus on certain salient pieces given the whole paragraph. Later on the Transformer model was proposed [93], and became the standard NLP neural model because it achieved superior task performance with pure attention-based architectures instead of using RNNs. The Transformer model has enabled giant neural models such as GPT-2 [77], GPT-3 [17], BERT [27] and XLNet [115]. Deep language models trained by neural networks has achieved significantly low perplexity [17, 27, 93], and the state-of-the-art performances on many language understanding tasks, including text summarization [79], document classification [76], question-answering [120] and conversation generation [17]. Unlike the

traditional n-gram models, neural networks can take longer context into the modeling process, which allows the model to understand and generate text on the semantic level.

NLP algorithms have also been applied to text input related areas. For example, Xie et al. [112] presented an encoder-decoder RNN model for text correction. Their model was built upon a sequence-to-sequence model for translation [10]. Neural models were also applied for gesture typing decoding [1]. In fact, commercial products related to text input such as Gmail has already released features like smart reply [55] and smart compose [20] that are backed by neural language models to generate meaningful email responses. An important advance of deep learning neural model over the traditional n-gram based model is the ability to incorporate longer context: instead of operating on character, or word level, the neural models could take the whole phrase or paragraph into consideration. As a result, neural language models not only yield more accurate results, but enable the semantic-level understanding of the text, which was impossible for traditional methods. Many of my previous work, including the TTC correction interaction, semantic level emoji suggestion, and Voicemoji, infused the neural networks into the system design. I plan to extensively utilize the neural network models in my proposed projects.

2.4 Design Principles for Human-AI Interaction

According to Beaudouin-Lafon [13], there are three paradigms for an interaction system: *computer-as-tool*, which extends human ability by providing enable facilities; *computer-as-partner*, which "embodies anthropomorphic means of communication" during the interaction; *computer-as-medium*, which behaves as a medium for human communication. Powered by the advanced machine learning algorithms and big data, text input technology is gradually moving from the the first paradigm, *computer-as-tool*, towards the second, *computer-as-partner*: instead of generating symbols on character by character, it is now helping expressing thoughts on word, phrase, and semantic level. The problem also emerges: how to balance the "user control" and "machine intelligence"? AI usually behaves like a black box, introducing uncertainty into the interaction; text entry tasks, on the other hand, require precision and predictability.

There are two main factors that affects the interaction quality: automation and controllability. Automation measures how many tasks the machine is in charge of, and to what degree they are controlled without the human participation. Controllability [78] reflects to what extent the users can control the interaction process, and to what extent they can predict and refine the result. Shneiderman [83] proposed a two-dimensional framework on the relationship of the two factors, pointing out that a system with both high levels of controllability and automation is considered as reliable, safe and trustworthy. There are multiple principles and guidelines on how to design the interaction with general AI systems [2, 51, 72]. For example, Horvitz [51] summarized several critical factors for mix-initiated user interfaces, such as *employing dialog to resolve key uncertainties* and *continuing to learn by observing*; Amershi et al. [2] consulted with design practitioners and proposed 18 guidelines for human-AI interaction, covering four stages of the interaction: initiation, during interaction, when wrong and over time. Those guidelines also apply to text entry systems: for example, the guideline *learn from user behavior* applies to the personalized user dictionary design in mobile keyboards; the guideline *employing dialog to resolve key uncertainties* can be demonstrated

on a auto-correction system by providing alternative candidates in the correction results. In order to deliver the high-quality intelligent text entry experience, I applied those guidelines during the process conducting my own research.

Chapter 3

Models and Metrics for Intelligent Text Input Methods

I present my work on the models and metrics to understand, quantify and evaluate the intelligent text entry process, which serve as the foundation of my other research work. Specifically, I introduce the *transcription sequences* (T-seq) model with new metrics based on the model for unconstrained text entry evaluation, and the *text entry throughput* which measures the information transmission rate of the text entry process, and combines the speed and accuracy into a single performance metric.

3.1 The Transcription Sequence Model: Enable Less Constrained Text Entry Evaluations

When seeking to quantify the performance of a new text entry method, creators can benefit from pre-existing testbeds, rather than having to build their own evaluation tools. Such pre-existing testbeds must be "method independent," working without any feature-specific knowledge of the text entry methods they evaluate. Therefore, such tools receive text and compute metrics (e.g., words per minute [65], various error rates [88], and more) without knowing the mechanisms by which that text is produced. The prevalent evaluation paradigm addressing this need is that of Soukoreff and MacKenzie [87, 88], which, in each text entry trial, presents a string that a participant transcribes. The accompanying model encoding a participant's text entry process is called the input stream (IS), which is a strictly sequential record of each character entry or BACKSPACE. Unfortunately, the IS model cannot accommodate many modern text entry behaviors, including using the mouse or arrow keys to position the cursor, text highlighting and replacement, auto-correction, word prediction, undo, and copy/paste, to name a few. Traditionally, the only means of error correction in the IS paradigm is BACKSPACE, and only then from the end of the currently entered text.

To enable less constrained text entry evaluation process, I invented a new model, the transcription sequence (T-seq) model, that supersedes the IS model for general-purpose method-independent character-level text entry evaluation. The T-seq model makes abstraction of the text entry process

into three parts: 1) the user's actions on the text entry method, 2) the black box text entry method itself, and 3) the resulting text output that enters the method-independent evaluation testbed. In short, (1) acts on (2) to produce (3). The actions can be further abstracted into three categories: *insertion*, *deletion* and *substitution*. A T-sequence is thus a sequence of snapshots of the entire transcribed string after each text-changing action is taken by the user. I designed an algorithm, called *INFER-ACTION*, to infer the action taken at each step by comparing every pair of successive snapshots. After the whole action sequences are recovered by the algorithm, character-level metrics, such as uncorrected error rate (UER), corrected error rate (CER) and total error rate (TER) can be computed. Based on the T-seq model, new metrics are proposed, such as character per action and correction efficiency, which offer new insights about the evaluation. Finally, I implemented and open-sourced a web-based successor to the TextTest desktop application [108] called TextTest++ (Figure 3.1), which encapsulates the T-seq model and enables text entry innovators to study their inventions on any platform or device capable of running a web browser.

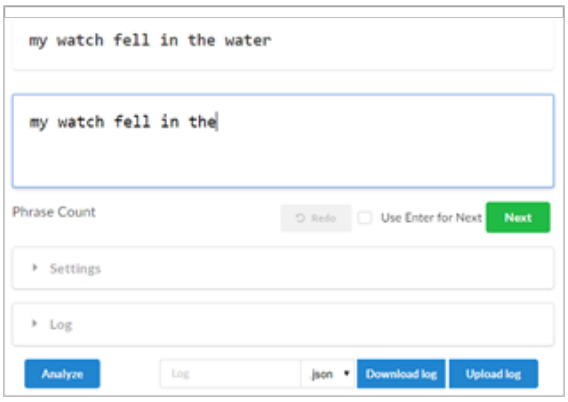

Figure 3.1: TextTest++ is a new web-based text entry evaluation testbed that produces the traditional metrics from the IS paradigm and the new metrics from the T-sequence paradigm

To evaluate the model, I conducted a laboratory experiment on three keyboards: a laptop keyboard, an on-screen desktop keyboard and a smartphone touch keyboard. The results showed that the model was capable of handling all modern text entry behaviors such as cursor-moving and auto-correction. To validate the correctness of the INFER-ACTION algorithm, I further ran extra experiments on the Dasher [102] gesture typing [59] and T9 [73] keyboards. The data from the two experiments showed that results generated from INFER-ACTION matched 100% with the ground truth, indicating that T-seq model was generalizable across different text entry methods.

3.2 Text Entry Throughput: Towards Unifying Speed and Accuracy in a Single Performance Metric

Human-computer input performance inherently involves speed-accuracy tradeoffs — the faster users act, the more inaccurate those actions are. Therefore, comparing speeds and accuracies separately can result in ambiguous outcomes: Does a fast but inaccurate technique perform better or worse overall than a slow but accurate one? For pointing, speed and accuracy has been unified for

over 60 years as throughput (bits/s) [24, 105], but to date, no similar metric has been established for text entry. In modern text evaluation tasks, participants are instructed to "proceed quickly and accurately" [88, 118]. However, every human actor has their own internal subjective speed-accuracy bias, which may change with purpose and context. Thus, separate measures of speed and accuracy will vary under different speed-accuracy conditions. The goal of this project is to devise, theoretically and empirically, a robust performance measure for text entry that conveys the information found in speed and accuracy measures, while also remaining stable across various speed-accuracy biases.

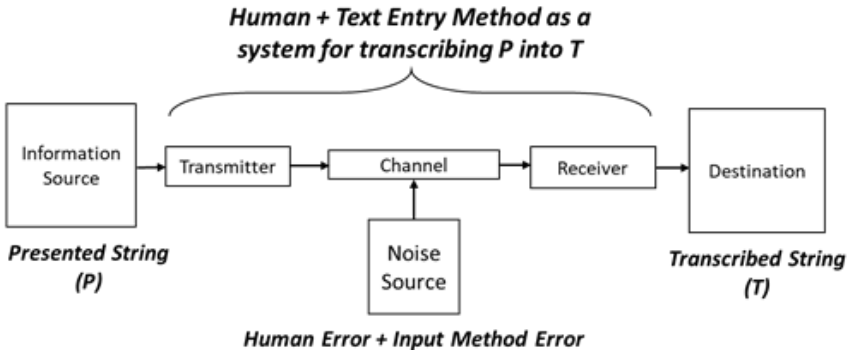

Figure 3.2: Shannon's information transmission model, with labels in bold mapping the model to the text entry transcription process

I propose that text entry evaluations are describable in terms of Shannon's model. A text input system is, quite literally and conceptually, a communications channel in the information-theoretic sense (Figure 3.2). Shannon's information theory [80] should therefore shed light on the evaluation of text input methods. Intuitively, the amount of information transmitted via a text input method per unit time, termed throughput, reflects the input efficiency of the method. In a text entry transcription task, the presented string (P) can be regarded as the information source; the transcribed string (T) can be regarded as the information destination; and the system of human-plus-input-method can be regarded as a discrete channel perturbed by noise, which are errors. Characters are signals transmitted through the input process (e.g., typing), modified by noise and displayed on the screen. From such an information transmission model, I derived a formula to calculate the *text entry throughput* based on text entry speed and the uncorrected error rate. Specifically, the amount of information transmitted from the source X to the destination Y during a text entry process was defined as "mutual information" $I(X,Y)$, which is calculated as follows:

$$I(X,Y) = H(X) - H_Y(X) \tag{3.1}$$

$H(X)$ represents the information, or "entropy" of the source, and $H_Y(X)$ represents the conditional entropy, which is called "equivocation" according to Shannon. It refers to the information lost during transmission. By converting the error rate into transmission probabilities, the mutual information of a text entry procedure can be calculated. As a result, the *text entry throughput* can be calculated without requiring any extra information besides the speed (WPM) and accuracy (UER) logs. To facilitate the usage of the metric, I open-sourced the script of calculating throughput from *TextTest++* logs.

I conducted an in-lab experiment to explore the practical value of the metric. To manipulate different speed-accuracy conditions, I designed a game-like text entry experience, where participants will gain scores when they met certain criteria, or loose scores when they failed to do so. Conditions biased towards accuracy had more strict rules on how accurate the transcription should be, while conditions biased towards speed encourage the participants to type fast without caring too much about accuracy. The experiment showed that for the same person with the same text entry method, throughput measure exhibited less variation compared to other text entry metrics (including WPM, UER and AdjWPM) across different speed-accuracy conditions, suggesting that the measure characterizes the communications channel itself, apart from a human actor's particular speed-accuracy bias. The results were validated via various statistical tests including *the coefficient of variation*, *non-parametric Friedman tests* and *confidence intervals* estimated from *bootstrapping*.

In this chapter, I presented the transcription sequence model and the text entry throughput for intelligent text entry evaluations, which enabled me to quantify the performance of different text entry methods. I now present my work on intelligent text entry methods in the following chapters.

Chapter 4

Intelligent Text Input Methods

Using machine learning algorithms, I design and implement text input interactions and systems that operate beyond the character level manipulation. In this chapter, I present my work on a phrase-level input system *PhraseFlow*, along with the work on mobile text editing interactions *Gedit* and *Type, Then Correct*.

4.1 PhraseFlow: Designs and Empirical Studies of Phrase-Level Input

Autocorrection has become an essential part of touchscreen smartphone keyboards. Due to the small screen size relative to the finger width, fast typing on a smartphone without autocorrection can produce up to 38% word errors [9,37]. To remedy the problem, given a sequence of touch points, a keyboard decoder can use spatial and language models to find the best candidate and performs correction on the typed text. Simulation studies show such auto-corrections can dramatically reduce the error rate in touch keyboards [37]. However, word-level decoding has two major drawbacks. First, at times it can be difficult for the decoder to determine if a word makes sense without incorporating the future input context. For example, if a user types *he loces*, the keyboard may correct *loces* to *loves*; however, if the user continues typing *in Paris*, the expected correction should be *lives*. Not incorporating the future context can either lead to wrong corrections or fail to correct the text. Second, space-related errors often cannot be handled well without future context. Word-level decoding uses the space key tap as an immediate and deterministic commit signal, thus does not afford the benefit of correcting for superfluous touch on it or alternative possible user intentions such as aiming for the C V B N keys above the space key. As a consequence, space-related errors such as *th e*, *iter ational* can not be properly handled. Furthermore, a word-level decoder often fails to correct contiguous text without spaces such as *theboyiscominghomenow*, as it mainly consider word candidates.

To address the problem, I designed *PhraseFlow*, a keyboard prototype that focused on designing and studying the interfaces to support the phrase-level decoding. Phrase-level decoding may continue to decode the touch points even if the space key is pressed, and outputs phrase candidates. *PhraseFlow* aims to address three essential types of questions in the phrase-level input interaction:

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

To implement *PhraseFlow*, I modified the Finite State Transducer (FST) based decoder [75] of Gboard to support phrase level decoding. Specifically, I disabled the reset action of the decoder when the space was entered, and enlarged the decoding span to incorporate larger input context. To explore the design space of *PhraseFlow*, I iterated on multiple options of: 1) visual correction effects; 2) decoding commit gesture and behavior; and 3) suggestion displays. The first version of *PhraseFlow* was with similar designs to the previous phrase-level input work [96, 97, 99], where the keyboard displayed phrases as candidates, and correct the input text on every nth space the user typed. The study results showed that phrase-level input with such designs introduced extra cognitive loads to the user, where the users were constantly distracted by the literal errors they typed, and had to spend time checking the results after the keyboard corrected multiple words at once. By incorporating empirical study results from the iteration, I improve the keyboard design by adding an first-in-first-out input buffer to make the keyboard 1) only correct one word at a time after the space was pressed; 2) be able to incorporate newer text in the decoding context in a continuous way. The keyboard also displayed real-time correction in place, reducing the feeling of uncertainty of the user. Figure 4.1 demonstrates the correction interaction of *PhraseFlow*.

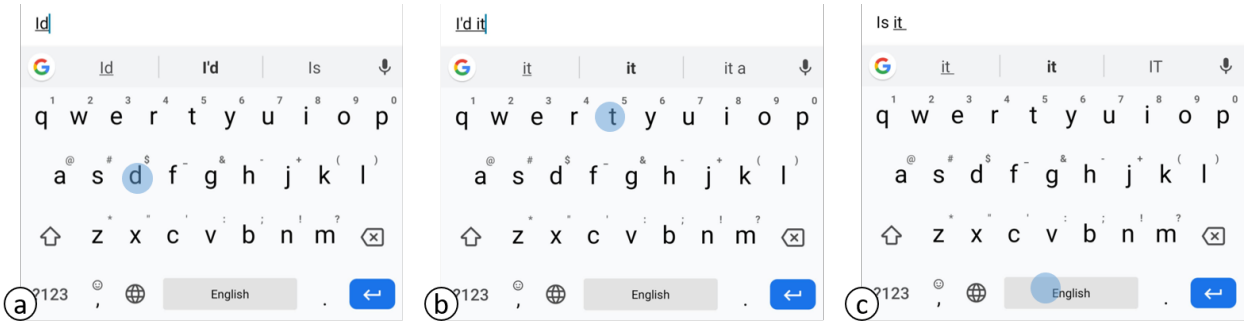

Figure 4.1: The final version of *PhraseFlow*. (a) When the user typed "id" (but meant "is") , (b) it was first corrected to "I'd"after the first space press. However, the correction was not committed (it stays in the buffer). (c) After the user typed text "it", the word was finally corrected and committed as "is" on the second space press

Computational studies show that phrase-level input reduces the error rate of auto-correction by over 16%, and an in-lab study shows that users could adopt *PhraseFlow* quickly, resulting in 19% fewer error without losing speed. To test the user acceptance of the keyboard, I conducted a six-day deployment study with 42 participants. During the study, participants used *PhraseFlow* as their primary keyboard, and filled a feedback survey every other day on their preference, perceived accuracy and perceived speed of using *PhraseFlow*. The survey results showed that overall 78.6% of the participants would like to have phrase-level typing in their future keyboards, in comparison to 7.1% of the participants disliked the feature. Overall, the study results suggest phrase level input is a promising feature for future mobile keyboards.

4.2 Gedit: Keyboard Gestures for Mobile Text Editing

On touch-based mobile devices, text editing could be a tedious process, which still largely borrows from desktop mouse interactions. Modeless editing operations [91] such as copy, paste and cut are often handled in a touch+widget [38] manner. However, the cursor is positioned using tap gestures, which are error prone because of the fat finger problem [100], especially when text characters are small [6]. Also, users must press long enough to exceed a time threshold to trigger selection mode, and later select "copy" in a popup menu to complete the operation. These extra steps significantly slow text editing on mobile touch screens. Moreover, if an editing operation happens during the text entry process, one must lift one's finger from the keyboard area to directly interact with the text input area, introducing unnecessary round-trips [36, 46, 49] and breaking the flow of typing.

To improve the editing experience on mobile devices, I designed a gesture-only system, *Gedit*, consisting of a set of on-keyboard gestures for cursor movement and text manipulation commands. *Gedit* contains ring and flick gestures for cursor control, bezel gestures for entering "editing mode," and letter-like gestures for copy, paste, cut, and undo. In editing mode, what was cursor movement becomes text selection. Editing gestures can be performed with one or two hands. For cursor movement, the user could perform a circular gesture clockwise or counter-clockwise to move the cursor left or right, or perform flick gestures to move the cursor up or down. For manipulation options, the user could perform a bezel gesture — swiping from the edge of the screen to enter the edit mode, and draw characters with another hand to execute commands. For one-handed mode, the bezel gesture and the character drawing gesture are combined into one command, as shown in Figure 4.2.

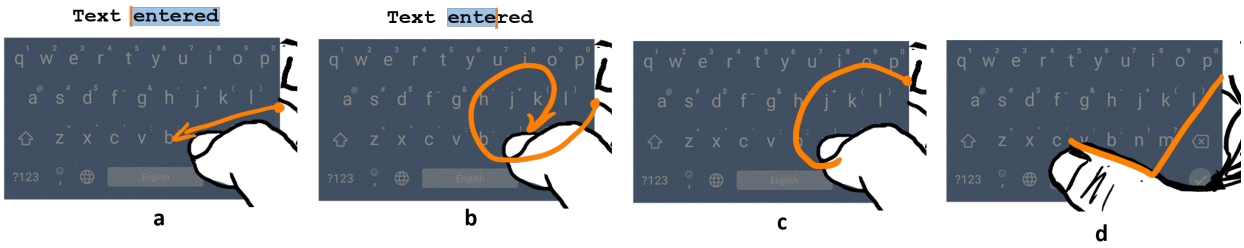

Figure 4.2: Some *Gedit* editing gestures in one-handed use. All gestures start from the right edge: (a) a flick left to select a word; (b) a clockwise ring gesture selects characters to the right of the text cursor; (c) the copy gesture "C"; and (d) the paste gesture "V"

To evaluate the efficiency and the usability of *Gedit*, I conducted a user study comparing it with the *de facto touch+widget* method. To encourage participants to use *Gedit*'s gestures and editing features, I designed a set of phrases that contain several appearances of a common string that is either rare (e.g., "Tchaikovsky") or long (e.g., "San Francisco"), which encouraged participants to use copy/cut and paste functions during typing. Results showed that gesture interactions sped up the text editing process compared to the *de facto touch+widget* editing approach of tapping keys and the input area to position the cursor. Participants especially appreciated and enjoyed the capability that *Gedit*'s gestures offered for one-handed use.

4.3 Type, Then Correct: Intelligent Text Correction Techniques for Touch Screens

Besides the slow editing process, another bottleneck for touch screen text input lies in the correction process. On mobile touch-based devices, text correction often involves repetitive backspacing and moving the cursor with repeated taps and drags over very small targets. A potentially fascinating premise is thus *What if we can skip positioning the cursor and deleting errors?* Given that the *de facto* method of correcting errors relies heavily on these actions, such a question is subtly quite radical. *What if we just type the correction text, and apply it to the error?*

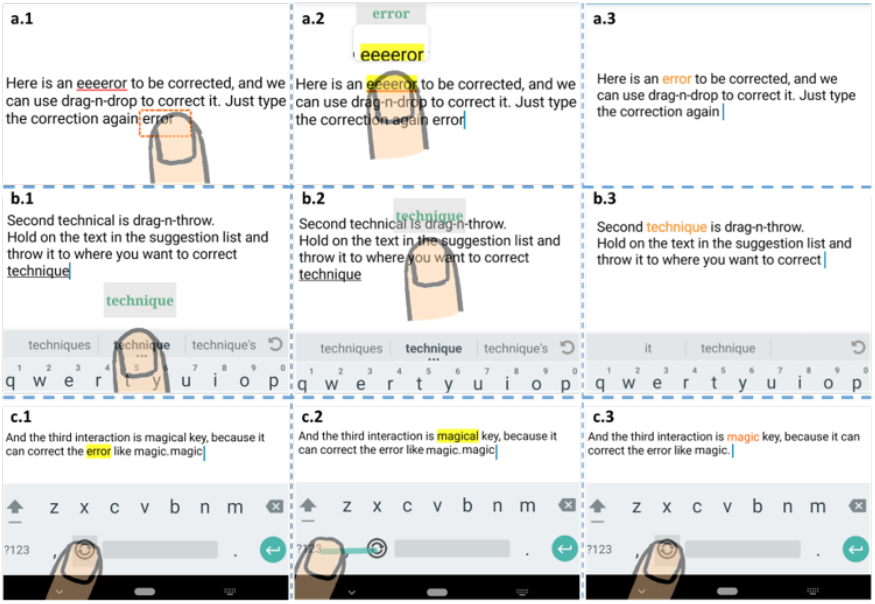

Figure 4.3: Our three interaction techniques. *Drag-n-Drop*: (a.1) type a correction and then touch it to initiate correction; (a.2) drag the correction to the error position. The touched text is highlighted, and the correction shows above the magnifier; (a.3) drop the correction on the error to finish. *Drag-n-Throw*: (b.1) dwell on a word from the suggestion list to initiate correction. The corresponding text will display above the finger; (b.2) flick the finger towards the area of the error: here, the flick ended on "the", not the error text "technical"; (b.3) the algorithm figures out the error successfully, and confirming animation appears. *Magic Key*: (c.1) tap the magic key (the circular button) to trigger correction. Here, "error" is shown as the nearest possible error. (c.2) drag left atop the magic key to show the next possible error in that direction. Now "magical" is highlighted. (c.3) tap the *Magic Key* again to commit the correction "magic".

The concept, called "Type, Then Correct" (TTC), inspired me to develop three novel text correction techniques. The first technique, *Drag-n-Drop*, is a simple baseline technique that allows users to drag the last-typed word as a correction, and drop it on the erroneous text to correct substitution and omission errors [108]. The second technique, *Drag-n-Throw*, is the "intelligent" version of *Drag-n-Drop*: it allows the user to flick a word from the keyboard's suggestion list towards the approximate area of the erroneous text. The deep learning algorithm finds the most likely error within the general target area based on the thrown correction, and automatically corrects it. *Drag-n-Throw* is faster than *Drag-n-Drop*, because the user does not need to drop the correction on the error location. Unlike the above two techniques, the third technique, *Magic Key*, does not require direct interaction with the text input area at all. After typing a correction, users simply press a

dedicated *Magic Key* on the keyboard, and the algorithm highlights possible error words according to the typed correction. One can then step through the possible error words by directionally dragging atop the *Magic Key*. When the desired error word is reached, users tap the *Magic Key* again to apply their correction. All three of our interaction techniques require no movement of the text cursor and no use of backspace. The illustration of each interaction technique is shown in Figure 4.3.

The *Drag-n-Throw* and *Magic Key* rely on NLP algorithms to detect possible error candidates given the correction. I applied a recurrent neural network (RNN) encoder-decoder model which was widely used in translation tasks. The encoder contains a character-level convolutional neural network (CNN) [57] and two bi-directional gated recurrent unit (GRU) layers [22]. The decoder contains a word-embedding layer and two GRU layers. To collect large amount of text correction data, I applied several artificial perturbations to online-review text datasets [131], such as typo simulation and word deformation. The model achived 75.68% and 81.88% accuracy on the CoNLL 2013 [71] and the Wikipedia revision datasets [123], respectively. To evaluate the interactions in real settings, I compared the TTC interactions with the *de facto* cursor-based correction interaction in a lab study. Two types of tasks were conducted: a correction task where participants needed to correct the given erroneous phrases, and a free composition task where the participants composed messages freely without correction and corrected the errors after they finished composing. The results showed that the two intelligent interactions, *Magic Key* and *Drag-n-Throw*, performed significantly faster than the cursor-based interactions in the correction task, and were preferred by participants. In the composition task, *Drag-n-Throw* achieved 87.9% success rate while *Magic Key* achived 97.0% success rate.

4.3.1 JustCorrect: Intelligent Post Hoc Text Correction Techniques on Smartphones

TTC inspired a follow-up work on mobile text correction interaction: *JustCorrect* [25], which was led by Wenzhe Cui from Stony Brook University, and I was a co-author of the project. *JustCorrect* employs the same concept as TTC, and simplifies the correction interaction one step further: the keyboard directly assumes the best error location based on the input correction, without the need to specify where the error is. In this way, after entering the correction at the end, the user can simply press a button to correct the error. Alternatively, the user could switch the entry mode (for example, switch from touch typing to gesture typing) to enter text. The keyboard will automatically regard the text as the correction and apply it to the error. The correction interaction of *JustCorrect* is shown in Figure 4.4. The lab experiment showed that correction performed by JustCorrect were faster than TTC interactions.

Figure 4.4: (1) The user enters a sentence with an error *jimo* using tap typing; (2) To correct *jimo* to *jumps*, they can either tap-type *jumps* and press the editing button (2a), or switch to gesture type *jumps*(2b). (3) JustCorrect then substitutes *jimo* with *jumps*. Two alternative correction options are also presented. The editing procedure involves no manual operations except entering the correct text.

The work in this chapter demonstrated that text input methods can benefit from the power of advanced machine learning algorithms. The FST support for continuous decoding enabled the phrase-level input interaction, which otherwise could not have been successful; Without gesture recognition algorithms, the interaction of *Gedit* could not have been implemented; Using neural network language models, I was able to train the algorithm on millions of text data from the real life, and make the keyboard recognize semantic and grammatical errors in the TTC interactions. In the next chapter, I further combine intelligent methods with non-textual symbol (emoji) input methods, exploring the applicability in computer-based communication and accessibility scenarios.

Chapter 5

Intelligent Entry Methods for Emojis

In this chapter, I present an empirical study of comparing the effect of the emoji suggestion mechanism on people's communication experience. I then introduce a speech-based emoji entry system for blind and visually impaired users.

5.1 Comparing Lexical and Semantic Emoji Suggestions

Most forms of text-based computer-mediated communication (CMC) lack non-verbal expressions like vocal tones, facial expressions, and gestures that are useful in face-to-face conversations. However, several studies have shown that emojis can facilitate affective communication [23, 53, 107]. Emojis are already widely used in text-based CMC, with nearly every instant messaging platform supporting their entry. As there is a large and growing set of emojis, manually searching for and selecting emojis can be a tedious task interrupting the flow of text entry. Commercial products that automatically suggest emojis have helped the emoji entry process become more seamless [52, 101]. These products usually come in two variations — lexical and semantic suggestions. With lexical suggestions (e.g., Gboard), relevant emojis appear in a candidate list based on recent keywords typed by the user. With semantic suggestions (e.g., Dango [52]) proposed emojis are based on the meaning of the message's content rather than on specific keywords. Examples of different suggestions are shown in Table 5.1.

Table 5.1: Examples of lexical and semantic emoji prediction. With lexical prediction, the suggested emojis are related to the literal meaning of certain keywords. With semantic prediction, the suggestions focus on the meaning of the sentence

Sentence	Lexical	Semantic
I enjoyed the fish tonight very much!	🐬🐟🍣🐠🍥	😋😁😊😄💗
I love him but he just ignored me...	😍😘😗😙😻	💔😟☹️😦😢
I'm tired of "happy birthday"	🎈🎁🪂🎉💗	✌️😒😑😐😪

Although emojis themselves are known to enrich conversations [23,53], the role that different emoji suggestion systems play has not been explored. Instead, prior work on suggestion systems has focused on retrieval precision and recall [12, 29, 31]. But how do different suggestion mechanisms influence emoji usage? How do they differ in terms of usability? How do they affect the chat experience?

To investigate these questions, I implemented a keyboard capable of offering both lexical and semantic emoji suggestions. I conducted an in-lab study with pairs of strangers using keyboards with three emoji suggestion mechanisms: no suggestions, lexical suggestions, and semantic suggestions. The participants were instructed to have conversations with each keyboard for 10 minutes on any topic, but they could also use the "recent activity" as a starting point. The results showed that there was no significant difference in the number of total characters and emojis entered by the participants in different conditions, indicating that the chatting experience between strangers is not influenced by the emoji use. The finding was not explored by previous literature focusing on communication between friends and family members [23, 53]. However, participants using the semantic level emoji suggestion keyboard picked significantly more emojis from the suggestions than those using the lexical suggestion keyboard.

To evaluate the emoji usage in daily settings, I also conducted a 15-day field deployment. The deployment contained three 5-day periods, where in the first and the last period participants used the keyboard with no emoji suggestions, and in the second period the participants used the keyboard with either lexical or semantic emoji suggestions. I found that emoji suggestion systems increased emoji usage overall, with users picking more emojis via semantic suggestions versus lexical suggestions or no suggestions.

From both studies, I found that although suggestion mechanisms did not have a significant effect on the participants' perceived chat experience, they facilitated users' needs of inputting emojis in various ways: 1) semantic suggestions were perceived as more relevant to the message content, while lexical suggestions were perceived as containing more unusual emojis 2) the semantic suggestions served as a clue to the tone of the message and even changed the user's input behavior in some cases.

5.2 Voicemoji: Speech-based Emoji Entry System

As emojis become an essential element of online communications including text messaging and social media posts, blind or low vision (BLV) users also encounter more emojis online. According to a recent study by Tigwell *et al.* [92], 93.1% of BLV users encounter emojis each month, and 82.7% of them utilize emojis at least once a month. However, due to emojis' similarity to images and the lack of accessibility support for screen readers [92], current emoji entry methods, including *emoji keyboards*, *emoji shortcuts*, and *built-in emoji search* are cognitively demanding and unreasonably time-consuming for blind and low vision (BLV) users. Such limitations hinder BLV users from using emojis easily, causing social exclusion for BLV users, and reducing their communication efficacy [95]. Through my interviews with BLV users, I find that there are four major challenges of current emoji

entry methods: 1) the entry process is time-consuming; 2) the results provided by the methods are not consistent with users' expectations; 3) there is a lack of support for discovering new emojis; and 4) there is a lack of support for finding the right emojis. In summary, the current state of searching for and inputting emojis for BLV users is inaccessible, tedious, and exclusionary.

In order to improve the emoji entry experience for BLV users, I designed and implemented *Voicemoji*, a voice emoji entry system that supports: 1) voice-based semantic-level emoji search, initiated by speaking the phrase "emoji search" + description + "emoji" (for example, "emoji search i am hungry emoji") 2) direct emoji entry with keywords, initiated by speaking the phrase "insert" + description + "emoji" (for example, "insert tornado emoji") 3) context-sensitive emoji suggestions, and 4) manipulation of emojis with voice commands, such as changing the emoji color or skin tone. Specifically, *Voicemoji* detects a set of keywords to trigger the emoji input function, and utilizes the results from the Google search engine to find the most relevant emojis. Powered by deep learning, it also suggests emojis based on the spoken content. With *Voicemoji*, the user can use ambiguous descriptions, such as, "ocean animal emoji," to get a group of emojis including squid 🦑, octopus 🐙, and tropical fish 🐠. Following a similar approach, exploration and learning of new emojis is also possible, which is exceptionally difficult with current emoji input methods. An interaction case of *Voicemoji* is shown in Figure 5.1.

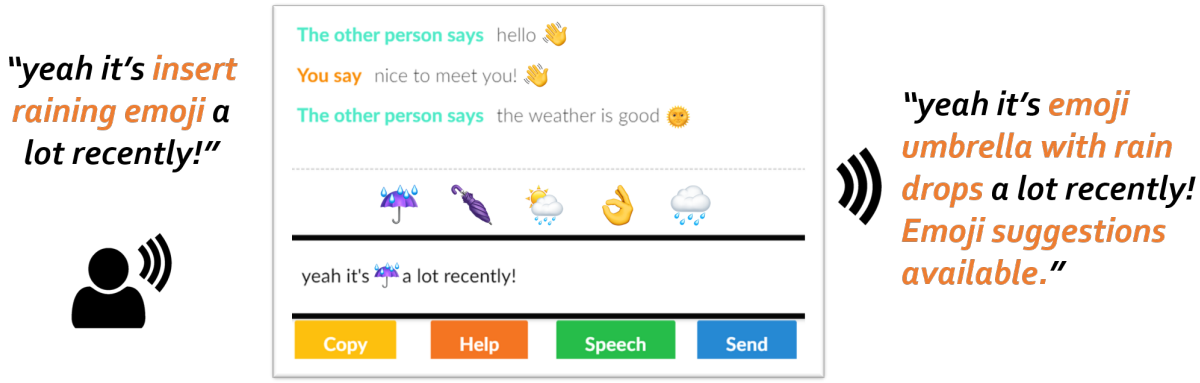

Figure 5.1: The flow of using Voicemoji. Voicemoji is a web application that allows the user to speak text and emojis. It also provides context-sensitive emoji suggestions based on the spoken content

Additionally, *Voicemoji*, at present, supports a rich emoji set accessible through two of the three most spoken languages in the world,[1] Chinese and English. This feature enhances the generalizability of the solution in two respects: (1) language independence (*i.e.*, the method can apply to multiple languages); (2) emoji independence (*i.e.*, the method can output all emojis in the current emoji set). I also open-sourced the implementation to support the research community and provide a platform for contributions from like-minded researchers and developers.

To understand the performance of *Voicemoji* and how it affected the emoji entry experience for BLV users, I conducted a remote controlled study with 12 BLV users from the US and China. In the study the participants were instructed to input emojis using either the iOS keyboard or *Voicemoji*. They also evaluated the emoji suggestion feature by speaking phrases via *Voicemoji* and selecting the suggested emojis that they thought were relevant. The results showed that

[1] https://www.babbel.com/en/magazine/the-10-most-spoken-languages-in-the-world

Voicemoji significantly reduced entry time by 91.2% compared to the Apple iOS keyboard and was preferred by all participants. Participants also found the suggested emojis were mostly relevant. Their feedback implied that *Voicemoji* allowed them to conveniently enter emojis, provided support for exploring and learning new emojis and enriched their online communication experience.

Chapter 6

Proposed Work

My previous work mainly focused on developing intelligent text input systems on mobile devices. In my proposed work, I would like to expand the using scenario to the everyday ubiquitous computing environment, where an input interaction is not limited on a single device's input space and can support cross-device input tasks (ubiquitous), is flexible to perform in mobile settings (flexibility), and can be employed on various surfaces (generalizability). Specifically, I propose *TypeAnywhere*, a QWERTY-based text entry interaction on any surface; *TypeAnywhere One*, a one-handed extension of *TypeAnywhere*; and *AnchorKeyboard*, a non-visual keyboard for large touch screens.

6.1 TypeAnywhere: A Ubiquitous QWERTY Text Entry Solution

Computing now has entered an ubiquitous era [104]: from IoT devices to AR/VR, computers are becoming every part of our life. As a fundamental interaction to communicate with the devices, there is yet no unified text entry solution that is low-friction and always available. Existing text entry solutions either are not mobile enough (e.g. the hardware keyboard), or requires the user to learn extra typing skills (e.g. [59, 64, 90, 117]), or are relatively low-throughput (e.g. watch-based methods [21, 42]). While speech based input is an alternative option, it might be not socially acceptable for certain situations.

QWERTY-based touch typing on physical keyboards remain the most common text input skill of computer users [28] as it utilizes multiple hands and fingers. Although the layout is not invented for optimal speed performance [26], the fact that people are so familiar with the layout has motivated a series of text entry research to build upon QWERTY [45,99,114,118]. I thus want to leverage and resemble the physical keyboard typing experience in this project yet without the mechanical keys: what if we can type on any surface on an imaginary QWERTY just like on a physical keyboard?

I have already started to exploring the answer of the question by developing *TypeAnywhere*, a QWERTY-based text entry solution for everyday use. The concept illustration is shown in Figure 6.1. *TypeAnywhere* uses wearable sensors on two hands that detect the finger tap actions, a decoder that converts the touch sequence into text, and a corresponding interface for text editing. The

hardware is the commercial Tap Strap [1] that uses accelerometers for tap detection. The detected finger tap sequence is fed to a neural decoder modified from the BERT model [27], which then display the output text on the typing interface. Inspired by the work of "Type, Then Correct" [126], I also designed a text correction interaction for *TypeAnywhere* without the need of cursor navigation.

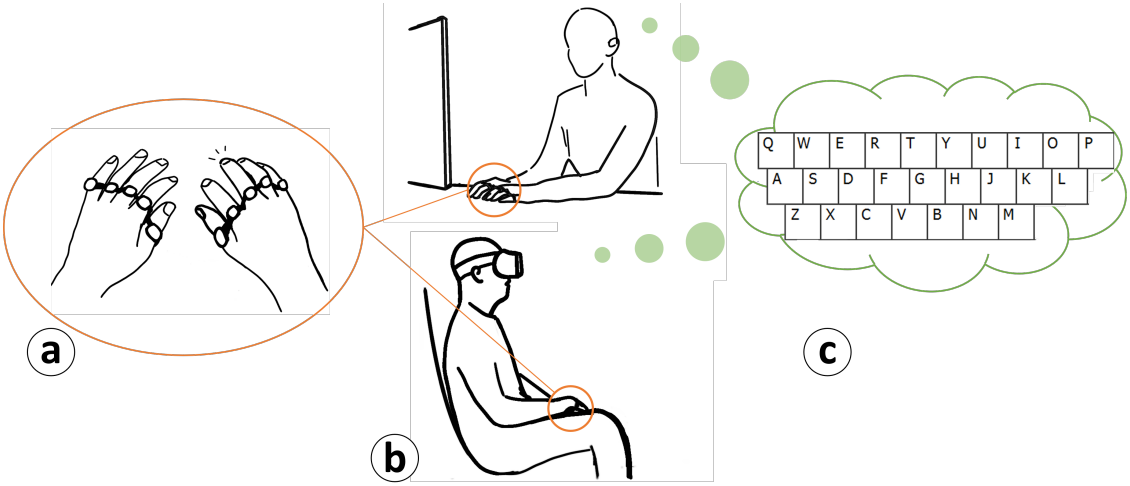

Figure 6.1: *TypeAnywhere* core concepts. (a) Users wear devices on their fingers that detect the finger taps. (b) With *TypeAnywhere*, the user can type on any surface such as the tabletop or their lap. (c) Users type with their own QWERTY key-to-finger mappings, resembling a physical keyboard typing experience

Typing interaction The hardware interface is two wearable devices called Tap Strap, which has five connected rings that can be worn on each finger (Figure 6.2). Each ring contains an accelerometer that detects when the finger performs a tap action, which is reported to have an over 98% detection accuracy. According to my knowledge, Tap Strap is the most accessible yet accurate device for finger tap detection on the market. Because of its small form factor, the user could wear the device comfortably enough.

Figure 6.2: (a) The Tap Strap device (b) Wearing two devices in TypeAnywhere

For the typing interaction, *TypeAnywhere* resembles the exact typing interaction on a physical keyboard: the user performs finger taps based on her own finger-to-key mapping, and the most possible character would be output on the user interface. The user could perform finger taps on hard surfaces such as tabletops and walls, or soft surfaces such as laps. The tap of the two thumbs

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

are mapped to the space key, as most people use thumb for space-key pressing [34]. Essential text editing functions are also mapped to certain multi-finger chords show in Figure 6.3.

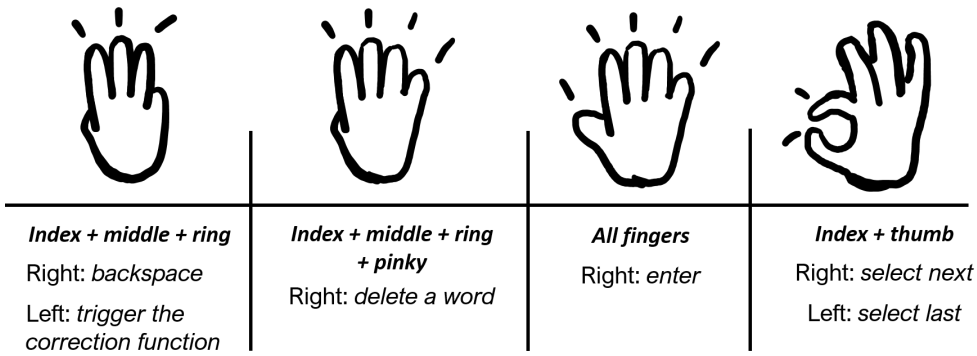

Figure 6.3: The tap chords for text editing functions of TypeAnywhere. For selection gestures, the user can either tap the thumb and index fingers on a surface, or perform a pinch gesture.

Training the language decoder The neural language model used for decoding the tap input is based on the BERT model [27]. Specifically, I added a linear classification layer on top of the BERT-base model to generate the character for each finger tap. The implementation is based on the open-sourced transformers library [110]. The model employed the bidirectional transformer structure [93], and was pretrained with masked language model and next sentence prediction tasks on 3.3 billion words [27]. Given the finger sequence, the model decodes it to text in real time. I trained multiple models with different finger-to-key mappings so that the users did not need to change their typing habit on the physical keyboard.

User study for evaluation To evaluate the performance of *TypeAnywhere* in real settings, I conducted a longitudinal user study. Although *TypeAnywhere* used the QWERTY typing interaction, I anticipated that it still took time for the user to get used to the device and the feeling of typing without a keyboard. I recruited 5 participants (age 23 - 28, all male) for the study via word-of-mouth and online forums, and required that the participants were able to perform QWERTY typing without looking at the keyboard, and were consistent with their finger-to-key mappings (*e.g.* always use the same finger to type a letter). Due to the COVID-19 situation, it was extremely difficult to recruit eligible participants, I thus also included one participant (I refer him as P1) who was not consistent with his finger-to-key mapping, but was able to commit his time for the study. P1 could type on a physical keyboard without looking at it, 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 a fixed key-to-finger mapping for the study.

The study was a 5-day longitudinal study. On each day, the participants first started to practice typing with the device on the web application. The phrase sets for practice and test were from the Mackenzie phrase set [66] and the Enron email phrase set [98]. I shuffled and devided them so that the phrases were different for practice and test, and also different for each day's evaluation. After 30 minutes' practice, they started the evaluation by typing 20 phrases different from the training set as fast and accurately as possible. their typing behavior was logged with the T-seq model [127]. On day 1, the participants performed the typing evaluation on their physical keyboards as a baseline. On day 5, they also performed a tapping-on-laps evaluation with TypeAnywhere. The evaluation

for the physical keyboard session contained 30 phrases, while the evaluation for the on-lap session contained 20 phrases which was the same as the tabletop condition.

Evaluation results Participants' average typing speed is shown in Table 6.1. The pick-up speed on day 1 is 33.33 WPM, implying that participants learnt the interactions without too much effort. The speed is increasing each day and the trend is still growing. Compared to the keyboard condition (70.36 WPM), participants reached 59.67 WPM with TypeAnywhere on the last day. The 15.2% (10.69 WPM) difference between the tabletop and keyboard conditions is smaller than the previously reported difference [34], which means that the users were able to type with TypeAnywhere fast as closely as their keyboard typing.

I also collected the feedback from participants during the debriefing session. The most mentioned benefit of TypeAnywhere was that the user no longer need to move the finger when typing: *"I can just tap the finger without caring about whether the position was right or not"* (P3). The other benefit was that the user could perform typing on any surface similar to typing on a keyboard, without the need to *"learn a new coding system"* (P2).

Table 6.1: The average performance on each day of all participants with TypeAnywhere. Keyboard and on-lap conditions are also listed. The values in parentheses are the standard deviation.

	Day 1	Day 2	Day 3	Day 4	Day 5	Keyboard	On-lap
Speed (WPM)	33.33 (12.25)	38.79 (11.72)	45.49 (12.81)	56.07 (14.33)	59.67 (16.64)	70.36 (21.69)	40.01 (12.88)
Character Error Rate (%)	1.43 (10.14)	1.40 (2.91)	0.33 (1.39)	0.90 (2.57)	1.03 (2.79)	2.84 (5.10)	1.22 (3.79)

6.2 TypeAnywhere One: One-handed TypeAnywhere

While typing with two hands on any surfaces has already lowered the bar of text entry in ubiquitous environments, there are situations where using two hands is not convenient or possible, such as situational impairments where one hand is occupied with another task, or for a person with motor impairments. To further complete the "ubiquitous text input" picture, I propose *TypeAnywhere One*, the one-handed version of *TypeAnywhere*.

Interaction design As the QWERTY layout is designed for two hand usage, I need to redesign the finger-to-key mapping for *TypeAnywhere One*. However, similar to the design choice of *TypeAnywhere*, I do now want to fix the mapping or invent a new mapping scheme, as that will take a long time for the users to learn. The user should still somehow be able to reuse their typing mapping on physical keyboards in one-handed typing. As a result, I decided to use the "mirrored" version of the QWERTY layout when it comes to the other hand. Specifically, when the users are entering the keys on the same side of their hand, they just follow the same key-to-finger mappings on the physical keyboard; when they enter keys which are normally entered by the other hand, they will use the same finger, but the different hand. For example, if one types the letter "A" with the little finger on the left hand, then one will type the same character with the right little finger when using *TypeAnywhere One* with the right hand. According to Grudin [47], the second common substitution error category for physical keyboard typing is *homologous* error, meaning that the letter typed by

the same finger in the same position but with the wrong hand, which accounted for 10% of the overall errors in the data. As a result, it might be more natural and intuitive for the users to type "mirrored" letters with the same finger.

Other than the finger-to-key mapping, I also need to design a set of gestures for selection. Because the interaction only uses five fingers, a finger sequence can be mapped to many word candidates. Hence it is essential to design selection interactions that can effectively lower the uncertainty of options.

Model design I will reuse the main neural network structure from *TypeAnywhere*. However, based on the results from my preliminary simulation, directly applying the model to one hand's finger sequence decreased the accuracy by around 8% and seldom provided the expected words in real settings. There are two ways to address the problem: 1) by fine-tuning the model with detailed structures and more data; 2) by designing disambiguation gestures to incorporate easy human-in-the-loop to select the word collaboratively. I will explore both directions and evaluate their performance.

User study The evaluation user study will be similar to that of *TypeAnywhere*, where the user will perform one handed typing on a hard surface and a soft one (possibly laps). For the hard surface condition, they will perform typing tasks twice with left and right hands. Ideally it will be a longitudinal study where I can further evaluate the learning effect.

6.3 AnchorKeyboard: Non-visual Keyboard for Large Touch Screens

Ten-finger text entry on large touch screens has been extensively studied, including the motor patterns [34], layout personalization [33] and novel decoding algorithms [82]. However, existing touch screen keyboards all require certain degree of visual attention, such as word candidate selection, during the interaction. For blind and visually impaired (BVI) users, they could not benefit from those keyboards. Currently, they use either on-screen keyboards with screen readers, or braille keyboards for typing on large touch screens, which is similar to their text entry interactions on mobile devices. Other solutions, such as using a hardware keyboard or overlay [54], reduce the flexibility and mobility because one has to carry the device around and the interaction position is fixed; yet voice input is sometimes not socially appropriate, and suffers from privacy concerns. In sum, the advantage of the large input space is not fully utilized. On the other hand, more commercial devices are offering only large touch screens as their input, such as the Surface Neo [2], ThinkPad X1 Fold [3], and most commonly, public kiosks. When using those devices, sighted people might also shift their visual attentions on other tasks while typing. Hence an accurate, robust yet easy-to-use non visual keyboard is needed for large touch screens.

The *AnchorKeyboard* is thus to address the issue by enabling non-visual QWERTY style text entry on large touchscreen. The core idea is to identify the typing finger using the touch information relative to other fingers: the user rests all fingers on the screen: when typing a letter, only the

[2] https://en.wikipedia.org/wiki/Surface_Neo

[3] https://www.lenovo.com/us/en/thinkpad-x1-fold

corresponding finger leaves and taps. In this way, other fingers can serve as anchors to identify the typing finger. The exact key can be inferred through the landing point of the finger touch, and can be adapted through time by applying machine learning algorithms. There is thus no fixed keyboard zone where the user has to visually pay attention to, and the user can just use their own finger-to-key mappings same as the physical keyboard, leveraging existing typing skills.

Keyboard design With anchors, the typing finger can be easily identified. However, we still need to decide which key the user is pressing. To solve the issue, I first applied linear regressions on the anchor points to determine the mid-row lines. Then based on the touch distance to the lines, the keyboard can determine the row of the key with empirically defined thresholds. As the typing proceeds, the keyboard can collect more touch points of each key, as shown in Figure 6.4. I then use those points to establish a personalized Bayesian spatial model to determine the probability of each key when a tap happens; the model keeps updating as it collects more data. The preliminary evaluation on my own data showed 96% accuracy with the proposed algorithm. I plan to evaluate the algorithm on more participants.

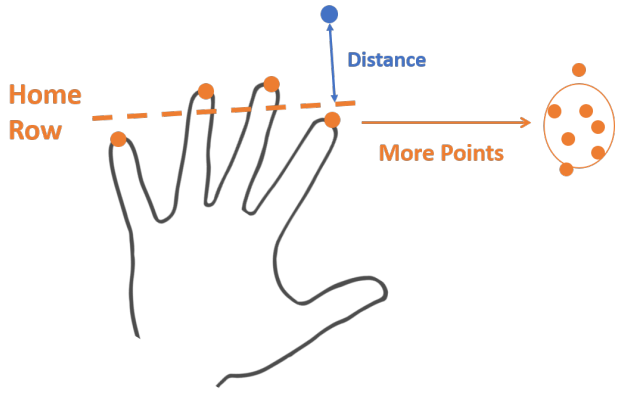

Figure 6.4: Illustration of the process of assigning a touch point to a key. *AnchorKeyboard* requires the user to first put all four fingers of a hand on the touch screen, and finds the homerow using linear regression with touched points. When a finger types, it determines the row of the touched point (blue dot in the figure) by its distance to the home row. When enough touch data is collected for a key, it will then build a Bayesian spatial model and assign the probability of each touch point using the model

Besides only inputting English alphabet letters, the keyboard should also be able to enter numbers and special characters as a complete text input system. I added mode switch gestures for switching between letter/punctuation/number modes, and designed simple swipe gestures for switching between upper/lower case of the letters. In the future, I plan to implement handwriting recognition systems for punctuation input, as it was proved effective for certain characters [32].

Keyboard evaluation I will first perform an offline evaluation task, where I collect typing data from the users and simulate the decoding algorithms on it. I will then improve the algorithm and perform the evaluation on two groups of people: the sighted users and BLV users. The procedure would be similar: transcribing text according to a presented string. However, I will also include special punctuation and numbers in the test phrase set to examine the gestural design. The experiment will also compare *AnchorKeyboard* with other touch-screen keyboards, such as a visible on-screen keyboard for sighted users, and a keyboard with screen readers for BLV users.

Chapter 7

Conclusions

Utilizing advanced AI algorithms, the text input interactions can be made natural and effortless to help users with the communication with machines beyond the traditional desktop situations. This dissertation will demonstrate intelligent text input systems in a full spectrum, including the entering and editing process of text, and the entering process of emojis. Furthermore, the proposed work focuses on providing text entry solutions for the ubiquitous computing environment, which might become prevalent in the near future. In short, the dissertation contributes to the field of HCI by demonstrating the thesis:

> *Artificial intelligence can enable and improve advanced text input interactions, including the entering and editing of text*[1]*, and the entering of emojis*[2]*; intelligent text interactions, in turn, warrant new text entry metrics for their evaluation*[3]*.*

7.1 Contributions

The specific contribution of this dissertation include (proposed future contributions in *italics*):

7.1.1 Theoretical Contributions

- The *transcription sequences* (T-seq) model for unconstrained text evaluations. By comparing adjacent strings within a transcription sequence, we can compute all prior text entry metrics, reduce artificial constraints on text entry evaluations, and introduce new metrics [127] (section 3.1).
- The *text entry throughput*, a metric that unifies the speed and accuracy. *Text entry throughput* is derived from the information theory [80], and it is not sensitive to the speed-accuracy trade-off [130]. The script of computing *text entry throughput* is open-sourced [4] (section 3.2).

[1] Demonstrated in Chapter4, 6
[2] Demonstrated in Chapter5
[3] Demonstrated in Chapter3
[4] https://github.com/DrustZ/Throughput

7.1.2 Artifact Contributions

- The *TextTest++* web platform for conducting text entry evaluation tasks. *TextTest++* provides interface and evaluation metrics based on the *T-seq* model and the *text entry throughput*. The implementation is open-sourced [5] (Chapter 3).
- The design and implementation of a phrase-level input method, *PhraseFlow*. *PhraseFlow* explores various design options through extensive studies to minimize the cognitive load of the user, and improve the practical usability of the phrase-level input [129] (section 4.1).
- The design and implementation of *Gedit*, a set of on-keyboard gestures for convenient text editing on mobile devices. *Gedit* enables the user to execute cursor moving and text manipulation commands with gestures without leaving the keyboard area, and supports one- and two-handed modes [128] (section 4.2).
- The concept of *Type, Then Correct* (TTC) for text correction interaction on touch screens. Instead of the traditional cursor-based text correction interaction, TTC allows the user to type the correction and apply it to the error text without moving the cursor [126] (section 4.3.
- The implementation of TTC concept with three novel correction interactions: *Drag-n-Drop*, *Drag-n-Throw* and *Magic Key*. The later two interactions utilize a neural network model for identifying error candidates. The implementation of the model is open-sourced [6] (section 4.3).
- *JustCorrect*, an extension of the *TTC* concept. *Justcorrect* further reduces the need of manually specifying the error location during the correction [25] (section 4.3).
- *Voicemoji*, a speech-based emoji entry interaction. *Voicemoji* contains several emoji entry commands and provides semantic emoji suggestions to support new emoji exploration. The implementation of *Voicemoji* is open-sourced [7] [125] (section 5.2).
- *TypeAnywhere, a text-entry interaction for ubiquitous computing settings. By detecting the finger taps with wearable devices, TypeAnywhere decodes the tap sequence into text with a neural network model, enabling users to perform QWERTY-style text entry on any surface with their own finger-to-key mappings (section 6.1).*
- *TypeAnywhere One, a one-handed text entry interaction similar to TypeAnywhere. TypeAnywhere One uses mirrored layout for a hand to type the characters on the other side of the keyboard (section 6.2).*
- *AnchorKeyboard, a non-visual QWERTY keyboard for large touch screens. AnchorKeyboard uses the fingers on the screen as anchors to identify the typed finger, and utilizes machine learning algorithms to determine the typed letters (section 6.3).*

[5] https://github.com/DrustZ/TextTestPP
[6] https://github.com/DrustZ/CorrectionRNN
[7] https://github.com/DrustZ/VoiceEmoji

7.1.3 Empirical Contributions

- The evaluation of human behaviors using phrase-level input keyboard *PhraseFlow*.
- The series of usability studies on the intelligent text entry interactions including the *Type, Then Correct*, *Gedit*, and *Voicemoji*.
- In-lab and deployment studies on comparing the effect of lexical and semantic emoji suggestion mechanisms on online communication [124] (section 5.1).
- Interviews to understand the current emoji entry experience of blind and low vision (BLV) users on mobile devices [125] (section 5.2).
- *Evaluations on the performance, learnability and feasibility of TypeAnywhere and TypeAnywhere One.*
- *Interviews to understand the ten-finger typing experience of BLV users on physical keyboards and touch screens. Data collection and analyses of BLV users' ten finger typing patterns on large touch screens (section 6.3).*

7.2 Schedule

The project schedule is planned in Figure 7.1.

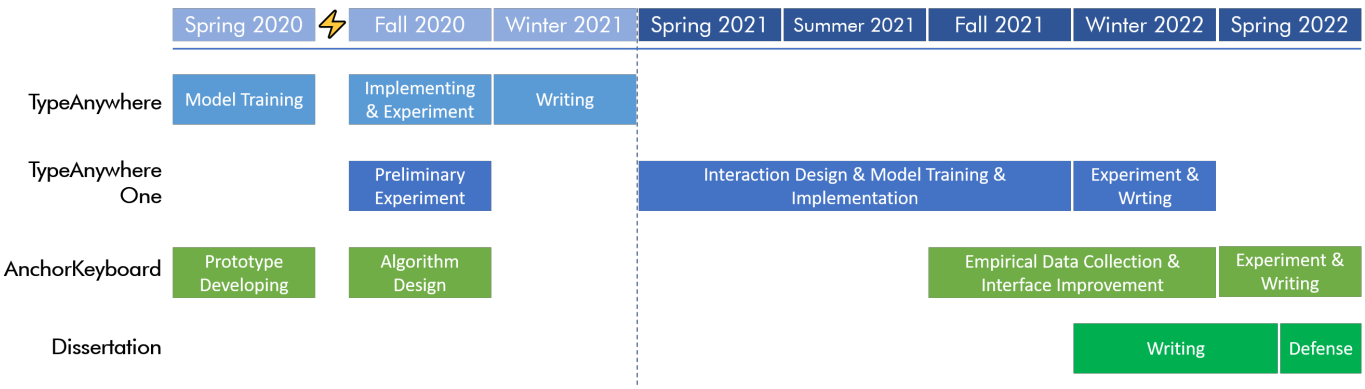

Figure 7.1: Proposed schedule in phases. Part of the proposed work was conducted from spring 2020 to winter 2021

Acknowledgement

I would like to thank many people, who helped me learn to stand up, walk and carry on through the academic ups and downs: to my advisor, Jacob O. Wobbrock, who introduced me into the text entry world with his simple yet elegant ideas, which influenced my way of doing research; to my committee members: Alexis Hiniker, who has supported me in pursuing alternative research possibilities and almost became my second mentor; Shumin Zhai, who took care of me during the unusual internship and offered thoughtful insights on many of my projects; Leah Findlater and James Fogarty, whose work motivated me to look for research problems in accessibility.

Part of my work is supported by grants from Baidu and Google. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect of any supporter.

I would also like to thank Abdullah X. Ali, Stephanie Ballard, Adam Berenzweig, Xiaojun Bi, Jacob Burke, Wenzhe Cui, Erin Beneteau, Rachel L. Franz, Liang He, Chris Holstrom, Yvette Iribe, Scott Jenson, Fanwen Ji, Alex Kale, Qisheng Li, Suning Li, Toby Jiajun Li, Yuying Liu, Alex Mariakakis, Martez E. Mott, Dichen Qian, Paul V. Roby, Luke Rodriguez, Milly Romeijn-Stout, Anne Spencer Ross, Ather Sharif, Jing Su, Qin Wang, Ruolin Wang, Xia Wang, He Wen, Benjemin Xie, Xuhai Xu, Jackie Junrui Yang, Dongbo Zhai, Ruohan Zhan, Xiaoyi Zhang, Yang Zhang, Yiming Zhang and Mingyuan Zhong.

Bibliography

[1] O. Alsharif, Tom Ouyang, F. Beaufays, S. Zhai, Thomas Breuel, and Johan Schalkwyk. 2015. Long short term memory neural network for keyboard gesture decoding. *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (2015), 2076–2080.

[2] Saleema Amershi, Dan Weld, Mihaela Vorvoreanu, Adam Fourney, Besmira Nushi, Penny Collisson, Jina Suh, Shamsi Iqbal, Paul N. Bennett, Kori Inkpen, Jaime Teevan, Ruth Kikin-Gil, and Eric Horvitz. 2019. Guidelines for Human-AI Interaction. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems (CHI '19)*. Association for Computing Machinery, New York, NY, USA, 1–13.

[3] Dario Amodei, Sundaram Ananthanarayanan, Rishita Anubhai, Jingliang Bai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Qiang Cheng, Guoliang Chen, Jie Chen, Jingdong Chen, Zhijie Chen, Mike Chrzanowski, Adam Coates, Greg Diamos, Ke Ding, Niandong Du, Erich Elsen, Jesse Engel, Weiwei Fang, Linxi Fan, Christopher Fougner, Liang Gao, Caixia Gong, Awni Hannun, Tony Han, Lappi Vaino Johannes, Bing Jiang, Cai Ju, Billy Jun, Patrick LeGresley, Libby Lin, Junjie Liu, Yang Liu, Weigao Li, Xiangang Li, Dongpeng Ma, Sharan Narang, Andrew Ng, Sherjil Ozair, Yiping Peng, Ryan Prenger, Sheng Qian, Zongfeng Quan, Jonathan Raiman, Vinay Rao, Sanjeev Satheesh, David Seetapun, Shubho Sengupta, Kavya Srinet, Anuroop Sriram, Haiyuan Tang, Liliang Tang, Chong Wang, Jidong Wang, Kaifu Wang, Yi Wang, Zhijian Wang, Zhiqian Wang, Shuang Wu, Likai Wei, Bo Xiao, Wen Xie, Yan Xie, Dani Yogatama, Bin Yuan, Jun Zhan, and Zhenyao Zhu. 2016. Deep Speech 2: End-to-End Speech Recognition in English and Mandarin. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48 (ICML'16)*. JMLR.org, 173–182.

[4] Ahmed Sabbir Arif, Sunjun Kim, Wolfgang Stuerzlinger, Geehyuk Lee, and Ali Mazalek. 2016. *Evaluation of a Smart-Restorable Backspace Technique to Facilitate Text Entry Error Correction.* Association for Computing Machinery, New York, NY, USA, 5151–5162.

[5] Ahmed Sabbir Arif and Wolfgang Stuerzlinger. 2010. *Predicting the Cost of Error Correction in Character-Based Text Entry Technologies.* Association for Computing Machinery, New York, NY, USA, 5–14.

[6] Ahmed Sabbir Arif and Wolfgang Stuerzlinger. 2013. Pseudo-Pressure Detection and Its Use in Predictive Text Entry on Touchscreens. In *Proceedings of the 25th Australian Computer-Human Interaction Conference: Augmentation, Application, Innovation, Collaboration (OzCHI '13)*. Association for Computing Machinery, New York, NY, USA, 383–392.

[7] Dvorak August. U.S. Patent US2040248A, May. 1936. Typewriter keyboard.

[8] Shiri Azenkot, Jacob O. Wobbrock, Sanjana Prasain, and Richard E. Ladner. 2012. Input Finger Detection for Nonvisual Touch Screen Text Entry in Perkinput. In *Proceedings of Graphics Interface 2012 (GI '12)*. Canadian Information Processing Society, CAN, 121–129.

[9] Shiri Azenkot and Shumin Zhai. 2012. Touch Behavior with Different Postures on Soft Smartphone Keyboards. In *Proceedings of the 14th International Conference on Human-Computer Interaction with Mobile Devices and Services (MobileHCI '12)*. Association for Computing Machinery, New York, NY, USA, 251–260.

[10] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2016. Neural Machine Translation by Jointly Learning to Align and Translate.

[11] Nikola Banovic, Ticha Sethapakdi, Yasasvi Hari, Anind K. Dey, and Jennifer Mankoff. 2019. The Limits of Expert Text Entry Speed on Mobile Keyboards with Autocorrect. In *Proceedings of the 21st International Conference on Human-Computer Interaction with Mobile Devices and Services (MobileHCI '19)*. Association for Computing Machinery, New York, NY, USA, Article 15, 12 pages.

[12] Francesco Barbieri, Miguel Ballesteros, Francesco Ronzano, and Horacio Saggion. 2018. Multimodal Emoji Prediction.

[13] Michel Beaudouin-Lafon. 2004. Designing Interaction, Not Interfaces. In *Proceedings of the Working Conference on Advanced Visual Interfaces (AVI '04)*. Association for Computing Machinery, New York, NY, USA, 15–22.

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

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

[16] 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)*. Association for Computing Machinery, New York, NY, USA, 49–58.

[17] 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.

[18] Stuart K. Card, Allen Newell, and Thomas P. Moran. 1983. *The Psychology of Human-Computer Interaction*. L. Erlbaum Associates Inc., USA.

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

[20] Mia Xu Chen, Benjamin N Lee, Gagan Bansal, Yuan Cao, Shuyuan Zhang, Justin Lu, Jackie Tsay, Yinan Wang, Andrew M. Dai, Zhifeng Chen, Timothy Sohn, and Yonghui Wu. 2019. Gmail Smart Compose: Real-Time Assisted Writing.

[21] 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 (UIST '14)*. Association for Computing Machinery, New York, NY, USA, 615–620.

[22] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation.

[23] Henriette Cramer, Paloma de Juan, and Joel Tetreault. 2016. Sender-Intended Functions of Emojis in US Messaging. In *Proceedings of the 18th International Conference on Human-Computer Interaction with Mobile Devices and Services (MobileHCI '16)*. Association for Computing Machinery, New York, NY, USA, 504–509.

[24] E. R. F. W. Crossman. 1957. The Speed and Accuracy of Simple Hand Movements. *The Nature and Acquisition of Industrial Skills.* (1957).

[25] 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 (UIST '20)*. Association for Computing Machinery, New York, NY, USA, 487–499.

[26] 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`

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

[28] 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.

[29] Ben Eisner, Tim Rocktäschel, Isabelle Augenstein, Matko Bošnjak, and Sebastian Riedel. 2016. emoji2vec: Learning Emoji Representations from their Description. In *Proceedings of The Fourth International Workshop on Natural Language Processing for Social Media*. Association for Computational Linguistics, Austin, TX, USA, 48–54.

[30] Anna Maria Feit and Antti Oulasvirta. 2014. PianoText: Redesigning the Piano Keyboard for Text Entry. In *Proceedings of the 2014 Conference on Designing Interactive Systems (DIS '14)*. Association for Computing Machinery, New York, NY, USA, 1045–1054.

[31] Bjarke Felbo, Alan Mislove, Anders Søgaard, Iyad Rahwan, and Sune Lehmann. 2017. Using millions of emoji occurrences to learn any-domain representations for detecting sentiment, emotion and sarcasm. *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing* (2017).

[32] Leah Findlater, Ben Lee, and Jacob Wobbrock. 2012. Beyond QWERTY: Augmenting Touch Screen Keyboards with Multi-Touch Gestures for Non-Alphanumeric Input. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '12)*. Association for Computing Machinery, New York, NY, USA, 2679–2682.

[33] 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 (CHI '12)*. Association for Computing Machinery, New York, NY, USA, 815–824.

[34] 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 (CHI '11)*. Association for Computing Machinery, New York, NY, USA, 2453–2462.

[35] Paul. M. Fitts. 1954. The information capacity of the human motor system in controlling the amplitude of movement. *Journal of Experimental PSychology* 74 (1954), 381–391.

[36] George Fitzmaurice, Azam Khan, Robert Pieké, Bill Buxton, and Gordon Kurtenbach. 2003. Tracking Menus. In *Proceedings of the 16th Annual ACM Symposium on User Interface Software and Technology (UIST '03)*. Association for Computing Machinery, New York, NY, USA, 71–79.

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

[38] Vittorio Fuccella, Poika Isokoski, and Benoit Martin. 2013. *Gestures and Widgets: Performance in Text Editing on Multi-Touch Capable Mobile Devices*. Association for Computing Machinery, New York, NY, USA, 2785–2794.

[39] Vittorio Fuccella and Benoît Martin. 2017. TouchTap: A Gestural Technique to Edit Text on Multi-Touch Capable Mobile Devices *(CHItaly '17)*. Association for Computing Machinery, New York, NY, USA, Article 21, 6 pages.

[40] Mayank Goel, Leah Findlater, and Jacob Wobbrock. 2012. WalkType: Using Accelerometer Data to Accomodate Situational Impairments in Mobile Touch Screen Text Entry. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '12)*. Association for Computing Machinery, New York, NY, USA, 2687–2696.

[41] Jun Gong and Peter Tarasewich. 2006. A New Error Metric for Text Entry Method Evaluation. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '06)*. Association for Computing Machinery, New York, NY, USA, 471–474.

[42] 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 (CHI '18)*. Association for Computing Machinery, New York, NY, USA, 1–14.

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

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

[45] 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 (CHI '16)*. Association for Computing Machinery, New York, NY, USA, 3817–3821.

[46] Tovi Grossman, Ken Hinckley, Patrick Baudisch, Maneesh Agrawala, and Ravin Balakrishnan. 2006. Hover Widgets: Using the Tracking State to Extend the Capabilities of Pen-Operated Devices. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '06)*. Association for Computing Machinery, New York, NY, USA, 861–870.

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

[48] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep Residual Learning for Image Recognition. *arXiv preprint arXiv:1512.03385* (2015).

[49] Ken Hinckley, Patrick Baudisch, Gonzalo Ramos, and Francois Guimbretiere. 2005. Design and Analysis of Delimiters for Selection-Action Pen Gesture Phrases in Scriboli. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '05)*. Association for Computing Machinery, New York, NY, USA, 451–460.

[50] 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 (CHI '15)*. Association for Computing Machinery, New York, NY, USA, 1233–1236.

[51] Eric Horvitz. 1999. Principles of Mixed-Initiative User Interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '99)*. Association for Computing Machinery, New York, NY, USA, 159–166.

[52] WHIRLSCAPE Inc. 2016. Dango - Your Emoji Assistant. `https://getdango.com`

[53] Minal Jain, Sarita Seshagiri, and Simran Chopra. 2016. How Do I Communicate My Emotions on SNS and IMs?. In *Proceedings of the 18th International Conference on Human-Computer Interaction with Mobile Devices and Services Adjunct (MobileHCI '16)*. Association for Computing Machinery, New York, NY, USA, 767–774.

[54] Shaun K. Kane, Meredith Ringel Morris, and Jacob O. Wobbrock. 2013. Touchplates: Low-Cost Tactile Overlays for Visually Impaired Touch Screen Users. In *Proceedings of the 15th International ACM SIGACCESS Conference on Computers and Accessibility (ASSETS '13)*. Association for Computing Machinery, New York, NY, USA, Article 22, 8 pages.

[55] Anjuli Kannan, Karol Kurach, Sujith Ravi, Tobias Kaufman, Balint Miklos, Greg Corrado, Andrew Tomkins, Laszlo Lukacs, Marina Ganea, Peter Young, and Vivek Ramavajjala. 2016. Smart Reply: Automated Response Suggestion for Email. In *Proceedings of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD) (2016)*.

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

[57] Yoon Kim, Yacine Jernite, David Sontag, and Alexander M. Rush. 2015. Character-Aware Neural Language Models.

[58] D. Klakow and Jochen Peters. 2002. Testing the correlation of word error rate and perplexity. *Speech Commun.* 38 (2002), 19–28.

[59] Per-Ola Kristensson and Shumin Zhai. 2004. SHARK2: 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)*. Association for Computing Machinery, New York, NY, USA, 43–52.

[60] 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 (CHI '15)*. Association for Computing Machinery, New York, NY, USA, 669–678.

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

[62] James R. Lewis. 1999. Input Rates and User Preference for Three Small-Screen Input Methods: Standard Keyboard, Predictive Keyboard, and Handwriting. *Proceedings of the Human Factors and Ergonomics Society Annual Meeting* 43, 5 (1999), 425–428.

[63] Yutang Lin. U.S. Patent US2613795A, Oct. 1952. Chinese typewriter.

[64] 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 (CHI '04)*. Association for Computing Machinery, New York, NY, USA, 671–678.

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

[66] 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 (CHI EA '03)*. Association for Computing Machinery, New York, NY, USA, 754–755.

[67] I. Scott MacKenzie and Kumiko Tanaka-Ishii. 2007. *Text Entry Systems: Mobility, Accessibility, Universality.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[68] 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)*. Association for Computing Machinery, New York, NY, USA, 25–31.

[69] Edgar Matias, I. Scott MacKenzie, and William Buxton. 1996. One-Handed Touch Typing on a QWERTY Keyboard. *Hum.-Comput. Interact.* 11, 1 (March 1996), 1–27.

[70] Eugene W. Myers. 1986. An O(ND) Difference Algorithm and Its Variations. *Algorithmica* 1 (1986), 251–266.

[71] Hwee Tou Ng, Siew Mei Wu, Yuanbin Wu, Christian Hadiwinoto, and Joel Tetreault. 2013. The CoNLL-2013 Shared Task on Grammatical Error Correction. In *Proceedings of the Seventeenth Conference on Computational Natural Language Learning: Shared Task*. Association for Computational Linguistics, Sofia, Bulgaria, 1–12.

[72] Donald A. Norman. 1994. How Might People Interact with Agents. *Commun. ACM* 37, 7 (July 1994), 68–71.

[73] Nuance. Nuance - Conversational AI for Healthcare and Customer Engagement. `https://www.nuance.com/index.html`

[74] 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 (CHI '13)*. Association for Computing Machinery, New York, NY, USA, 2799–2802.

[75] Tom Ouyang, David Rybach, Françoise Beaufays, and Michael Riley. 2017. Mobile Keyboard Input Decoding with Finite-State Transducers.

[76] Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. GloVe: Global Vectors for Word Representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, Doha, Qatar, 1532–1543.

[77] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language Models are Unsupervised Multitask Learners. (2019).

[78] Quentin Roy, Futian Zhang, and Daniel Vogel. 2019. Automation Accuracy Is Good, but High Controllability May Be Better. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems (CHI '19)*. Association for Computing Machinery, New York, NY, USA, 1–8.

[79] Alexander M. Rush, Sumit Chopra, and Jason Weston. 2015. A Neural Attention Model for Abstractive Sentence Summarization.

[80] Claude E. Shannon. 1948. A mathematical theory of communication. *Bell System Technical Journal* 27 (1948), 379–423.

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

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

[83] Ben Shneiderman. 2020. Human-Centered Artificial Intelligence: Reliable, Safe & Trustworthy. *International Journal of Human–Computer Interaction* 36, 6 (2020), 495–504.

[84] Miika Silfverberg, I. Scott MacKenzie, and Panu Korhonen. 2000. Predicting Text Entry Speed on Mobile Phones. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '00)*. Association for Computing Machinery, New York, NY, USA, 9–16.

[85] Shyamli Sindhwani, Christof Lutteroth, and Gerald Weber. 2019. ReType: Quick Text Editing with Keyboard and Gaze. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems (CHI '19)*. Association for Computing Machinery, New York, NY, USA, 1–13.

[86] Brian A. Smith, Xiaojun Bi, and Shumin Zhai. 2015. Optimizing Touchscreen Keyboards for Gesture Typing. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems (CHI '15)*. Association for Computing Machinery, New York, NY, USA, 3365–3374.

[87] R. William Soukoreff and I. Scott MacKenzie. 2001. Measuring Errors in Text Entry Tasks: An Application of the Levenshtein String Distance Statistic. In *CHI '01 Extended Abstracts on Human Factors in Computing Systems (CHI EA '01)*. Association for Computing Machinery, New York, NY, USA, 319–320.

[88] 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 (CHI '03)*. Association for Computing Machinery, New York, NY, USA, 113–120.

[89] Caleb Southern, James Clawson, Brian Frey, Gregory Abowd, and Mario Romero. 2012. An Evaluation of BrailleTouch: Mobile Touchscreen Text Entry for the Visually Impaired. In *Proceedings of the 14th International Conference on Human-Computer Interaction with Mobile Devices and Services (MobileHCI '12)*. Association for Computing Machinery, New York, NY, USA, 317–326.

[90] 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 (CHI '15)*. Association for Computing Machinery, New York, NY, USA, 3643–3652.

[91] Larry Tesler. 2012. A Personal History of Modeless Text Editing and Cut/Copy-Paste. *Interactions* 19, 4 (July 2012), 70–75.

[92] Garreth W. Tigwell, Benjamin M. Gorman, and Rachel Menzies. 2020. Emoji Accessibility for Visually Impaired People. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems (CHI '20)*. Association for Computing Machinery, New York, NY, USA, 1–14.

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

[94] Dan Venolia and Forrest Neiberg. 1994. T-Cube: A Fast, Self-Disclosing Pen-Based Alphabet. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '94)*. Association for Computing Machinery, New York, NY, USA, 265–270.

[95] Veroniiiica. 2018. How Do People with Vision Impairments Use Emoji? `https://www.perkinselearning.org/technology/blog/how-do-people-vision-impairments-use-emoji`

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

[97] 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)*. Association for Computing Machinery, New York, NY, USA, 1–14.

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

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

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

[101] Will Walmsley. 2015. Exploring Emoji: The Quest for the Perfect Emoticon. `http://minuum.com/exploring-emoji-the-quest-for-the-perfect-emoticon/`

[102] David J. Ward, Alan F. Blackwell, and David J. C. MacKay. 2000. Dasher—a Data Entry Interface Using Continuous Gestures and Language Models. In *Proceedings of the 13th Annual ACM Symposium on User Interface Software and Technology (UIST '00)*. Association for Computing Machinery, New York, NY, USA, 129–137.

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

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

[105] A. Welford. 1968. Fundamentals of skill / A.T. Welford. Methuen, London, England, 137–160.

[106] L. West. 1967. Vision and kinesthesis in the acquisition of typewriting skill. *The Journal of applied psychology* 51 2 (1967), 161–166.

[107] Sarah Wiseman and Sandy J. J. Gould. 2018. *Repurposing Emoji for Personalised Communication: Why 🍕 Means "I Love You"*. Association for Computing Machinery, New York, NY, USA, 1–10.

[108] 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.

[109] Jacob O. Wobbrock, Brad A. Myers, and John A. Kembel. 2003. EdgeWrite: A Stylus-Based Text Entry Method Designed for High Accuracy and Stability of Motion. In *Proceedings of the 16th Annual ACM Symposium on User Interface Software and Technology (UIST '03)*. Association for Computing Machinery, New York, NY, USA, 61–70.

[110] 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).

[111] 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.

[112] Ziang Xie, Anand Avati, Naveen Arivazhagan, Dan Jurafsky, and Andrew Y. Ng. 2016. Neural Language Correction with Character-Based Attention.

[113] 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 (CHI '20)*. Association for Computing Machinery, New York, NY, USA, 1–13.

[114] 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 (UIST '19)*. Association for Computing Machinery, New York, NY, USA, 883–899.

[115] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. 2020. XLNet: Generalized Autoregressive Pretraining for Language Understanding.

[116] 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 (CHI '20)*. Association for Computing Machinery, New York, NY, USA, 1–13.

[117] 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 (CHI '17)*. Association for Computing Machinery, New York, NY, USA, 705–715.

[118] 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 amp; Technology (UIST '15)*. Association for Computing Machinery, New York, NY, USA, 539–548.

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

[120] Adams Wei Yu, David Dohan, Minh-Thang Luong, Rui Zhao, Kai Chen, Mohammad Norouzi, and Quoc V. Le. 2018a. QANet: Combining Local Convolution with Global Self-Attention for Reading Comprehension.

[121] Chun Yu, Ke Sun, Mingyuan Zhong, Xincheng Li, Peijun Zhao, and Yuanchun Shi. 2016. *One-Dimensional Handwriting: Inputting Letters and Words on Smart Glasses*. Association for Computing Machinery, New York, NY, USA, 71–82.

[122] Difeng Yu, K. Fan, H. Zhang, Diego Monteiro, Wenge Xu, and H. Liang. 2018b. PizzaText: Text Entry for Virtual Reality Systems Using Dual Thumbsticks. *IEEE Transactions on Visualization and Computer Graphics* 24 (2018), 2927–2935.

[123] Torsten Zesch. 2012. Measuring Contextual Fitness Using Error Contexts Extracted from the Wikipedia Revision History. In *Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics*. Association for Computational Linguistics, Avignon, France, 529–538.

[124] Mingrui Ray Zhang, Alex Mariakakis, Jacob Burke, and Jacob O. Wobbrock. 2021a. Acomparative study of lexical and semantic emoji suggestion systems. In *Proceedings of iConference 2021*. Springer, Switzerland.

[125] Mingrui Ray Zhang, Ruolin Wang, Xuhai Xu, Qisheng Li, Ather Sharif, and Jacob O. Wobbrock. 2021b. Voicemoji: Emoji entry using voice for visually impaired people. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems (CHI '21)*. ACM, New York, NY, USA.

[126] Mingrui Ray Zhang, He Wen, and Jacob O. Wobbrock. 2019a. 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 (UIST '19)*. Association for Computing Machinery, New York, NY, USA, 843–855.

[127] 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 (UIST '19)*. ACM, New York, NY, USA, 831–842.

[128] Mingrui Ray Zhang and Jacob O. Wobbrock. 2020. Gedit: Keyboard Gestures for Mobile Text Editing. In *Proceedings of Graphics Interface 2020 (GI 2020)*. Canadian Human-Computer Communications Society / Société canadienne du dialogue humain-machine, 470 – 473.

[129] 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)*. ACM, New York, NY, USA.

[130] Mingrui Ray Zhang, Shumin Zhai, and Jacob O. Wobbrock. 2019b. 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*. ACM, New York, NY, USA.

[131] Xiang Zhang, Junbo Zhao, and Yann LeCun. 2016. Character-level Convolutional Networks for Text Classification.

[132] 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 (CHI '18)*. Association for Computing Machinery, New York, NY, USA, 1–13.