Towards More Intelligent, Intuitive, and Inclusive Communication with Computers in Text and Images

Mingrui Zhang

A dissertation submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington

2022

Reading Committee:

Jacob O. Wobbrock, Chair

Alexis Hiniker

Shumin Zhai

Leah Findlater

Program Authorized to Offer Degree:

Information Science

University of Washington

Abstract

Towards More Intelligent, Intuitive, and Inclusive Communication with Computers in Text and Images

Mingrui Zhang

Chair of the Supervisory Committee:

Professor Jacob O. Wobbrock

Information School

Communication is fundamental to human experience and our interaction with computers. The efficiency of human communication relies largely on the quality of the medium. Modern computing devices offer mediums such as keyboards to interact with information, yet they are still primitive (e.g., one has to press every character of a phrase) and often fail to support different user needs (e.g., limited emoji support for visually-impaired users). Thus an important question is, how to make the keyboard understand our languages like a human being, so that communication with computers can be intelligent, intuitive and inclusive?

In this dissertation, I demonstrate how to design, build and evaluate communication interactions using the power of artificial intelligence. My dissertation addresses the above question in three different strands of work: (1) Intelligent text entry and editing interactions that understand the user intention; (2) Assistive systems that help blind or low vision users to communicate with pictorial information such as emojis and GIFs; (3) Models and metrics that evaluate intelligent input systems on their performance and impact on human behaviors. Together, the work demonstrates the thesis statement: *Artificial intelligence can enable and improve advanced text production and accessible interactions with pictures; in addition, new metrics for text entry enable the evaluation of advanced capabilities.*

TABLE OF CONTENTS

LIST OF FIGURES

Figure Number Page

LIST OF TABLES

ACKNOWLEDGMENTS

Eight years ago, I was riding my bike with three friends from college in the mountains. We were trying to finish the famous line from Sichuan to Tibet - a 2000-km long road, which is 3000 meters above sea level on average. The riding was dangerous, tough, and extremely tiring — I even fell off the bike and got my wrist fractured. I was imagining about how excited and proud I would be on the finish day. But when I finally made it to the destination city Lhasa, I just felt exhausted and relieved.

It is also hard for me to have any strong emotion at the end of this academic journey - there was just too much. It has brought me high hopes each time I started drafting a new project plan, punched me in the face with paper rejections[1], thrown me into the valley of shit[2] where I doubt the value of my research and ability, and exposed me to unfamiliar but fascinating areas that ignited my curiosity. As a Ph.D., my contribution to the world so far is the several papers I've published, which may just expand the boundary of human knowledge a tiny bit; but the five years have contributed enormously to the shape of my life. I am more than lucky to have all the people traveling with me - my advisor, mentors, collaborators, schoolmates, friends, and family - it is you who shaped the journey, and we together accomplished this achievement!

I will be forever indebted to my dear advisor, Jacob O. Wobbrock. I still remember the first time we met, outside Mary Gates Hall, you walked me into the lab but suddenly shut the door before I could enter, and smiled through the window saying "come inside" — the joke instantly relieved my stress. I remember all the conversations happening in your office,

[1] My first project got seven rejections before it was published!

[2] `https://thesiswhisperer.com/2012/05/08/the-valley-of-shit/`

where we argued about our opinions and passionately drew stuff on the whiteboard, where many "a-ha" moments happened. The freedom you granted me to explore unfamiliar areas, the kind words you sent when I lost my confidence, the wise advice you provided when I was either clueless or impetuous - all of those made me an "independent researcher", and you should be proud of it! In our first meeting, you told me, "as one of my Ph.D. students, you probably will see your advisor as an amazing guy in your first two years. Then in the mid years you might think that I am a good collaborator but what I've achieved is not that unrealistic. As you are near the end, you will just view me as a normal guy who is not helpful anymore." Jake, to me you're never just "a normal guy": you are a great mentor and friend; and I am always learning from you.

My "unofficial" advisor, Alexis Hiniker, is just an amazing human being. Words cannot express how I am grateful for your help - whether it's financial, emotional, or academic. Since you've invited me to join our first project [22], I was exposed to many new topics which I would never have chance to explore without you. You made me appreciate the value of qualitative research and user-centered methods, whereas before I was only keen about algorithmic and technical aspects. You always came up with clever workarounds when I almost gave up due to technical limitations, and always had the curiosity to ask "what if" questions to inspire us. I feel joyful and grateful working with a mentor who is always smiling, supportive, and has a heart of gold.

I knew Shumin Zhai even earlier than Jake - I sent an email asking for internship opportunities in my junior year. We've kept in contact since then, but I've never dreamed about collaborating or even working with Shumin. To me, Shumin is a role model as a research scientist who shares a similar cultural background with me; who is always humble, diligent, and trying to provide opportunities for younger generations. Over the years we've had many successful collaborations and built a great relationship. As I am going to start my career, I will devote my time to supporting young researchers and giving back to the community, just as Shumin has taught me.

Thank you, James Fogarty and Leah Findlater to be on my committee! You've shown great support attending my yearly reviews and providing useful feedback. It's also a great experience working with James for the DUB community!

To all the iSchool staff and faculties I've interacted with - Thank you for being there helping me navigate through administrative matters. I appreciate the timely help from Wendie Phillips related to my meta Ph.D. questions including VISA, internship, graduation, and all others - she is a heroine! Thank you, Marie Stucke and Ai Nguyen for handling the HR-related stuff. Thank you Victor Aque for organizing all the student activities - you are a friend of i17! Thank you Carmen Parisi for keeping our workspace organized, and Mitchel Ayer for working me around the graduation steps!

Showing love to my Seattle Ph.D. group - Qisheng Li, Yiming Zhang, Qin Wang, Yuying Liu and Yi Chu! There are countless times we hung out together just for fun, did unplanned road trips to relive stress, had deep discussions about science and life, or just stayed there for each other when needed - you made my journey no longer lonely; to my cat Belle for listening to my poorly skilled piano pieces; to Dichen Qian for telling me "everything will be fine" in the ER; to Jing Su, Fanwen Ji, and Luyu Liu, who taught me how to grow up.

To my cohort (Mina Tari, Erin Beneteau, Alex Kale, Milly Romeijn-Stout, Yvette Iribe, Stephanie Ballard, Rachel Franz, Chris Holstrom, Luke Rodriguez, including Benjamin Xie) - i17 rocks! I cannot imagine how I will survive my first year without you! The retreats, happy hours, birthday cards, flowers, gift cards when I was in the hospital, general exam packages, and zoom parties - they truly showed how supportive and caring a cohort can be.

To my ACE lab members (Martez E. Mott, Alex Mariakakis, Anne Spencer Ross, Abdullah X. Ali, Ather Sharif, Lisa Elkin, Mingyuan Zhong, Junhan Kong, Zhuohao Zhao) - thank you! I have shared joyful time talking gossip and funny stories with you, learned useful tips for surviving graduate school, and some of us even collaborated on projects [269, 270, 276]. It is an honor working with everyone!

Finally, to my family - thank you my mother Xia Wang, my father Dongbo Zhai, and my sister Jiaxin Zhai, for believing in me, accepting who I am, pulling me out of the darkest time after the car accident, and answering my call even if it was late night in China. Although I haven't been back home for years, I feel connected with you every moment. Family is my deepest connection with the world.

Thank you, Seattle! You've left me so many precious memories and even though you treated me poorly during the winter with the dark-and-moody-and-rainy weather, you're still my favorite city - I appreciate how welcoming, vibrant, and open you are.

Many people helped and accompanied me through the five years. I would like to write down their names here: Abdullah X. Ali, Adam Berenzweig, Xiaojun Bi, Liang He, Suning Li, Tianshi Li, Toby Jiajun Li, Yifang Liu, Yuqun Liu, Paul V. Roby, Ather Sharif, Jacob Burke, Sida Gao, Jun Gong, Huy Viet Le, Haomin Long, Tim Paek, Hang Qi, Miao Ren, Ruolin Wang, He Wen, Xuhai Xu, Junrui Yang, Ruohan Zhan, Xiaoyi Zhang, and Yang Zhang. Thank you, I'm here because of each of you.

Now, without further ado, please enjoy the fruits of my Ph.D. research. :)

To my parents, and my biological father —

for raising, supporting and inspiring me.

Chapter 1

INTRODUCTION

Communication is a fundamental element of human civilization; language is a fundamental element of communication. We think, write and talk through language and text every day. As a result, we humans have made numerous inventions to help express our minds efficiently. Figure 1.1 shows a clay board from more than five thousand years ago, carved with one of the earliest human languages: Cuneiform[1]. Since then, the way we communicate has evolved into various forms: papermaking was invented in China around 25 CE, then typography in 1000 CE, and the first commercially successful typewriter "Sholes and Glidden" came out in 1874, which made the keyboard become the standard input interface for computers — even on touch screens, most people are stilling using the QWERTY keyboard as their main input method.

Compared with the history of writing and printing, the history of typing has been only more than a hundred years, and there are still a lot of opportunities to improve the interaction. Researchers have developed advanced features such as auto-correction and word prediction to ease the effort of typing, and designed sophisticated coding mechanisms [134], layouts [142] and gestures [121] to boost the performance. Yet, keyboards constantly fall short of our expectations: auto-correction makes wrong suggestions [127]; word prediction is rarely used by fast typists [116]; layouts and coding mechanisms (such as chorded keyboards) that are theoretically faster than the QWERTY touch keyboard fail to become popular among users [27]. Communicating with digital information can also be challenging for people with disabilities - we now have richer options for online conversations (such as images and animations) while keyboards still focus mainly on text-based communications.

---

[1] https://en.wikipedia.org/wiki/Cuneiform

Figure 1.1: A salary receipt written in Cuneiform on a clay board. The text means *The salary is composed of cash and beer. If the payer is short on cash, then beer can be used for the payment.*

On the other hand, when talking with each other, we can express our minds freely and efficiently, despite the fact that people use natural language for conversations, which contains more ambiguity than keyboard commands. The difference is that human do not focus on a single piece of text - we tend to understand an utterance based on its context: what is said before and after. Based on the context (e.g., who we are talking to), we will also adjust our linguistic style and the content to make the expression understandable to others. Context is thus key for an efficient conversation.

Inspired by the human-to-human interaction, this dissertation aims to address the shortcomings of current input interactions by making the communication more intelligent, intuitive and inclusive, allowing the user to express their minds seamlessly. Specifically, in my work: 1) I develop intelligent text input interactions to understand the user's intention during then entry and editing process, supporting efficient text production; 2) I apply user-centered de-

sign methods to build accessibility systems, helping blind or low vision (BLV) users to interact with pictorial information (such as emojis and animated GIFs) using natural language; 3) I derive models and metrics to evaluate those intelligent communication systems on their performances and impact on human behaviors.

My research vision is that communication with computers should be intelligent, intuitive and inclusive. Taken together, the work presented in this dissertation shall demonstrate the following thesis:

> *Artificial intelligence can enable and improve advanced text production and accessible interactions with pictures; in addition, new metrics for text entry enable the evaluation of advanced capabilities.*

1.1 Problems of Current Communication Interactions

Whether we're on a desktop or mobile device, typing on keyboards remains the most common way of expressing our thoughts. However, there are often times our keyboards make mistakes and barely understand us [171]. For one thing, all the mechanical commands required to operate the text (such as pressing buttons, moving cursors) act as extra layers between our mind and the outputted text, adding burden and no real benefit; for another, the typing interaction is deterministic (e.g., to type a character one has to press a certain key), because of which the keyboard interface provides little flexibility for people with different abilities. In order to express one's mind smoothly to a computer, one has to memorize the layout and the coding scheme of a keyboard interface, making artificial movements. What are the problems, and how can we solve them?

One problem is **the keyboard is not fully utilizing the context**. Here the context means the language context around a piece of text, such as the surrounding words and phrases. Keyboards are command-based: they output the exact literal resulting from the user actions, without trying to understand what is being typed. Even with advanced features such as auto-correction, it only improves the text quality based on their spellings and frequencies, instead of their semantic meanings [82]. For example, when someone types *Today I learned a neuttal*, the keyboard would correct the word to *neutral* as it is similar to the typo; but

if the user continues to type *network model*, then probably the text should be corrected to *Today I learned a* ***neural*** *network model* instead of *Today I learned a* ***neutral*** *network model*. However, as the keyboard just corrects the typo as soon as the user finishes the word, it does not consider the whole input context around the typo *neuttal*, thus is not able to make the right correction which is semantically more reasonable.

The other problem lies in the interaction. Firstly, **current communication interactions do not tolerate ambiguity**. Human communication is inherently ambiguous: we have different expressions to describe the same meaning, and we use natural languages rather than a fixed set of commands to talk with each other. For example, to talk about the emoji 😂, one may say "the laughing crying emoji" or "the LOL emoji" instead of its official name "face with tears of joy emoji". However, when we input information to computers, there is no ambiguity allowed - we have to execute the exact set of commands, and small deviations can lead to different results.

Secondly, **current communication interactions are not adaptive to users' abilities**. When talking to people with different backgrounds, we tend to adjust the linguistic style and the content to let them understand easily; when communicating with a computer, everyone needs to perform the same operations and the information will be delivered to each user in the same way, regardless of their abilities. The lack of adaptability of the interface can create significant barriers for people with disabilities when communicating with the device, especially for non-textual information such as emojis [215] and animated GIFs [75].

1.2 Addressing the Problem of Context by Artificial Intelligence

To utilize the language context, one promising approach is to combine the keyboard interface with the power of artificial intelligence (AI). The advancement of machine learning (ML) algorithms, especially deep learning algorithms, has created language models able to incorporate larger context and achieve high performance on tasks including summary, question-answering and conversation [31, 183, 261]. Using machine learning algorithms, I design and implement intelligent text production interactions and systems that operate beyond the character level manipulation. The series of work in text production has covered

the full spectrum of the interaction, including **text entry** and **text editing**.

PhraseFlow [273] is a keyboard that accelerates the entry interaction via incorporating a larger language context in the decoding process, allowing the user to type in a more error-tolerant and flexible way. Unlike traditional word-level decoding methods, the keyboard decodes the text at phrase-level and can correct previous text based on the subsequent input sequences. In this way, it can make corrections from *Today I learned a* ***neutral*** *network model* to *Today I learned a* ***neural*** *network model*, or from *commu ication* to *communication* — both examples require the text before and after the errors to find a semantically meaningful correction. Current word-level auto-corrections will commit the correction after the word is finished and space is typed. *PhraseFlow* voids the immediate commission after the space press; instead, it commits the correction in word chunks, i.e., correcting the text for every n words entered. To achieve the functionality, I adjust the finite-state transducer (FST) [166] decoder of Gboard [83] so that it could incorporate a larger language context. .

Type, Then Correct [271] is a novel text correction concept for mobile touch devices. Current text correction processes on mobile touch devices are laborious: users either extensively use backspace, or navigate the cursor to the error position, make a correction, and navigate back, usually by employing multiple taps or drags over small targets. However, in a conversation with friends, if we said something wrong, we can just re-speak the correct utterance without even mentioning the mistake (e.g., "Some coffee please. Oh sorry, tea"), because the other person understands the context of the correction. *Type, Then Correct* aimes to resemble this experience on touch screens: to revise an error, the user just types the correction first, and then applies the correction with a set of interactions to the error directly. There is no need to perform the cursor-positioning and error-deleting actions. Backed by neural networks, the keyboard can locate possible error candidates given the correction, and commit corrections accordingly. I developed three interactions to support the concept, including *Drag-n-Drop*, *Drag-n-Throw* and *Magic Key.*

TypeAnywhere [275] is a QWERTY-based text production system for ubiquitous computing environments. Computing now has entered a ubiquitous era [237]: 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. *TypeAnywhere* tries to offer a solution by answering the question: *what if we can type on any surface on an imaginary QWERTY just like on a physical keyboard?* With *TypeAnywhere*, users can leverage their touch-typing skills from physical keyboards to perform text entry on any surface, wearing a finger-worn device that can detect finger taps. The decoder of TypeAnywhere is a neural language model that is able to translate the finger-tap sequences into expected text without relying on tap locations. As the model can incorporate the whole context including the previous text and the current tapping sequence, it achieves high decoding accuracy compared to the traditional n-gram based models.

1.3 Addressing the Problem of Interaction with a User-centered Approach

Computers are deterministic, while human communication is ambiguous. We use natural language to convey the same meaning with different expressions, yet when talking with computers, we have to perform a set of predefined operations. In return, computers will also talk to us in a predefined manner. As a result, the interface can not adapt to people with different abilities, creating extra barriers for them to communicate with digital information.

To create inclusive communication interfaces that are adaptive to the user's ability, I focused my thesis work on supporting blind or low vision (BLV) users to communicate with pictorial information, including emojis and animated GIFs. BLV users often rely on a tool called a screen reader to interact with their computing devices: the tool creates a focus on the screen, and the UI element being focused will be read out via the speaker. BLV users can then move the focus to navigate through the whole screen, and use keyboards or gestures to interact with elements. While screen readers support reading and inputting text-based information, there is a lack of support for using them to interact with pictorial information, such as emojis and GIFs. To improve the experience, I conducted interviews and co-design sessions with BLV users, and developed systems enabling easy communication with pictorial information via natural languages, which are more ambiguous than predefined commands and outputs.

Voicemoji [270] is a voice-based system that supports emoji search, entry and exploration. Keyboard-based emoji entry can be challenging for BLV users: they have to sequentially navigate emoji lists using screen readers to find the desired emojis, which is a slow and tedious process. On the other hand, the prevalence of emojis has also influenced how BLV users communicate online. According to a study in 2020 [215], 93.1% BLV users encounter emojis each month and 82.7% use emojis monthly. To create a better experience for BLV users to interact with emojis, I conducted interviews to understand their needs and challenges, and created *Voicemoji* based on the findings. *Voicemoji* utilizes search engines to support natural language style emoji queries, and utilizes neural network models to provide context-sensitive emoji suggestions through speech output. With *Voicemoji*, BLV participants achieved an 11x speedup when inputting an emoji compared to the standard emoji keyboard.

Ga11y [276] is an automated GIF annotation system for BLV users. Animated *Graphics Interchange Format* images, or "GIFs," are looped animations comprising a sequence of images, and are popular forms of content on the Web, messaging, and social media. Unlike emojis, whose descriptions are created and standardized by the Unicode Consortium[2], GIFs are mostly created by individual users, and there are no standard descriptions for screen readers to use. To address the challenge, I developed *Ga11y*, which combines computer vision and crowdsourcing to supply GIF annotations. The system contains three components: 1) an Android client in which users can trigger annotation requests, 2) a server for GIF matching and annotation storage, and 3) a web interface for annotation. The user can use the mobile client to record the GIF and request annotations. The server then compares the similarity of the requested GIF with existing GIFs in the database, and generates an automated description using computer vision if no similar GIF is found. In the meantime, the requested GIF will be displayed on the annotation website, where volunteers can provide their annotations. The human annotation will be returned when a similar GIF is requested the next time. In this way, users get timely annotations even when GIFs are new to the server, and over time, the number of human-annotated GIFs increases.

---

[2] `https://unicode.org/consortium/consort.html`

1.4 Evaluating Intelligent Communication Interactions

It is important to have intelligent systems assisting our communication with devices, and it is equally important to evaluate their performance and impact. Do they really help? What is the improvement in speed and accuracy? How do they affect user behaviors when composing a message? The last part of my thesis work tries to answer those questions. I derive metrics, models and tools to enable evaluation of text production systems with advanced features such as auto-correction and word prediction, and conduct studies to investigate how different emoji suggestion mechanisms affect the users' emoji usage and their message composition styles.

The *transcription sequence (T-seq)* model [272] is a method-independent text entry evaluation. Previously, the *input stream* model [200, 201] was used to conduct text entry experiments and compute performance metrics, like words per minute and error rates. However, the model uses a strictly serial character representation for encoding the text entry process, making it unable to support many modern text entry features such as cursor moving, auto-correction and prediction. To address these limitations, I introduce the concept of transcription sequences: for each new input, a snapshot of the entire transcribed string unto that point is captured. By comparing adjacent strings within a transcription sequence, we can compute all extant metrics, reduce artificial constraints on text entry evaluations, and introduce new metrics.

The *text entry throughput* [274] is a metric combining the speed and accuracy performance of a text entry method. Human-computer input performance inherently involves speed-accuracy tradeoffs—the faster users act, the more inaccurate those actions are. Therefore, comparing the speed and accuracy separately can result in ambiguous outcomes: how to compare the overall performance of two text entry techniques, if one is faster but more error-prone than the other? Based on Shannon's information theory [191], I mapped the text entry process as an information transmission model, and derived the throughput of the model as the overall information transmission rate of the process. The metric could serve as a performance measurement for any text entry method, essentially unifying the speed

and accuracy. Thus it is robust to different speed-accuracy trade-off biases.

To evaluate how emoji suggestion systems affect a user's emoji usage and writing behavior, I conducted studies to compare *lexical and semantic emoji suggestions* [269]. Although emojis themselves are known to enrich conversations [44, 101], the role that different emoji suggestion systems play has not been explored. To answer the question, I implemented a keyboard capable of offering both lexical and semantic emoji suggestions, and conducted both in-lab and field studies to measure how the emoji usage and the message composition differed with different emoji suggestion mechanisms: no suggestions, lexical suggestions, and semantic suggestions. The results indicated 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.

1.5 Dissertation Overview

The remaining part of this dissertation is structured as follows:

Chapter 2 positions the dissertation in the context of the related work, and provides background knowledge that the work is built upon. It introduces related research corresponding to the three aspects of the thesis work: 1) for intelligent text production interactions — text input and editing interactions, machine learning algorithms for natural language processing are reviewed; 2) for accessible pictorial information interactions — the usage and accessibility of emoji and animated GIFs in online communication, and techniques supporting BLV users to access that information are discussed; 3) for evaluating communication systems, models and metrics for text entry systems are reviewed.

From there, the following eight chapters are organized into three parts, each representing a coherent topic. Part 1 describes the work of intelligent text production interactions:

- Chapter 3 introduces *PhraseFlow*, a phrase-level keyboard that can correct previous

text based on the subsequent input sequences. I describe the interface design and implementation of the keyboard, the comparison with the word-level keyboard, and results from the in-lab and deployment studies.

- Chapter 4 presents *Type, Then Correct*, a novel text correction concept that allows the user to first type the correction, and apply it to the error with three interactions: *Drag-n-Drop*, *Drag-n-Throw* and *Magic Key*. I describe the design considerations of the three techniques, the encoder-decoder model used to locate the error text given the correction, the study design, and results of the performance of each interaction.
- Chapter 5 talks about *TypeAnywhere*, a text entry system that allows the user to perform QWERTY typing on any surface with a pair of finger-worn devices. I explain the decoding mechanism, the interface design, the training process of the neural language model, and the results of a five-day longitudinal study with ten participants.

Part 2 describes the work of accessible pictorial information interactions:

- Chapter 6 introduces *Voicemoji*, a voice-based system that enables BLV users to search and explore emojis. I conduct user interviews for understanding BLV users' experience using current emoji input methods. Based on the user needs, I then present the design, implementation, and evaluation of the system.
- Chapter 7 presents *Ga11y*, a computer vision + crowdsourcing GIF annotation system for BLV users. I talk about the system development, user studies of GIF description task designs, study methods, and the user feedback.

Part 3 describes the work of evaluating intelligent communication systems:

- Chapter 8 introduces the *transcription sequences model* for unconstrained text entry evaluation. I talk about algorithms to calculate traditional and new metrics from the model, and the validation experiments of the model correctness.
- Chapter 9 explains the *text entry throughput*, which unifies speed and accuracy into a single performance metric. I discuss the similarity between the text entry system and the information transmission system in the information theory, the derivation

and calculation of the metric, and the experiments demonstrating the stability of the metric among different speed-accuracy trade-off conditions.

- Chapter 10 presents a comparative study of *lexical and semantic* emoji suggestion systems on how they affect the user's emoji usage and input behavior. The implementation of the emoji suggestion keyboard, the in-lab and deployment study design, and the quantitative and qualitative results of different suggestion mechanisms are presented.

Chapter 11 discusses reflections and insights from the work; Chapter 12 summarizes the contribution and proposes opportunities for future research.

Chapter 2

RELATED WORK

Research on improving the communication interaction with the device has been around even before the computers were invented: people already started to optimize the keyboard layout since the typewriter became popular[1]. In this section, I cover related work from three main aspects: 1) interaction techniques and algorithms for text production, including entry and editing interactions, machine learning algorithms for natural language processing; 2) accessible communication interactions for pictorial information, including the usage and accessibility of emoji and GIFs, and assistive techniques for BLV users to interact with them; 3) models and metrics for evaluation text entry systems.

2.1 Text Entry and Editing Interactions

Text input serves as a fundamental interactive part of almost any human-computer system, enabling expression, communication, and information capture. 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. In this section, we review research in text entry and editing techniques designed for different computing scenarios.

2.1.1 Ten-finger QWERTY Touch Typing

As the mainstream typing interaction, Ten-finger touch-typing on a QWERTY layout has been studied extensively since at least the 1970s [85, 184, 185, 190]. To understand how

[1] `https://en.wikipedia.org/wiki/Dvorak_keyboard_layout`

people perform touch typing with different finger-to-key mappings, Feit et al. [62] observed 30 participants typing on a physical keyboard. They found that people type with different finger motion characteristics and even different finger-to-key mappings, and that people who did not type with the standard finger-to-key mapping (Figure 2.1)[2] could also reach high levels of performance, over 70 WPM. Through clustering they identified six mapping strategies for the right hand and four strategies for the left. In a follow-up study [54], they found that rollover key-pressing was a key factor for fast typing, and faster typists often used more fingers to type. In other work, Findlater et al. [66] performed empirical studies of ten-finger typing on touch screens and found that the typing speed (58.5 WPM) was 31% slower than the physical keyboard without any feedback.

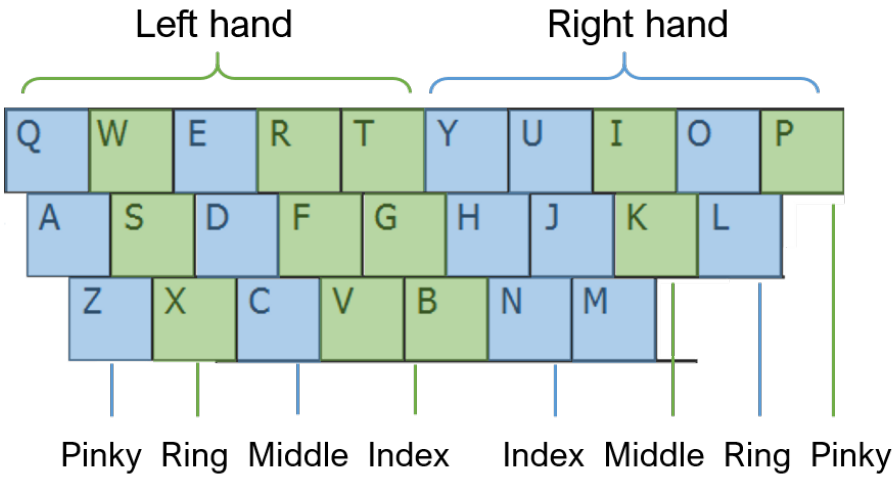

Figure 2.1: The standard finger-to-key mapping of the letters on a QWERTY keyboard. The mapping of the thumbs to the spacebar is not shown.

There are several projects also trying to enable QWERTY-style ten-finger typing without using a keyboard. Goldstein et al. [79] proposed a QWERTY-style typing interaction using a pair of gloves. However, their evaluation was only conducted on a mock-up without a functioning prototype. The Canesta prototype [180] projected a keyboard on a table to en-

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

able QWERTY-style typing. TOAST [193] was an eyes-free keyboard that enabled the user to type on large touch screens without a visible keyboard. With its Markov-Bayesian algorithm for spatial decoding, participants reached an average speed of 44.6 WPM. However, this technique could only be applied on devices with large touch screens. Relatedly, Findlater et al. [65] showed how machine learning could be used to design touch screen keyboards on interactive tabletops that adapt at runtime to evolving finger positions. ATK [259] provided for mid-air typing using computer vision hand-tracking, reaching an average speed of 29.2 WPM. However, mid-air typing lacks tactile feedback and can easily cause fatigue. Richardson et al. [178] implemented a vision-based text entry system relevant to this project, where hand motion captured by cameras was fed into a neural network decoder that then output the decoded text corrected by another neural language model. Although they did not perform a user study to evaluate their system, the offline CER was reported to be 2.22%. While promising, such computer vision-based systems still require fixed-position cameras overlooking the hands wearing arrays of markers, making such systems impractical for ubiquitous computing.

2.1.2 Text Input Interactions 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 [134] is a one-hand chording keyboard, where the user grabs the device and press multiple keys on a grid to enter a character. The reported average speed was 26 WPM after 400 minutes (6.67 hours) of practice. Smart watch text entry interactions also falls into this category [84, 96, 124, 165, 257, 258], where the user performs text entry through the interface of the watch device. For example, COMPASS [258] utilizes the rotor of the watch for character selection, and reached 12.5 WPM with a dynamically positioned cursor. WatchWriter [84] implemented both tap- and swipe-based [121] text entry on a watch, achieving 24 WPM with 3.7% character error rate (CER). Velocitap [222] studied committing one word, multiple words, and one sentence at a time for smartwatch

input.

The other category of text entry interactions for wearables is freehand typing with external sensors. For example, TipText [254] (average speed 13.3 WPM) and BiTipText [253] mapped the QWERTY layout onto a user's index finger and enabled "subtle" text entry with thumb-to-index taps sensed by capacitive overlays. BiTipText reached an average speed of 23.4 WPM. QwertyRing [86] was a ring that detected taps of the index finger and reached 21 WPM. Finger-T9 [250] (average speed 5.42 WPM) mapped the layout of a 9-key number pad to a user's finger segments for the thumb to tap. WrisText [81] detected wrist motions by proximity sensors and enabled joystick-like whirling input for text entry, reaching an average of 15.2 WPM. PinchType [60] (average speed 12.54 WPM) grouped letters that would be typed by the same finger on a QWERTY keyboard, and detected finger pinches for QWERTY-style input. SCURRY [113] was a glove-like device that detected finger taps with gyroscopes and accelerometers, and users entered text by pointing and clicking. ARKB [123] projected a see-through augmented reality keyboard using a head-mounted display, and tracked finger movements through cameras. TelemetRing [211] was a wireless ring-shaped keyboard that detected finger taps, similar to the finger-worn devices used in this project. However, TelemetRing used a chording system for text entry and did not report users' performance (as neither did SCURRY and ARKB).

Besides wearable typing, text entry techniques for different usage scenarios are also investigated extensively, including typing in the air [259], in VR [263], for glasses [262], for accessibility purposes [14, 248], and even on music instruments [61]. 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. [222, 223, 226]), without taking the longer context and the higher-level information into the decoding process. Challenges to finding satisfactory methods include learning new skills [81, 134, 262] and slow performance (i.e., less than 20 WPM). In sum, the search for effective text entry methods beyond traditional desktop computing environment is both longstanding and ongoing.

2.1.3 Text Correction Behaviors and Techniques

While much previous work focused on user behaviors during text entry, there have been a few projects that focused on the text editing process. Researchers have found that typing errors are common using touch-based keyboards: in a text entry study on smartwatches [124], out of 888 phrases, participants made 179 substitution errors, 31 omission errors, and 15 insertion errors. In a big data study of keyboarding [54], substitution errors (1.65%) were observed more frequently than omission (0.80%) and insertion (0.67%) errors. On the other hand, current text correction techniques are left wanting in many ways: for example, sending error-ridden messages caused by typos and auto-correction [119] is of greatest concern when it comes to older adults. Moreover, Komninos et al. [118] observed and recorded in-the-wild text entry behaviors on Android phones, finding that users made around two word-level errors per typing session, which slowed text entry considerably. Also, participants "predominantly employed backspacing as an error correction strategy."

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 [83]. Fuccella et al. [72, 73] 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 [9] 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 [157] to compare the edit distance of the text and the word in a dictionary. ReType [195] is a desktop keyboard based correction interaction that locates the the error positions guided by gaze information and correct them with the keyboard input. Commercial products also exhibit a variety of text correction techniques. Gboard allows a user to touch on a word and replace it by tapping on another word in a suggestion list. However, the technique is only limited to misspellings.

The Grammarly [3] keyboard keeps track of the inputted text, and provides corrections in the suggestion list. However, because Grammarly provides correction suggestions without guidance (e.g., it provides all possible error correction options without knowing which one the user wants to correct), the suggestion bar can become cluttered in the presence of many suggestions. Finally, some keyboards such as the Apple iOS 9 keyboard support indirect cursor control by treating the keyboard as a trackpad. Unfortunately, prior research [187] has shown that this design brings no time or accuracy benefits compared to direct pointing.

2.2 Machine Learning Algorithms for Natural Language Processing

To accelerate the information input process, keyboard interfaces nowadays will be equipped with advanced decoding algorithms to mitigate the user errors caused by the noisy human input, as well as to minimize the effort of mental and physical operations. Those algorithms are usually backed by machine learning models: from the traditional n-gram models to neural networks, language models act as an essential role for keyboards to understand the user's intention.

2.2.1 Statistical Language Decoding Models

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 [35, 70, 82, 109]. The spatial model relates intended keys to the probability distributions of the input information (such as touch coordinates) and other features [15,65,260,280]. The distribution is then combined with a language model, such as an n-gram back-off model [109], to correctly decode noisy touch events into the intended text [70, 82]. 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. [82]. Practical keyboards may also model spelling errors by adding letter insertion and deletion probability estimates in its decoding algorithms [166].

---

[3] https://app.grammarly.com/

Alternative text input interactions with intelligent decoding algorithms was also proposed over the past decades: Kristensson and Zhai invented SHARK^2 [121], 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 [65] that could adapt its layout to the users typing behavior gradually. A follow-up work TOAST [193] employed a Markov model in the keyboard decoding process and achieved 44.6 WPM on big touch screens. The Finger Fitts Law [25] proposed a dual-distribution to model finger's touch point distribution accurately; WalkType [78] incorporated the accelerometer data to improve the touch accuracy during walking conditions; Yin et al. [260] proposed a hierarchical spatial backoff model to make the touchscreen keyboards adaptive to individuals and postures. Weir et al. [236] utilized the touch pressure to "lock" the characters during decoding. Zhu et al. [280] showed that participants could type reasonably fast on an invisible keyboard with adjusted spatial models. Vertanen et al. [226] developed VelociTap, a phrase-level decoder for mobile text entry. Together with its follow-up projects [222, 223], 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.

2.2.2 Deep Learning for Natural Language Processing

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 [16], 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 [219], 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 [175], GPT-3 [31], BERT [53] and XLNet [255]. Deep language models trained by neural networks has achieved

significantly low perplexity [31, 53, 219], and the state-of-the-art performances on many language understanding tasks, including text summarization [183], document classification [170], question-answering [261] and conversation generation [31]. 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. [252] presented an encoder-decoder RNN model for text correction. Their model was built upon a sequence-to-sequence model for translation [16]. Neural models were also applied for gesture typing decoding [4]. In fact, commercial products related to text input such as Gmail has already released features like smart reply [108] and smart compose [36] 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.

2.3 Accessible Communication Interactions for Pictorial Information

Last section reviews interaction techniques for text production. However, online communication is not only about text: pictorial information, such as emojis and animated GIFs is now part of the internet culture, adding richer expressiveness on top of pure text. In this section, I review the current status of emoji and GIF usage online and their accessibility issues (especially for BLV users), and tools for making them accessible.

2.3.1 The Usage and Accessibility of Emoji and GIFs

Emojis and GIFs are both pictorial information that has been used extensively over the internet. Emojis are a set of pictorial Unicode characters with visual representations of expressions, activities, objects, and symbols. Since they were inducted into the Unicode Standard in 2009 [51], the usage of emojis has increased dramatically. A 2015 report by

Swiftkey [210] revealed that users inputted over a billion emojis over four months; a similar report in 2017 from Emojipedia [33] showed that five billion emojis were sent *daily* on Facebook messenger. Because of their pictorial appearance, emojis "convey information across language, culture, lifestyle and diversity" [2]. In fact, people sometimes even use pure emoji combinations unaccompanied by text to covey their expressions (*e.g.,* 📚✈️ = *book a flight*) [41, 101].

People use emojis for different purposes. Emojis can be used to provide additional emotional context or situational information [44], change the tone of a message, engage the recipient, or maintain a relationship [44, 110]. People also use emojis in highly personalized and contextualized ways to create "shared and secret uniqueness" [110,179,214,241]. An example provided by Wiseman and Gould [241] showed that a romantic couple used the pizza emoji 🍕 to mean "I love you" because of their shared love for pizza. In general, the usage of emojis improves the expressiveness of online communication [98, 107, 230].

It is not surprising that people who are blind or have low vision (BLV) also use emojis in their written communications. According to Tigwell *et al.* [215], over 93.1% BLV users engage with emojis at least once a month. These people's purposes when using emojis are the same as those of sighted people, including enhancing message content, adding humor, and altering tone. Unfortunately, the accessibility and usability of emoji interfaces for BLV users is lacking, although there have been some efforts to improve upon this situation.

Images in *Graphics Interchange Format*, called "GIFs," are in a file format that can contain multiple images played in an animation loop. Such files are usually extracted from clips in videos [58] and from animations and cartoons created by artists as "stickers" [233]. Compared to emojis, which are controlled by the Unicode Consortium, GIFs are more "democratic," where in theory, everyone can create or modify GIF content, enabling personalized communications [269, 279]. GIFs are commonly used on social media platforms such as Facebook, Tumblr, Twitter, and Reddit [74], and in online messaging apps, such as Facebook Messenger, WhatsApp, and Wechat [279]. By 2018, the GIF database and search engine service *Tenor* reportedly had 12 billion searches every month [212], while another service, *Giphy*, reached 700 million monthly active users in 2019 [206]. Clearly, the

popularity of GIFs online is immense.

Researchers have found that GIFs make online interaction more engaging than static images or text [102] because of their animation, storytelling capabilities, and utility in expressing emotions [17]. Jiang et al. [103] summarized users' motivations for sending GIFs online, including to convey emotion, to express nuanced meanings that were hard to convey with text, to make humorous and eye-catching posts, and to start engaging conversations. GIFs are "a visual language unto themselves, and an emotive vocabulary made out of culture" [151]. Indeed, many GIFs are blended with pop culture and memes, where contextual information is vital to understand their meanings [103, 106, 256]. For example, the source of a GIF, the meaning of the text meme on the GIF, and the usage of a GIF are all deeply embedded in one's cultural background [93, 151].

Unfortunately, much of the GIF content remains inaccessible to BLV users. According to Gleason et al. [74], only 0.04% of GIF content on Twitter contained alternative text in February 2020. According to the study I conducted with BLV participants ($N = 19$) [276], all had encountered GIFs online, but most of the time the contents were unlabelled, causing participants to either ignore GIFs or ask for help from sighted people. Although computer vision techniques are able to generate reasonable descriptions for many static images, correctly describing the contents of animated GIFs is still a challenging research problem [128], let alone providing the contextual and cultural information needed to understand them.

2.3.2 Assistive Techniques for Interacting with Pictorial Information

Visual content such as images, videos, stickers, badges, memes, and emojis can enhance online communication and social interaction. Prior work mainly has focused on improving the accessibility of static pictorial information, such as pictures [144, 155] and memes [75, 76, 173] posted on social media. Those work utilized human-in-the-loop plus automatic methods such as optical character recognition and scene description. However, there is little effort on making emojis and GIFs accessible.

Owing to their inaccessible pictorial nature, emojis can be easily misunderstood by BLV

users. For example, the same emoji can have different definitions on different platforms, and can also be read differently by different screen readers. This inconsistency can cause frustration and misunderstanding [215]. Furthermore, many emojis have similar descriptions, such as 😄 (*Grinning Face with Smiling Eyes*) and 😊 (*Smiling Face with Smiling Eyes*), which are hard for a person to distinguish without visual portrayals. As a consequence, research shows BLV users can lack confidence when selecting emojis [215]. To help remedy this problem, Kim *et al.* [111] combined machine learning and k-means clustering to analyze the conversation and recommend emojis that represent various contexts, which can ease the challenge of selecting appropriate emojis for BLV users. Web developers utilized the Aria-label with emoji text to standardize the description for screen readers [192]. Researchers also designed emojis [39] that can be combined with Braille text. And prior work has addressed the problem of how to make emoji *output* more accessible [43, 235].

The common solution for making images or animated GIFs accessible is to use alternative text, which is a method of adding text descriptions to images so that a screen reader can read it for BLV users. Researchers have investigated the usability of alternative text extensively, including how framing affects users' trust [144], what granularity descriptions should have in different usage scenarios [208], auto-generated captions using the surrounding text of an image [87] or using user-generated comments [231], alternate designs such as multimodal and interactive alt-text for rich visual content [154]. For example, Gleason et al. [74] proposed using audio descriptions to supply emotive qualities to the descriptions of animated GIFs.

Outside some prior academic research, there has been little recognition in industry that the inaccessibility of emojis and GIFs is a problem. For emojis, the predominant way to select and input an emoji is to visually search over an emoji keyboard, which offers emojis in a multi-page menu, grouped by theme. This method is imprecise and slow, even for a sighted user; it is even more unusable for blind or low vision (BLV) users. To navigate among emoji options, a BLV user has to make a swipe or drag the finger to move the focus onto the next emoji, and continue to step until the expected emoji is spoken, which could take tens of minutes. Emoji keyboards such as *Gboard* have built-in emoji search functions.

However, the emoji search function is based on keywords that are assigned manually; hence, this function does not provide much flexibility. For example, users can enter "fruit" to find fruit emojis, but "fruits" produces zero results. Another issue of keyword-based search is the poor transferability between languages: searching emojis on a Chinese keyboard often leads to fewer results than performing the same search on an English keyboard because the names of emojis are defined in English, and there are no official emoji names in Chinese.

For GIFs, Twitter has launched the alt-text function for GIFs [218], and many GIF services such as *Giphy* and *Gboard* offer a one-word description of GIFs, such as "laugh" or "dance" when exploring GIFs; however, providing such short descriptions does not aid users in understanding nuanced expressions contained in animated GIFs. Neither does it help them to decide which GIFs to use. Furthermore, once a GIF is selected and sent from a keyboard like *Gboard*, it become unlabelled at its destination (e.g., for its viewer or recipient).

2.4 Models and Metrics for Text Entry Evaluation

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 [126], while others prohibited erroneous characters from appearing [220], or disabled all error correction [142]. Those evaluations were both unnatural and constrained, and could lead to further errors in the

transcription [145].

In 2001, seminal work by Soukoreff and MacKenzie [200] began to loosen these constraints by using the Levenshtein minimum string distance algorithm [125] to calculate errors based on the edit distance between two strings[4]. 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 [201] showed how to 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 [201] 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 [247, 259]. 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 [247]. Other error-related metrics such as Cost per Correction [11, 80] and word error rate [114] have been introduced. However, as before, these metrics rely on the sequential input stream paradigm, and therefore are similarly constrained. Method-independent evaluations based

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

on the IS model cannot accommodate many modern text entry behaviors.

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 [141] proposed Adjusted Words per Minute (AdjWPM) as

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

where E refers to the uncorrected error rate [201]. 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 [67] in 1954 for aimed pointing movements. In 1969, Fitts' colleague Pew introduced the "speed-accuracy operating characteristic," noting that "the relationship between speed and accuracy of performance under a wide variety of task conditions reveals a linear relationship between log odds in favor of a correct response and reaction time" [18] (p. 16). Fitts' law is an empirical model that combines the spread-of-hits and pointing time. Inspired by Shannon information theory [191], 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 [45] provided a correction to normalize the speed-accuracy tradeoff with his corrected throughput measure, which was later popularized by Welford [238] (pp. 147-149). Throughput, whose units is bits per second (bits/s), characterizes an aimed pointing task for its performance efficiency. For the mathematical details of calculating throughput, see MacKenzie, who gives a nice overview [10] (pp. 106-109).

In order to derive a metric reflecting the information transmission speed for text entry, Sourkoreff and Mackenzie [201] defined Utilized Bandwidth and Wasted Bandwidth to represent the amount of useful information transferred during text entry. However, both metrics

reflect percentages of correct keystrokes and do not take time into account. Thus, they are not unified speed-accuracy metrics. Soukoreff, in his doctoral dissertation [199], further proposed a model attempting to derive the metric. However, Soukoreff's model only considers correct characters in the transcribed string, and calculates throughput in terms of the information contained in the entry rate of correct characters. This calculation is both theoretically and practically insufficient.

2.5 Conclusion

As can be seen, the research of information communication interactions has a long history since it is one of the essential areas of HCI. Researchers have explored the area through different aspects, by inventing new input interactions for novel scenarios, deriving intelligent decoding algorithms, and designing systems to make the information accessible. The thesis work is heavily built upon previous research, and extends the state of the communication interactions towards a humanlike stage, attempting to make them more intelligent, intuitive, and inclusive.

Part I

INTELLIGENT TEXT PRODUCTION INTERACTIONS

Text production is the main interaction to communicate information with computers. The first part of my work demonstrates how we could leverage AI to build intelligent text input and editing interactions, accelerating the input process and enabling ubiquitous text entry by incorporating the language and action context.

Chapter 3

PHRASEFLOW: DESIGNS AND EMPIRICAL STUDIES OF PHRASE-LEVEL INPUT

Decoding on phrase-level incorporates more language context which improves the correction accuracy compared to that on word-level according to previous research. However, how phrase-level input affects the user typing behavior, and how to design the interaction to make it practical remain underexplored. In this chapter, I present PhraseFlow, a phrase-level input keyboard that is able to correct previous text based on the subsequent input sequences. Computational studies show that phrase-level input reduces the error rate of autocorrection by over 16%. I found that phrase-level input introduced extra cognitive load to the user that hindered their performance. Through an iterative design-implement-research process, I optimized the design of PhraseFlow that alleviated the cognitive load. An in-lab study shows that users could adopt PhraseFlow quickly, resulting in 19% fewer errors without losing speed. In real-life settings, I conducted a six-day deployment study with 42 participants, showing that 78.6% of the users would like to have the phrase-level input feature in future keyboards.[1]

3.1 Introduction

Autocorrection has become an essential part of touchscreen smartphone keyboards. Due to the small screen size relative to the finger width, fast typing on a smartphone without autocorrection can produce up to 38% word errors [15, 70]. 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

[1]This chapter is adapted from: Mingrui "Ray" Zhang, Shumin Zhai. (2021). PhraseFlow: Designs and Empirical Studies of Phrase-Level Input. Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '21). Yokohama, Japan (May 8-13, 2021). New York: ACM Press.

show such auto-corrections can dramatically reduce the error rate in touch keyboards [70]. Indeed, commercial mobile keyboards such as Gboard [83], SwiftKey [150] and the iOS keyboard all provide word-level decoding, which corrects the latest typed literal string to an in-vocabulary word: for example, correcting *loce* to *love*. Banovic et al. [18] has shown that with a good autocorrection decoder, the user typed 31% faster than without autocorrection.

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

One possible solution to the above problems is to decode touch points on phrase-level, instead of only decoding and correcting the touch points of the last word. Phrase-level decoding may continue to decode the touch points even if the space key is pressed, and outputs phrase candidates. Velocitap [226] was one of the first attempts towards this idea: it presented a sentence-based decoder that was able to correct multiple words at a time. Follow-up projects by Vertanen and colleagues [222, 223] further investigated smartwatch devices decoding accuracy and typing performance on word level, multi-words level and sentence level. I show examples of word and phrase-level correction results in Table 3.1, based on actual results from Gboard and a version of my phrase-level keyboard, PhraseFlow, presented later in this chapter.

However, making phrase-level input practical faces many challenges. First, corrections beyond the last typed word require the user to pay attention to the early part of the

Table 3.1: Correction examples with word-level Gboard and phrase-level PhraseFlow. Phrase-level decoding can correct previous text using the future input context to avoid false corrections (row 1 & 2) or no corrections (row 3 & 4). It is also able to correct space-related errors (row 5 & 6)

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

phrase being typed; Second, delayed correction of the previous text requires the user to trust that the decoder would eventually and successfully correct the errors. If the phrase auto-correction failures, the delayed manual repair cost could be higher. Building upon the previous work, I present PhraseFlow, a keyboard prototype that focused on designing and studying the interfaces to support the phrase-level decoding. I limited the scope only to touch typing, in contrast to gesture typing [267]. 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?

I modified the Finite State Transducer (FST) based decoder [166] of Gboard to support phrase level decoding. I then performed simulation tests on the touch data collected from

a composition task. The results show that word-level decoding had 7.76% word error rate (WER) while phrase-level decoding had 6.47% WER, a 16.6% relative error reduction on this data set. Space related errors were also corrected by PhraseFlow.

To explore the design space of PhraseFlow, I iterated on multiple options of: 1. visual correction effects; 2. decoding commit gesture and behavior; and 3. suggestion displays. I first built a version of PhraseFlow with similar designs to the previous phrase-level input work [222, 223, 226]. The study results showed that phrase-level input with such designs introduced extra cognitive loads to the user, and alternative designs were needed to mitigate the effect. By incorporating empirical study results from the iteration, my final version keyboard managed to reduce the cognitive load and reached a comparable level of performance of the commercial keyboard. To test the user acceptance of the keyboard, I conducted a six-day deployment study with 42 participants. During the study, participants used PhraseFlow as their primary keyboard. The survey results showed that overall 78.6% of the participants would like to have phrase-level typing in their future keyboards, in comparison to 7.1% of the participants disliked the feature. Overall, the study results suggest phrase level input is a promising feature for future mobile keyboards.

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

3.2 Challenges of Designing PhraseFlow

The input chunk for typewriter-like physical keyboards is on *character level*: each key press modifies one character at a time. With smart functions such as auto-correction and word-prediction, touchscreen keyboards have enlarged the input chunk into *word level*: a string of characters inaccurately entered can be corrected into a likely intended word upon the press of the space key, which relaxes the need to type each character accurately. The basic research question of PhraseFlow is **how to further enlarge the input chunk to *phrase level***, as multiple words are corrected through one operation. Studies in human factors and psychology tended to find word as the basic chunk of typing [104, 186]. This means

that people mainly focus on only the current word when typing. Enlarging the input chunk into *phrase level* requires the user to pay extra attention on the previous text, which might hinder the typing performance. PhraseFlow therefore needs to overcome three new design challenges:

C1. How to signal the change when corrections happen. As the phrase-level input might change multiple words at the same time (and change the same word multiple times), I need to design effective feedback that is salient enough to inform the user about the correction, yet unobtrusive to avoid distracting the user from typing.

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

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

To my knowledge there is no single research method that can lead to all the insights needed to make significant keyboard performance progress. I therefore applied a variety of HCI research methods to address the challenges before us, including prototyping, simulation (offline computational tests), and lab-based composition or transcription typing, with both performance and subjective experience measurements. As a research vehicle I built PhraseFlow based on the Gboard [83] code base, bearing all its strengths and limitations. On the positive side, I leveraged many years of engineering work of Gboard on product polishing, computational performance and UI iteration, so a meaningful difference caused by the phrase-level input could be found against a strong baseline. On the other hand, Gboard as a commercial keyboard has a very compact language model with short span ngrams. Note

that previous work on the trade-off between the language model size and its correction power, albeit on a limited data set, did not show dramatic increase in accuracy from very large n-gram models [226].

3.3 Phrase-Level Decoder

The current decoder of Gboard [83] is a finite-state transducer (FST) [166] containing a spatial keyboard model and a n-gram language model consisting a 164K English word vocabulary and 1.3M ngrams (n up to 5). The original decoder would *commit* the last word and reset its status when the space key was pressed, and then restart the FST state with the touch points of the next word. For example, if the user typed *inter* and pressed the space key, the decoder would reset and output *inter* as the best candidate; when the user continued typing *ational*, the decoder would only decode the touchpoints of *ational*, failing to correct the whole typing to *international*.

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

Similar to VelociTap [226], to enable decoding touch sequence without word-delimiters, I also decreased the penalty of omitting a space between words, so that the decoder was able to handle contiguous text without space in between, such as *whatstheweathertoday*.

3.4 PhraseFlow V1.0

Figure 3.1 shows the interface of PhraseFlow v1.0. The workflow is as follows:

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

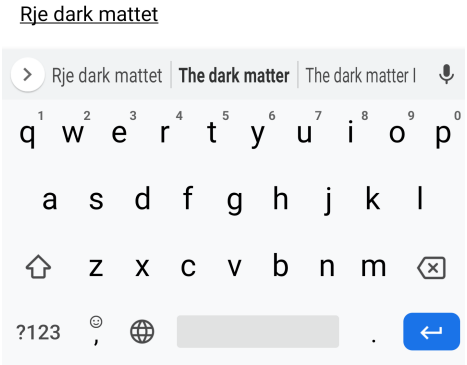

Figure 3.1: The interface of PhraseFlow v1.0. The keyboard layout was the same as Gboard. The typed text here is *Rje dark mettet*, and the autocorrection candidate *The dark matter* is in bold. Three candidates are shown in the list: the literal string, the autocorrection candidate, and the second best candidate

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

3.4.1 Interface and Interaction Design

For the first version of PhraseFlow, I explored several design options on the commit method, visual effect of correction, suggestion bar display, and active text marker. I chose those options as they were reported to affect the typing performance in the previous work [3, 222, 223, 226].

Commit Method Since the space press was no longer a commit action with PhraseFlow, I needed to design a new interaction to let the user commit the correction candidate. Previous work [226] considered using swipe as the commit method. However, swipe also required the user to perform a very different gesture during tap-typing, and the gesture also confused the user with gesture typing on mobile keyboards.

I made PhraseFlow commit the correction to the active text on every nth space the user typed, *i.e.*, when the user typed the nth space in the active text (we call it *nth space commit method*). The rationale was that the users were already used to the space commit method with current keyboards, thus extra interaction for committing would increase the cognitive and manual control cost. Space press was a necessary step to compose the text, thus it was natural as a committing option. If n was set to 1, PhraseFlow would behave exactly the same as the current word-level decoding keyboards, i.e. committing the text on each space press. A larger n would potentially offer greater post correction power, but also demands more user attention on the longer span active text.

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

Visual Effect of Correction As pointed out in the design challenge section C1, it is important to design a good signal when correction happens. For example, if *is coning home* is corrected to *is coming home*, the user should be able to notice the change. I experimented with three feedback effects to indicate the correction after the user performs a committing action: 1) *Background flash*. When a string of text was corrected to another

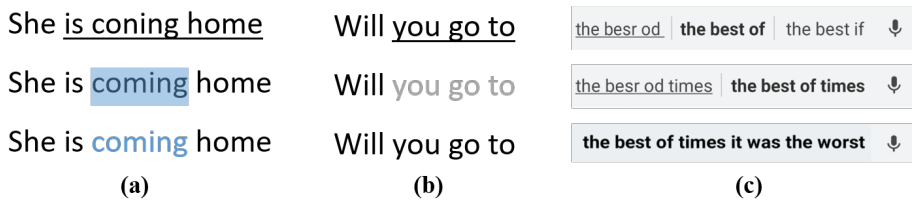

Figure 3.2: (a) Three visual effects of correction: when committing *is coning home*, the second row shows the *Background Flash* effect; the third row shows the *Color Flash* and the *Color Change* effect. (b) Three markers of the active text. (c) The suggestion bar display. As the input lengthens, the number of candidates decreases from three (top) to two (middle), and eventually one (bottom)

one, its background color would flash for 400ms. This is used by many current commercial keyboards. 2) *Color flash*. The color of the changed text would flash when it was corrected. 3) *Color change*. The color of the text would change to blue when it was corrected, and would change back when a new input action (such as cursor-moving, typing) happened. The three effects are shown in Figure 3.2(a).

Active Text Marker As is often suggested [164, 239], a system's internal state should be appropriately represented to the user. To make the user aware the range of the text that might be changed. I studied three active text markers of the active text shown in Figure 3.2(b): *underline*, *gray color*, and *no-marker*. The purpose was to have a visual effect that was not distracting but could also be informative of the active text range, echoing to design challenge C2. The no-marker design is used in iOS .

Suggestion Bar Display Since the decoded candidate may contains multiple words, the default display option of the suggestion bar, *i.e.*, always displaying three candidates, might make the user feel overwhelmed. To reduce the cognitive load of the user (challenge C2) and also make the candidate text always visible, I adopted a dynamic displaying design illustrated in Figure 3.2(c): the suggestion bar will first display all three candidates; as the text grows, it will only display two candidates and eventually decrease to one candidate if

the active text is too long before the committing. In this way, I can show the complete candidate information without squeezing or hiding the text.

3.4.2 Study 1: Evaluating Design Options

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

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

3.4.3 Study 2: Performance Simulation of PhraseFlow V1.0

This study used simulations, or "computational experiments" [70] to measure the accuracy of PhraseFlow v1.0 on autocorrection. Unlike Fowler et al. [70] which used model generated data in their simulation, I used a "remulation" approach [24] in this study: I recorded touch input data set collected in a text composition task. I then ran the data set through both the PhraseFlow v1.0 decoder and its Gboard word-level baseline decoder in a keyboard simulator. Emulating user typing behavior on a mobile phone, the simulator took touch coordinate sequences as input, then simulated the noisy touch input on a keyboard layout as input to the decoder. The simulator then compared the decoder output with the expected

text, and calculated Word Error Rate (WER) of the output results.

To collect the evaluation data set, I conducted a composition study with 12 participants (7 male, 5 female) to gather their touch points on a keyboard without auto-correction functions. Modelled after the study of composition types by Vertanen and Kristensson [225], I designed six composition prompts listed in Table 3.2. The participants were instructed to type a long message based on the prompt fast and not to care about making errors. For each prompt, the typing lasted for three minutes. The study was conducted on a Pixel 3 smartphone, and autocorrection was disabled for the test. After the composition of each prompt, I asked the participants to read their raw text and type the corresponding correct message they intended to compose on a laptop. To ensure that participants typed the correct text on laptop, the experimenter and the participants reviewed the text together and corrected the errors if there were any. I logged their raw touch points, the raw text, and the corresponding correct text for simulation. In total, I collected 72 phrases composed of 4955 words. The average word length for prompt P1 to P6 was 63 words, 75 words, 66 words, 69 words, 64 words and 77 words respectively. Participants were compensated with $25 for the 45-minute study.

Table 3.2: Six prompts for the composition tasks

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

I measured the error rate of the original raw data in regards to the provided correct text, using character error rate (CER) and word error rate (WER). WER is the word-level edit

distance [125]. The average CER was 6.18% (SD=2.9%) and the average WER was 26.4% (SD=11.1%). To conduct the offline computational evaluation, I kept the space key touch points but removed all punctuation-related touch points in the log data, fed the logged touch points into the simulator as the raw input (by replaying the touch points), and compared the simulation output with the correct text provided by the participants. I compared the current word-level decoder of Gboard, and the PhraseFlow decoder with the every-fourth-space commit method. The WER for the Gboard baseline was 7.76%, and 7.13% (n=4) for PhraseFlow. This 8.1% relative error reduction was a modest but clear improvement in error correction even on a mobile grade compact language model (set at 164K vocabulary and 1.3M ngrams). The WER was 7.14%, 7.03% and 7.08% when n was set to 2, 3 and 5. Given the similar performance, I fixed on $n = 4$ because of the result of study 1.

It is difficult to precisely compare this result with those by Vertanen and colleagues [222,226]. Their results varied with a large set of decoding (from 0.4 M to 194 M word ngrams), UI (including no space separation between words), task (such as Enron mobile phrase set transcription), and form factor (phone vs. watch) variations. They generally show that more errors were corrected in phrase-level decoding than in word-level decoding. For example, in one simulation that replayed phrase input data collected on a watch-sized keyboard but assuming perfect space separation between words, they showed the character error rate (CER) reduced from 2.3% to 1.8% when the input and decoding size increased one word to five or six words. Assuming character errors were evenly distributed in words and the average word length were 4.7 letters as in common English, the corresponding word error rates (WER) were reduced from 10.35% to 8.18%[2].

3.4.4 Study 3: Pilot Study of PhraseFlow V1.0

I conducted a pilot study to test my design of PhraseFlow v1.0 in comparison to a word level baseline. I developed a text editor application shown in Figure 3.3 as the experimental apparatus. The application uses the transcription sequence model of Zhang and Wobbrock [272] for evaluation. Transcription sequence contains the sequence of the transcribed text

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

each time its value changes, and by comparing the adjacent two sequences, it is able to analyze the dynamic text change during the typing procedure. Instead of composition tasks, I chose text transcription tasks for evaluation studies, as the text contents were controlled. Six participants (4 male, 2 female) were recruited via convenience sampling. The participants were told to enter the text shown on the screen as accurately and fast as possible using two keyboards: PhraseFlow v1.0 and unmodified Gboard. They were also allowed to use their comfortable posture for the task (all used two-thumb posture). The order of the keyboard was balanced. I randomly sampled 30 phrases from the Mackenzie phraseset [140] for each keyboard session. Afterwards, I conducted a short interview to gather their feedback. The participants were compensated with 15 USD for the 30-minute study.

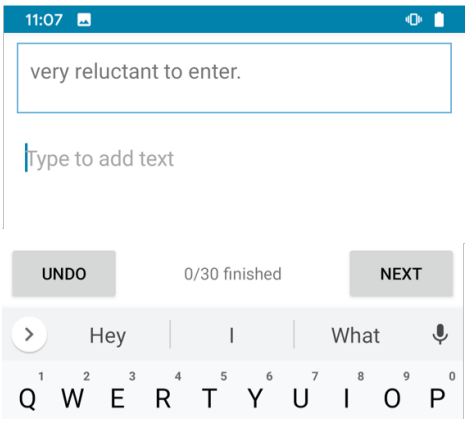

Figure 3.3: The text editor application used in the study. Hitting ”undo” will restart the current trial

In total, 360 phrases were collected. I calculate words per minute (WPM) by subtracting the first character timestamp from the last character timestamp, as noted in [136]. The average typing speed was 52.3 WPM in the Gboard condition, and 43.5 WPM in the PhraseFlow condition. I also calculated the character error rate (CER) and word error rate (WER). The average CER was 0.6% in the Gboard condition, and 0.7% in the PhraseFlow condition. The average WER was 2.3% in the Gboard condition and 2.4% in the PhraseFlow condition.

Table 3.3: The average speed and accuracy of each participant

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

Table 3.3 shows the performance of each participant.

To gain a deeper understanding of their typing behaviors, I analyzed the inter-key interval (IKI) [54] which was the time difference between two keypress events. I divided the IKI into two categories: the in-word IKI, *i.e.,* the time interval between two key presses within a word; and the between-word IKI, *i.e.*, the time interval between the presses of space and the next word. The average in-word IKI was 0.192s for Gboard and 0.210s for PhraseFlow. The average between-word IKI was 0.265s for Gboard and 0.311s for PhraseFlow. The larger difference on the between-word IKIs indicated that participants spent longer time to start typing a new word after pressing a space.

3.4.5 Discussion of the Pilot Study

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

reviewing them, as P1 commented, "*there is just too much going on in the suggestion bar. I have to read longer text and sometimes the number of options changes. It is distracting.*"

The analysis of IKIs also sheds light on participants' cognitive load during typing. Cognitive load may reflect the exterior information need during a task [209]. Studies in writings have found that shorter pauses contributed to higher writing fluency and lower cognitive load [5], and people generally had longer pauses at word boundaries [149]. The trend was similar from this study results: the in-word IKIs were on average shorter than the between-word IKIs, either in Gboard or PhraseFlow condition. However, the larger between-word IKIs of PhraseFlow (0.311 second) than Gboard (0.265 second) indicated that after pressing spaces, the participants might have carried greater cognitive load as they paused longer before starting typing the new word.

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

3.5 PhraseFlow V2.0

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

3.5.1 Buffer commit method

PhraseFlow v1.0 would commit all the active text and apply the phrase level candidates at once when the nth space was pressed, which caused the user to spend time reviewing the correction after the commit. The user also had to spend more cognitive resources to manage the results of space presses, since each space press behaves differently (some would commit corrections while the rest would not). In version 2, I designed a first-in-first-out buffer commit method as shown in Figure 3.4: when the nth space was pressed, only the first

word in the active text would be committed with its correction candidate; the remaining text would stay active. For example, if n was set to 3, when the user typed *thid is the* and space, only *thid* was committed and corrected to *this*; the active text then became *is the* . In this way, there would always be a buffer in the active text, which enabled the decoder to correct text in a continuous manner. The space press behaves more consistently therefore causing no *surprise* to the user. Buffer style can also handle space related errors better than committing multiple words at once, as all space presses will be decoded within the buffer. A punctuation key press would commit and clear all remaining text in the buffer.

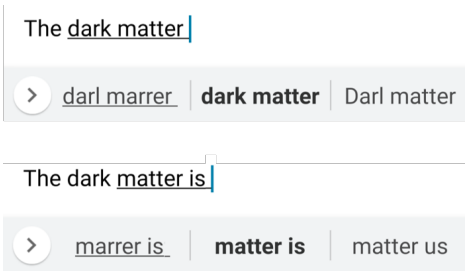

Figure 3.4: Buffer commit method in PhraseFlow v2.0. If the keyboard commits on every 3rd spaces, the upper figure shows the active text is *dark matter*, and the lower figure shows when the space is pressed after *is*, *dark* is committed (*"dark matter is"* already had 2 spaces)

3.5.2 Real Time Correction Feedback

In the pilot study of PhraseFlow v1.0 I learnt that showing the raw text distracted the users and caused the uncertainty whether the text would be fixed later. I thus applied the partial decoding results to the active text: when the user typed a space, the active text before the space would be updated to the suggested corrections, providing the real time correction feedback to the user. As shown in Figure 3.4, the raw text typed was *darl marrer*, but the active text was displayed as the correction. Since the correction was already shown as the active text, I only show the word-level candidates of the latest word in PhraseFlow v2.0 to make the suggestion bar more familiar to the users and less distracting.

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

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

I compared two keyboards: Gboard and PhraseFlow v2.0. The buffer length n was set to 4 for the nth space buffer commit method. There were three identical parts of the study, with each part containing two transcription sessions with each keyboard. The order was balanced, and for each session, 20 different phrases from the Mackenzie phrase set were randomly selected. Before the formal sessions, participants practiced five phrases with each keyboard as a warm up. Participants were told to type as fast and accurately as possible, and that they could take a break between sessions. After the typing task, the participants filled out an SUS usability survey and a NASA-TLX survey [88] (for measuring the perceived workload). I also briefly interviewed the participants on their thoughts of using the two keyboards.

3.5.4 Results

In total, I tested on 20x2x3x12=1440 phrases. The results are shown in Figure 3.5. I analyzed all metrics using Wilcoxon signed-rank test rather than the potentially more sensitive parametric variance analysis. I observed that none of my performance metrics followed a normal distribution.

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

Error Rate For Character Error Rate (CER), there was no significant difference between Gboard and PhraseFlow ($p > .1$). The average CER for Gboard was 0.017 while for PhraseFlow was 0.015. However, the Word Error Rate (WER) for PhraseFlow was significantly

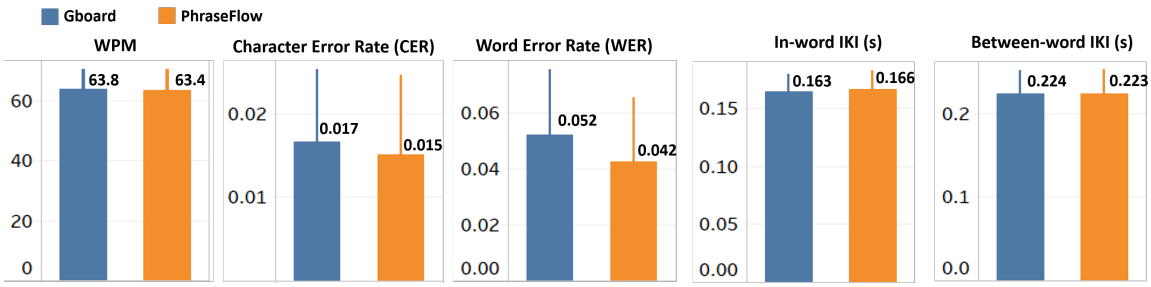

Figure 3.5: The in-lab study results of Gboard and PhraseFlow. Error bars are one standard deviation

lower than Gboard ($p < .05$). The average WER was 0.052 for Gboard and was 0.042 for PhraseFlow, a 19.2% reduction.

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

Subjective Scores The SUS and TLX scores are shown in Figure 3.6, and the difference between the two keyboards was not significant for SUS ($p > .1$) or TLX($p > .1$). The median SUS score was 83.8 for Gboard and 80.0 for PhraseFlow, and the median SUS score was 3.4 for Gboard and 3.4 for PhraseFlow. I looked into the individual SUS scores, finding that five participants rated higher score for Gboard, five higher for PhraseFlow, and two rated the same scores, which meant the preferences among the participants were split.

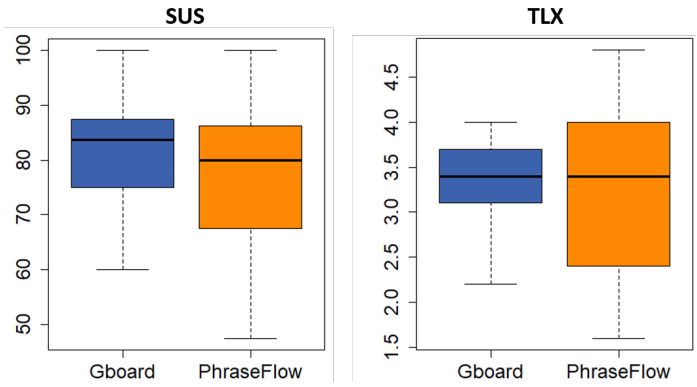

Figure 3.6: The SUS and TLX scores of the two keyboards. For SUS score, the higher the better; for TLX score, the lower the better

3.5.5 Discussion of the In-lab Study

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

I conducted an analysis on the second aspect in the study data using the transcription

sequence log. If text before the last word was corrected but the space was not pressed, I counted that correction as a manual correction. I found in total 79 error instances that were corrected manually. Among them, 49 instances were autocorrection failures, *i.e.*, not corrected by the keyboard. For example, *wat* was not corrected to *war*, and *stattenmsy* to *statement*. This kind of error was caused by two overlapping reasons: the literal string typed was too far apart from the correct text, or the language model and decoding algorithm were not strong enough to handle the errors. Interestingly, there were also 30 instances that could have been corrected by the keyboard (as validated offline), but were manually corrected before the autocorrection. For example, *jo* to *no*, and *tat* to *that*. This showed that the participants did not trust enough on the correction ability of the keyboard, so they manually corrected the word while it was still in the active text.

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

3.6 Study 5: Deployment Study

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

3.6.1 Preparation

To prepare for the deployment study, I further improved PhraseFlow v2.0 prototype based on the in-lab study results. Specifically, I lowered n for the *n*th space commit method, and added personalization. The committing buffer length was decreased from committing at

every 4th space to 2nd and 3rd to reduce the text area needed attention during typing. I ran the simulation test again using the touch points from the composition study, with the nth space set to 2, finding that the decoder had a WER of 6.47%, which had 16.7% relative error reduction compared to the 7.76% WER of Gboard. I also incorporated the personalization feature of Gboard in PhraseFlow. Personalization could learn the words that user typed and add them to the user-vocabulary, which was a necessary feature for daily text entry such as typing names, emails and abbreviations.

3.6.2 Participants

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

3.6.3 Procedure

Before the deployment, I clearly expressed the goal of the study was to "evaluate an experimental feature of the keyboard for future improvement", and that participants should provide "objective and factual" evaluations in the later surveys, so as to minimize the experimenter demand effects [52]. I sent the participants the keyboard Android application Package (APK), together with explanations on the phrase-level decoding feature. I explicitly told the participants that I would not log any data from the keyboard to address their privacy concerns. To keep the study at a manageable length, I made the committing buffer length an between-subject variable by separating the participants into two groups: one with n set to 2 (committing at every 2nd space), and the other with n set to 3 (we refer to the conditions as *buffer-2* and *buffer-3* in the following sections). There were 21 participants in each group. Participants were told to use PhraseFlow as their primary keyboard through-

out the study. The study lasted for six days, with a survey sent out to the participants on the 2nd, 4th and the last day. The survey contained three questions asking for their perception and preference of PhraseFlow in comparison to previous word-level keyboard, and open-ended comments. All questions were shown in Table 3.4.

Table 3.4: Survey questions for the deployment study. Response for Q1 - Q3 is a Likert scale from 1 (don't like it at all / much slower than before / much better than before) to 5 (like it very much / much faster than before / much better than before); for Q4 it's open response

Survey Question
Q1. Do you think you would like to use the "phrase level correction" feature in a future mobile keyboard?
Q2. How do you think of your typing speed after using this keyboard with the new feature?
Q3. How do you think of the accuracy of this keyboard with the new feature?
Q4. Do you have any comments when using PhraseFlow?

3.6.4 Results

On overall preference, most (78.6%) participants in the study liked (ratings of 4 and 5) the PhraseFlow prototype and a small number (7.1%) did not (ratings of 1 or 2). Table 3.5 gives more detailed rating statistics. The participants rated PhraseFlow positively on preference ("like to use", mean score ranging from 4.0 to 4.3 depending on days and the committing buffer length), perceived speed (mean score from 3.0 to 3.6) and perceived accuracy (mean score from 3.3 to 3.7). Note that a neutral 3.0 rating would match PhraseFlow to the participants' years of experience using word level keyboard in their everyday mobile interaction. Notably participants rated their overall preference of phrase level input higher than their perceived speed and accuracy improvement individually, suggesting their positive experience had more contributing factors than speed and accuracy alone.

Table 3.5: The average scores ± 1 SD of the survey responses on preference, speed and accuracy of the PhraseFlow.

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

Preference As illustrated in Figure 3.7, users' ratings of PhraseFlow varied with committing buffer length and over days of use. For example, on the overall preference ("like to use") and in the shorter committing buffer (*buffer-2*) group on Day 2, one user's rating was negative (2 on the scale of 1 to 5) , two neutral, 12 positive, and 6 very positive. On Day 4 and Day 6, the negative score in this group disappeared. The trend of the longer committing buffer (*buffer-3*) group, however, was the opposite. All users were positive or neutral on Day 2, but three turned negative on Day 6. User comments suggested "*false correction happens*" and "*backspace cannot revert the correction*" as the primary reasons for the decreased rating.

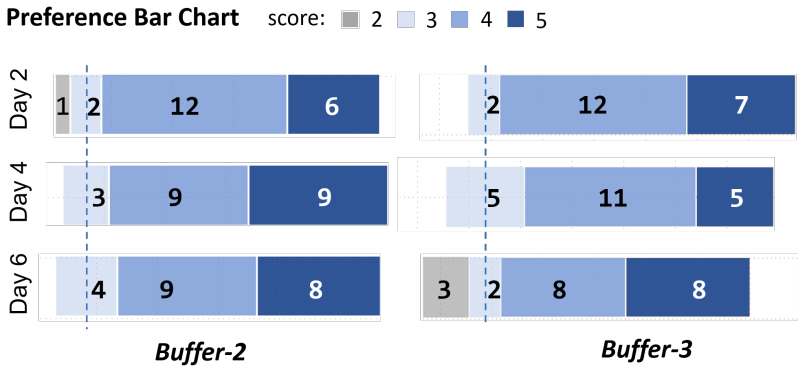

Figure 3.7: Distributions of PhraseFlow preference scores from 1 (*don't like it at all*) to 5 (*like it very much*). A score of 3 meant neutral opinion (represented with dashed lines)

Speed According to Figure 3.8, both groups increased their scores gradually, especially

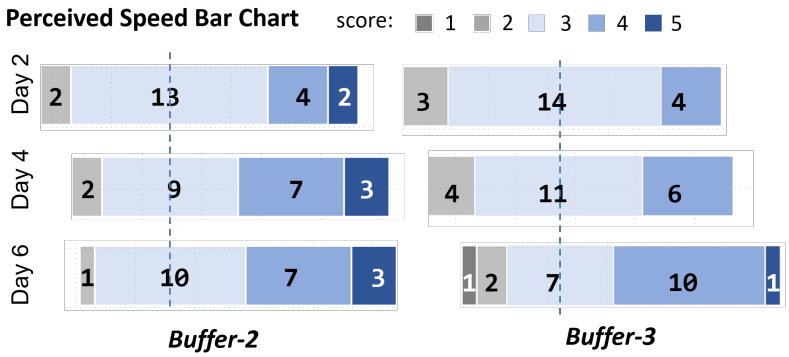

Figure 3.8: Distributions of speed responses from the three surveys. Compared to the word-level decoding keyboards the participants used before, PhraseFlow was rated from 1 (*much worse*) to 5 (*much better*). A score of 3 meant neutral opinion (represented with dashed lines)

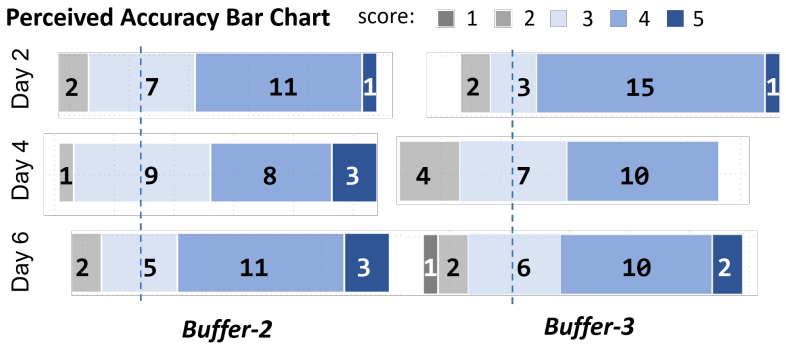

Figure 3.9: Distributions of accuracy responses from the three surveys. Compared to the word-level decoding keyboards the participants used before, PhraseFlow was rated from 1 (*much worse*) to 5 (*much better*). A score of 3 meant neutral opinion (represented with dashed lines)

for the *buffer-3* group. It appears that underlining the active text revealed the nature of *phrase-level correction* effectively enough to encourage user to type faster and trust the keyboard more during the study.

Accuracy As shown in Figure 3.9, people in *buffer-2* group gradually increased their ratings from neutral to more accurate, or even much more accurate. However, people in *buffer-3* group decreased their scores, while they initially rated high accuracy in the first survey.

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

Participants also mentioned that they got used to PhraseFlow after using the keyboard for a while. For example, P13 complained about the correction happened too frequent in the first two surveys, but got used to it in the last survey: "*I have gotten more used to it auto correcting a text twice. Love the feature.*" Participants also worried about the inaccuracy when they type uncommon words in the first two surveys. As they continued using the keyboard, they reported that the experience was smoother because of the personalization feature. Although six days were long enough, five participants reported that they still corrected immediately once an error happened as with their previous keyboards. P25 commented that "*It's hard to suppress the urge to correct the first word, so I don't think I really get to the part where PhraseFlow does its best work.*"

3.7 Discussion

Complementing the in-lab study, the deployment study results showed that despite being a novel user experience, the phrase-level decoding keyboard prototype PhraseFlow 2.0 was favorably received by most users during and after a few days of use. It appears the iterative

design, development and research process has led to a version of phrase typing keyboard practical and beneficial enough as the primary keyboard in daily use by most users. In the following I drew further insights and lessons from the PhraseFlow project to inform future phrase level keyboard design. I also discuss the current limitations and future directions for the longer term.

3.7.1 Design Suggestions for Phrase-level Decoding Keyboard

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

Visual cues need to manage the user's attention judiciously. It is important to use visual cues to convey the states of the system (such as the active text), and notify when changes happen. This is consistent with recommendations in [6]. The choice of the visual effect also matter. Using underline to mark the span of the active text under decoding provides sufficient information of the range yet not too distracting (Figure 3.2(b)). On the other hand, correction effects should catch user's attention and thus a salient effect with abrupt onsets [105] need to be applied. In my study, background flash was proven noticeable and preferred by users (Figure 3.2(a)).

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

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

Optimizing the decoding span length. While a longer span of active text provides the decoder with more context hence more correction information in principle [222, 223, 226], it is not the longer the active span the better. Even if a large n-gram language model approximately matches the user's input sequence, there is always a chance for false-correction. Second, a longer span of active text imposes higher cognitive costs in terms of user attention and requires the trust from the user. The analysis of the in-lab study showed that users tended to immediately manually correct their errors before the decoder took actions. While the user generally focused on the latest word during typing [104, 185], paying attention to multiple words would also increase the cognitive load. Third, more distant errors that are uncorrected or falsely corrected imposes higher manual correction cost. I recommend the active text span be limited to two or three words, at least for the current mobile grade language models. Importantly, even an active decoding span of two words offers many fundamental benefits of phrase level typing identified earlier in the chapter, including better ability to correct space key errors.

3.7.2 Future Directions of Phrase Level Typing

Larger and more accurate language models are needed to support higher quality phrase-level decoding. Traditional word-level decoding primarily depends on unigram and bigram language models. Phrase-level decoding requires trigram and even higher ngram models to prevent frequent back-off to unigrams and bigrams which by definition do not maintain longer distance word connectivity. However, increasing frequent high order ngram grows the model size exponentially. Neural network language models trained by deep learning techniques such as the smart compose model [36] is promising at modelling

longer distance language context and may become practical on mobile devices memory and compute wise in the near future.

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

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

3.7.3 Limitations

Although both in the in-lab study and the deployment study gave evidence of benefit to PhraseFlow V2,there are undoubtedly important limitations to the PhraseFlow project. The search of effective phrase typing solutions is by no means exhaustive. There could be similar or even better UI and interaction designs than what I iterated to in PhraseFlow V2. One limitation of the deployment study was that I did not log quantitative data because of the privacy concern, which prevented us from gaining more insights on the realistic usage. PhraseFlow mainly focused on tap typing in English language. Hence, the lessons learnt from PhraseFlow might not apply to all languages. For some languages, the relevant context might be three or more words away, which requires larger language models. For

languages like Chinese, phrase-level decoding is already incorporated into the Pinyin typing methods, but after the decoding, extra steps are required to commit the candidates. During my study, some participants suggested it would be desirable if the keyboard could correct multiple languages together, which was also beyond the scope of this work. Finally, I started but did not get very far in phrase level gesture typing. Gesture typing decodes at whole word level to begin with [267] and does not have the error prone space key operations for both space insertion and word commit.

3.8 Conclusion

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

Chapter 4

TYPE, THEN CORRECT: INTELLIGENT TEXT CORRECTION TECHNIQUES FOR MOBILE TEXT ENTRY USING NEURAL NETWORKS

Current text correction processes on mobile touch devices are laborious: users either extensively use backspace, or navigate the cursor to the error position, make a correction, and navigate back, usually by employing multiple taps or drags over small targets. In this paper, I present three novel text correction techniques to improve the correction process: *Drag-n-Drop*, *Drag-n-Throw*, and *Magic Key*. All of the techniques skip error-deletion and cursor-positioning procedures, and instead allow the user to type the correction first, and then apply that correction to a previously committed error. Specifically, *Drag-n-Drop* allows a user to drag a correction and drop it on the error position. *Drag-n-Throw* lets a user drag a correction from the keyboard suggestion list and "throw" it to the approximate area of the error text, with a neural network determining the most likely error in that area. *Magic Key* allows a user to type a correction and tap a designated key to highlight possible error candidates, which are also determined by a neural network. The user can navigate among these candidates by directionally dragging from atop the key, and can apply the correction by simply tapping the key. We evaluated these techniques in both text correction and text composition tasks. The results show that correction with the new techniques was faster than de facto cursor and backspace-based correction. The techniques apply to any touch-based text entry method.[1]

---

[1]This chapter is adapted from: Mingrui "Ray" Zhang, He Wen, Jacob O. Wobbrock. (2019). Type, Then Correct: Intelligent Text Correction Techniques for Mobile Text Entry Using Neural Networks. Proceedings of the 32nd Annual ACM Symposium on User Interface Software & Technology (UIST '19). New York: ACM Press.

4.1 Introduction

Text entry techniques on touch-based mobile devices today are generally well developed. Ranging from tap-based keyboard typing to swipe-based gesture typing [267], today's mobile text entry methods employ a range of sophisticated algorithms designed to maximize speed and accuracy. Although the results reported from various papers [181,226] show that mobile text entry can reach reasonably high speeds, some even as fast as desktop keyboards [226], the daily experience of mobile text composition is still often lacking. One bottleneck lies in the text correction process. On mobile touch-based devices, text correction often involves repetitive backspacing and moving the text cursor with repeated taps and drags over very small targets (*i.e.*, the characters and spaces between them). Owing to the fat finger problem [227], this process can be slow and tedious indeed. In this chapter, I will introduce a project that applies techniques in Natural Language Processing (NLP) to improve the text correction interaction for touch screen text entry.

Correcting text is a consistent and vital activity during text entry. A study by MacKenzie and Soukoreff showed that backspace was the second most-common keystroke during text entry (pp. 164-165) [139]. Dhakal et al. [54] found that during typing, people made 2.29 error corrections per sentence, and that slow typists actually made and corrected more mistakes than the fast typists.

For immediate error corrections, *i.e.*, when an error is noticed right after it is made, the user can press backspace to delete the error [202]. However, for overlooked error corrections, the current cursor movement-based text correction process on smartphones is laborious: one must navigate the cursor to the error position, delete the error text, re-enter the correct text, and finally navigate the cursor back. There are three ways to position the cursor: 1) by repeatedly pressing the backspace key [202]; 2) by pressing arrow keys on some keyboards or making gestures such as swipe-left; 3) by using direct touch to move the cursor. The first two solutions are more precise than the last one, which suffers from the fat finger problem [227], but they require repetitive actions. The third option is error-prone when positioning the cursor amidst small characters, which increases the possibility of cascading

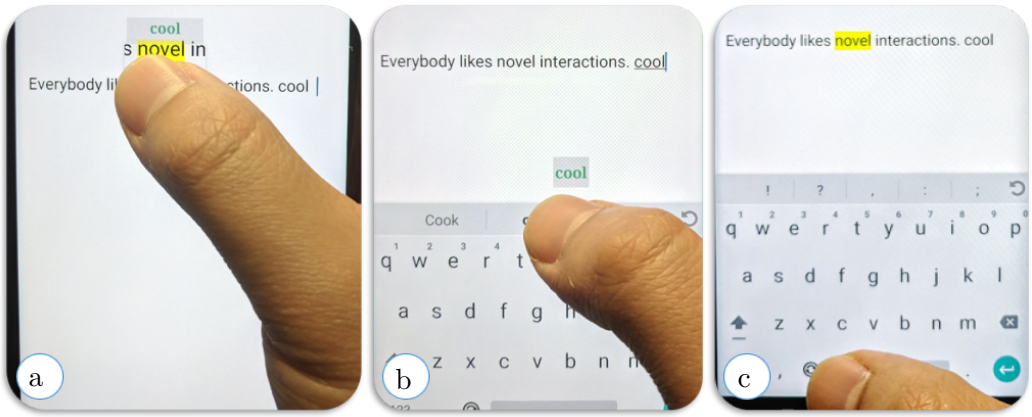

Figure 4.1: The three correction interactions of Type, Then Correct: (a) *Drag-n-Drop* lets the user drag the last word typed and drop it on an erroneous word or gap between words; (b) *Drag-n-Throw* lets the user drag a word from the suggestion list and flick it into the general area of the erroneous word; (c) *Magic Key* highlights each possible error word after the user types a correction. Directional dragging from atop the magic key navigates among error words, and tapping the magic key applies the correction

errors [12]; it also increases the cognitive load of the task and takes on average 4.5 seconds to perform the tedious position-edit-reposition sequence [72].

The project in this chapter is based on the premise: 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?* "Type, Then Correct" (TTC) contains three interactions (Fig. 4.1): 1) *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 [247]. 2) *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 and automatically corrects it. 3) *Magic Key* does not require direct interaction with the text input area at all. After typing a correction, the user simply presses

a dedicated key on the keyboard, and the deep learning algorithm highlights possible errors according to the typed correction. The user could then dragging atop the key to navigate through the error candidates and tap the key again to apply the correction. All three of my interaction techniques require no movement of the text cursor and no use of backspace.

To understand users' reactions to the techniques, I evaluated them in two text entry tasks: (1) a text correction task, and (2) a text composition task. I compared the three techniques with the de facto cursor-movement and backspace-based method in use on smartphones today. The results reveal that the two "intelligent" techniques, Drag-n- Throw and Magic Key, result in faster correction times, and were preferred over the de facto technique. Moreover, my methods do not conflict with existing gestural interactions, including gesture typing, and therefore can be used on any touch-based device including smartphones and tablets.

4.2 The Three Interactions

I present the design and implementation of the three interaction techniques of Type, Then Correct (TTC). The common features of these interactions are: 1) the first step is always to type the correction text at the current cursor position, usually the end of the current input stream; 2) all correction interactions can be undone by tapping the undo key on the keyboard (Fig. 4.2, top right); 3) after a correction is applied, the text cursor remains at the last character of the text input stream, allowing the user to continue typing without having to move the cursor. A current, but not theoretical, limitation is that I only allow the correction text to be contiguous alphanumeric text without special characters or spaces.

4.2.1 Drag-n-Drop

Drag-n-Drop is the simplest interaction technique. With *Drag-n-Drop*, after typing the correction, the user then drags the correction text and drops it on the error location. As shown in Fig. 4.3a.1, if the finger's touchdown point is within the area of the last word, the correction procedure will be initiated. The user can then move the correction and drop it either on another word to substitute it, or on a space to insert the correction.

While moving the correction, a magnifier appears above the finger to provide an enlarged

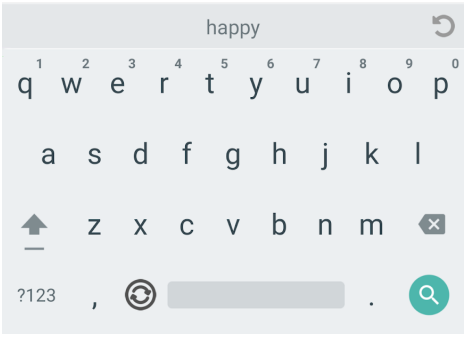

Figure 4.2: The customized keyboard interface. The undo key is located in the top-right corner. The *Magic Key* is the circular key immediately to the left of the space bar

image of the touched text; text to be corrected will be highlighted in a yellow background (Fig. 4.3a.2). When the finger drags over an alphanumeric character, I highlight its surrounding text bounded by any special character or space. When the finger drags over a space character, I highlight the single space character. The correction text also displays above the magnifier during the drag to remind the user what the correction is.

Similar to Shift by Vogel and Baudisch [227], I adjusted the input point to 30 pixels above the actual contact point, to reflect the user's perceived input point [95]. Vogel and Baudisch suggested that "users perceived the selection point of the finger as being located near the top of the finger tip" [23, 227], while the actual touch point was roughly at the center of the finger contact area [189], as shown in Fig. 4.4. After the correction is dropped on a space (for insertion) or on a word (for substitution), there is an animated color change from orange to black, confirming the successful application of the correction text.

4.2.2 Drag-n-Throw

Similar to *Drag-n-Drop*, *Drag-n-Throw* also requires the user to drag the correction. But unlike *Drag-n-Drop*, with *Drag-n-Throw*, the user flicks the correction from the word suggestion list atop the keyboard, not from the text area, allowing the user's fingers to stay near the keyboard area. As before, the correction text shows above the touch point as a

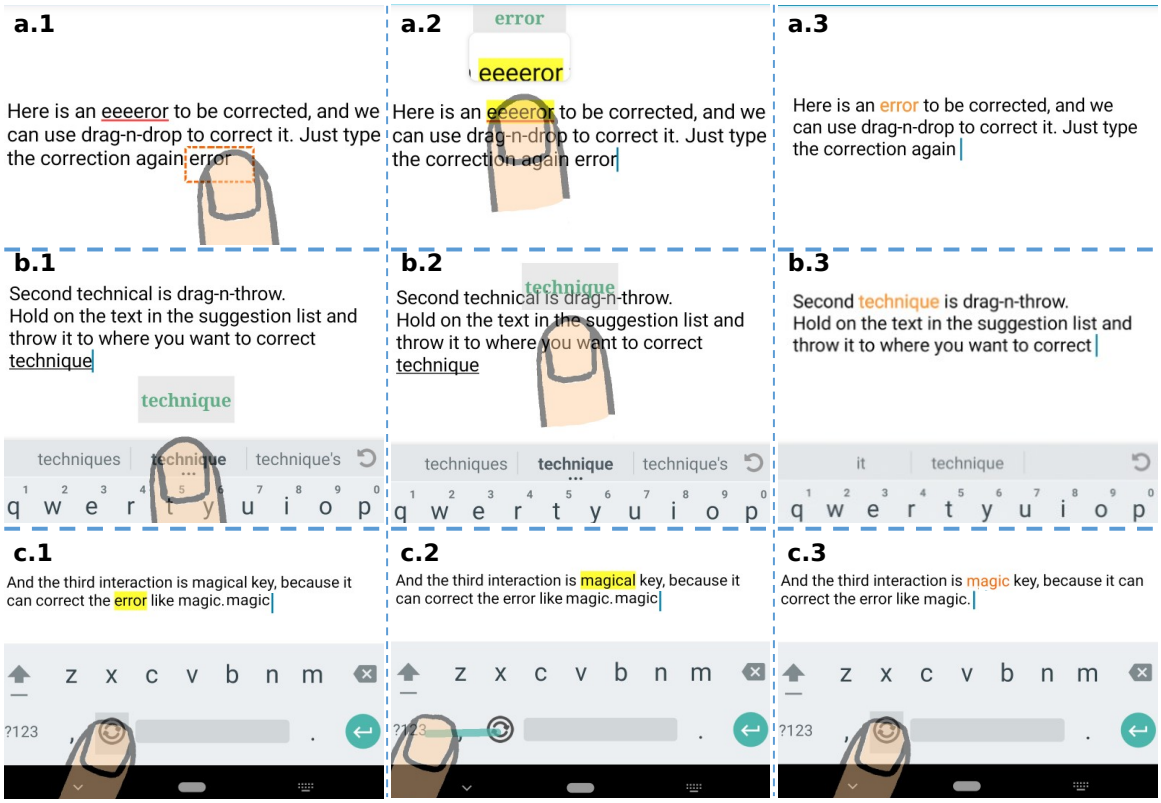

Figure 4.3: The three interaction techniques. *Drag-n-Drop*: (a.1) Type a word and then touch it to initiate correction; (a.2) Drag the correction to the error position. Touched words are highlighted and magnified, and the correction shows above the magnifier; (a.3) Drop the correction on the error to finish. *Drag-n-Throw*: (b.1) Dwell on a word from the suggestion list to initiate correction. The word 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 determines 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 potential error. (c.2) Drag left from atop the magic key to highlight the next possible error in that direction. Now, "magical" is highlighted. (c.3) Tap the magic key again to commit the correction "magic"

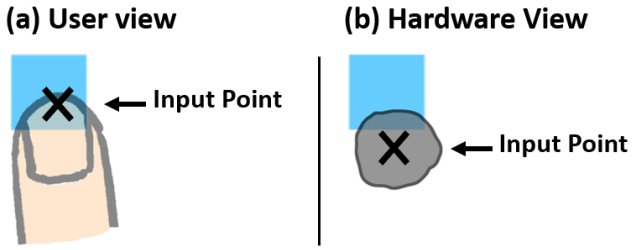

Figure 4.4: Perceived input point: (a) the user views the top of the fingernail as the input point [95]; (b) but today's hardware regards the center of the contact area as the touch input point, which is not the same. Figure adapted from [227]

reminder (Fig. 4.3b.1). Instead of dropping the correction on the error position, the user throws (*i.e.*, flicks) the correction to the general area of the text to be corrected. Once the correction is thrown, my deep learning algorithm determines the error position, and corrects the error either by substituting the correction for a word, or by inserting the correction. Color animation is displayed to confirm the correction. The procedure is shown in Fig. 4.3b.1-3.

I enable the user to drag the correction from the suggestion list because it is quicker and more accurate than directly interacting with the text, which has smaller targets. Moreover, my approach provides more options and saves time because of the word-completion function. For example, if the user wants to type "dictionary", she can just type "dic" and "dictionary" appears in the list. Or, if the user misspells "dictonary", omitting an "i", the correct word still appears in the list because of the keyboard's decoding algorithm.

Throwing the text speeds up the interaction beyond precise drag-and-dropping. The user does not have to carefully move the finger to drop the correction. The deep learning algorithm will output candidate positions for substitution or insertion within the general finger-up area. (More implementation details are explained below.) In my implementation, if the candidate is within 250 pixels of the finger-lift point in any direction, the error will be corrected and confirmed by color animation. Otherwise, there will be no effect. The

250-pixel threshold was derived empirically from iterative trial-and-error. Larger thresholds allow corrections too far away from the finger-lift point, which can cause unexpected results and user frustration. Smaller thresholds reduce the benefits of "throwing" and eventually start to feel like "dropping."

4.2.3 Magic Key

Drag-n-Drop required interaction within the text input area; Drag-n-Throw kept the fingers closer to the keyboard but still required some interaction in the text input area. With Magic Key, the progression "inward" toward the keyboard is fulfilled, as the fingers do not interact with the text input area at all, never leaving the keyboard. Thus, round-trips [69] between the keyboard and text input area are eliminated.

With Magic Key, after typing the correction, the user taps the magic key on the keyboard (Fig. 4.3c.1), and the possible error text is highlighted. If a space is highlighted, an insertion is suggested; if a word is highlighted, a substitution is suggested. The nearest possible error to the just-typed correction will be highlighted first; if it is not the desired correction, the user can drag from atop the magic key to left to show the next possible error. The user can drag left or right from atop the magic key to rapidly navigate among different error candidates. Finally, the user can tap the magic key to commit the correction. The procedure is shown in Fig. 4.3c.1-3. To cancel the operation, the user can simply tap any key (other than undo or the magic key itself).

4.3 The Correction Algorithm

In this section, I present the deep learning algorithm for text correction and its natural language processing (NLP) model, the data collection and processing procedures, and the training process and validation results.

4.3.1 Expected Correction Categories

I first list error types that my model should correct:

Typos A typographical error ("typo") happens when a few characters of a word are mistyped.

For example, *fliwer (flower)* or *feetball (football)*. Among typos, misspellings can usually be auto-corrected by current keyboards; however, auto-correction might yield another wrong word. For example, *best (bear)* or *right (tight)*. The model should be able to handle different types of typo errors.

Grammar Errors Grammar errors caused by one mistaken word should be corrected, such as misuse of verb tense, lack of articles or pronouns, subject-verb disagreement, etc.

Semantic Substitution The model should also be able to substitute words that are semantically related to the correction, such as synonyms and antonyms. For example, "what a nice day" can be corrected to "what a beautiful day". Semantic substitution is not necessarily correcting an error, but is useful when the user wants to change the expression.

4.3.2 The Deep Neural Network Structure

Inspired by Xie et al. [252], I applied a recurrent neural network (RNN) encoder-decoder model similar to the translation task for text corrections. The encoder contains a character-level convolutional neural network (CNN) [112] and two bi-directional gated recurrent unit (GRU) layers [38]. The decoder contains a word-embedding layer and two GRU layers. The overall flow of the model is shown in Fig. 4.5, and the encoder-decoder structure is shown in Fig. 4.6 [2].

Traditional recurrent neural networks (RNN) cannot output positional information. The key insight is that instead of outputting the whole error-free sentence, I make the decoder only output five words around the proposed correction position, *e.g.*, the correction word and its four neighboring words (two before, two after). If there are not enough words, the decoder will output the flags *<bos>* or *<eos>* instead for beginning-of-sentence and end-of-sentence, respectively. To locate the correction position, I compare the output with the input sentence word-by-word, and choose the position that aligns with most words. For the example in Fig. 4.5, I first tokenize the input and add two *<bos>* and two *<eos>* to the start and end of the tokens. Then I compare the output with the input:

[2]The model and data processing codes are available at `https://github.com/DrustZ/CorrectionRNN`

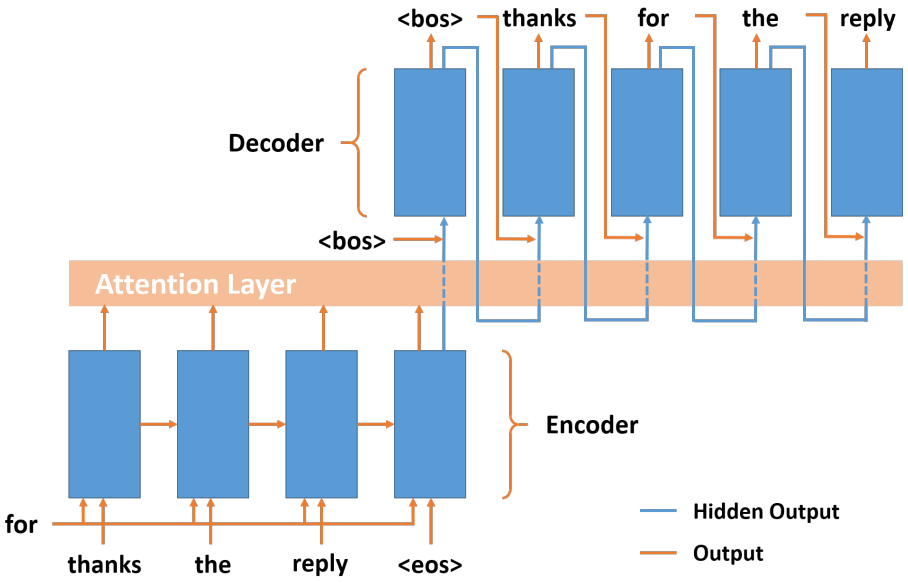

Figure 4.5: The encoder-decoder model for text correction. The model outputs five words in which the middle word is the correction. In this way, I get the correction's location

```
Input: <bos> <bos> thanks the reply <eos> <eos>
CS:          <bos> thanks for the   reply
CI:          <bos> thanks the reply
```

Above, "CS" means compare for substitution, which finds the best alignment for substitution (it uses all five words of the output trying to align with the input); "CI" means compare for insertion, which finds the best alignment for insertion (it only use the first and last two words of the output for alignment, as the center word is the insertion correction). In the example, CI has best alignment of four tokens (<*bos*>, thanks, the, reply), thus "for" will be inserted between "thanks" and "the". If the number of aligned tokens is the same in both comparisons, I would use insertion in my implementation.

I now explain the details of the encoder and the decoder (Fig. 4.6). For the encoder, because there might be typos and rare words in the input, operating on the character level is more robust and generalizable than operating on the word level. I first apply the character-level

CNN [112] composed of *Character Embedding, Multiple Conv. Layers* and *Max-over-time Pool layers* (Fig. 4.6, left). The character-level CNN generates an embedding for each word at the character level. The character embedding layer converts the characters of a word into a vector of $L \times E_c$ dimensions. I set E_c to 15, and fixed L to 18 in my implementation, which means the longest word can contain 18 characters (longer words are discarded). Words with fewer than 18 characters are appended with zeroes in the input vector. I then apply multiple convolution layers on the vector. After convolution, I apply max-pooling to obtain a fixed-dimensional (E_w) representation of the word. In my implementation, I used convolution filters with width [1, 2, 3, 4, 5] of size [15, 30, 50, 50, 55], yielding a fixed vector with the size of 200. E_c was set to 200 in the decoder.

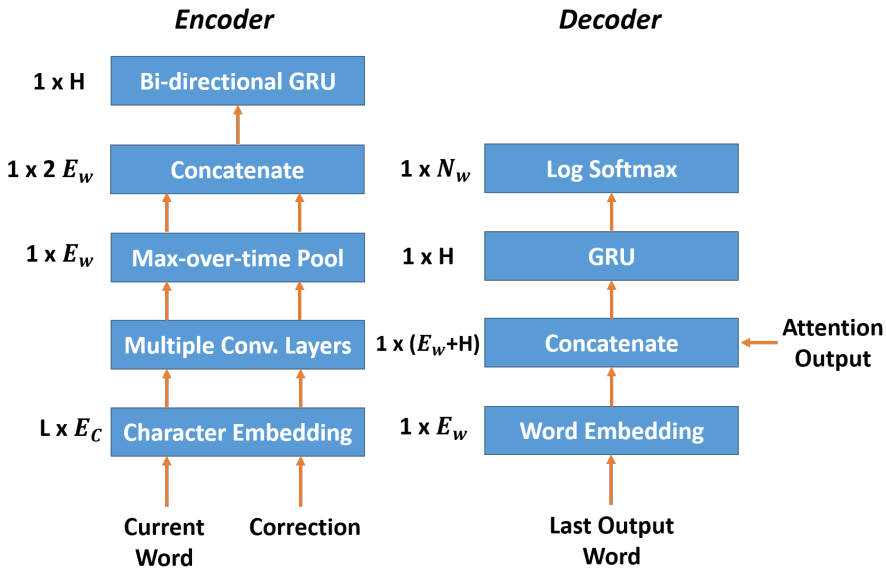

Figure 4.6: Illustration of the encoder and decoder, which is every vertical blue box in Fig. 4.5. L is the length of characters in a word; E_c is the character embedding size; H is the hidden size; E_w is the word embedding size; N_w is the word dictionary size

I also needed to provide the correction information for the encoder. I achieved this by feeding the correction into the same character-level CNN, and concatenated the correction embedding with the embedding of the current word. This yielded a vector of size $2E_w$,

which was then fed into two bi-directional GRU layers. The hidden size H of GRU was set to 300 in encoder and decoder.

The decoder first embedded the word in a vector of size E_w, which was set to 200. Then it was concatenated with the attention output. I used the same attention mechanism as Bahdanau et al. [16]. Two GRU layers and a log-softmax layer then followed to output the predicted word.

4.3.3 Data Collection and Processing

I used the *CoNLL 2014 Shared Task* [161] and its extension dataset [32] as a part of the training data. The data contained sentences from essays written by English learners with correction and error annotations. I extracted the errors that were either insertion or substitution errors. In all, I gathered over 33,000 sentences for training.

To gather even more training data, I perturbed several large datasets containing normal text. I used the Yelp reviews (containing two million samples) and part of the Amazon reviews dataset (containing 130,000 samples) generated by Zhang et al. [277]. I treated these review data as if they were error-free texts, and applied artificial perturbation to them. Specifically, I applied four perturbation methods:

1. **Typo simulation**. In order to simulate a real typo, I applied the simulation method similar to Fowler et al. [70]. The simulation treated the touch point distribution on a QWERTY layout as a 2-D Gaussian spatial model, and selected a character based on the sampling coordinates. I used the empirical parameters of the spatial model from Zhu et al. [280]. For each sentence in the review dataset, I randomly chose a word from the sentence, and simulated the typing procedure for each character of the word until the simulated word was different from the original word. I then applied a spellchecker to "recover" the typo. This maneuver was to simulate the error of auto-correction functions, where the "corrected" word actually becomes a different word. I then used the "recovered" word as a typo if it was different from the original word, or used the typing simulation result if the spellchecker successfully recovered the typo.

2. **Dropping words** To enable the model to learn about insertion corrections, I randomly dropped a word from a sentence, and labelled the dropped word as the correction. I prioritized dropping common stop words first if any of them appeared in the sentence, because people were most likely to omit words like a, the, my, in, and very.

3. **Word deformation** I randomly changed or removed a few characters from a word. If the word was a verb, I would replace it with a word sharing the same lexeme. For example, I would pick one of broken, breaking, breaks, or broke to replace break. If the word was a noun, I would use a different singular or plural word. For example, I would replace star with stars. Otherwise I would just remove a few characters from the word.

4. **Semantic word substitution** This perturbation enabled the model to learn semantic information. For a given word in the sentence, I looked for words that were semantically similar to it and made a substitution. I used the GloVe [170] Twitter-100 model from Gensim [176] to represent similarity. Synonyms and antonyms were generated using this method.

For each sentence in the review data set, I randomly applied a perturbation method. I then combined the perturbed data with the CoNLL data, and filtered out sentences containing less than 3 words or more than 20 words. In all, the final training set contained 5.6 million phrases.

For testing, I used two datasets: *CoNLL 2013 Shared Task* [161], which was also a grammatical error-correction dataset, and the Wikipedia revision dataset [264], which contains real-word spelling errors mined from Wikipedia's revision history. I generated 1665 phrases from the CoNLL 2013 dataset, and 1595 phrases from the Wikipedia dataset.

4.3.4 Training Process

I implemented the model in PyTorch [168]. I only included lowercase alphabetical letters (*a-z*) and ten numerals (0-9) in the character vocabulary of the encoder. I used the Adam optimizer with a learning rate of .0001 (1e-4) for the encoder and .0005 (5e-4) for the

decoder, and a batch size of 128. I applied weight clipping of $[-10, +10]$, and a teacher forcing ratio of 0.5. I also used dropout with probability 0.2 in all GRU layers. For the word embedding layer in the decoder, I labeled words with frequencies less than 2 in the training set as *<unk>* (unknown).

4.3.5 Results

Table 4.1 shows the evaluation results on the two testing datasets. The recall is 1 because all the testing data contained errors. I regarded a prediction as correct if the error position predicted was correct using the comparison algorithm described above.

Table 4.1: The performance of my correction model on the two testing data sets

Dataset	Accuracy
CoNLL 2013	75.68%
Wikipedia Revisions	81.88%

4.3.6 Other Implementation Details

I developed a custom Android keyboard and a notebook application to implement the three text correction interaction techniques. The keyboard was based on the Android Open Source Project (AOSP)[3] from Google. In building on top of this keyboard, I added the long-press interaction on suggested words for *Drag-n-Throw*.

The notebook application was built on an open-source project Notepad[4], and most of the interactions were implemented as part of the notebook application. For *Drag-n-Drop*, when a user touched within the last word area (within 100 pixels of the *(x, y)* coordinate of the last character), the interaction was initiated. I used the default magnifier on the Android system and added a transparent view showing the correction above the finger as it moves.

[3] `https://android.googlesource.com/platform/packages/inputmethods/LatinIME/`

[4] `https://github.com/farmerbb/Notepad`

For *Drag-n-Drop* and *Magic Key*, the keyboard needed to communicate with the notebook application. The keyboard used the Android Broadcast mechanism to send the correction and endpoint of the throw gesture of *Drag-n-Throw*. When the information was received, the notebook would search within the three lines near the release point. For each line, the notebook extracted up to sixty surrounding characters near the release point, and sent them to a server running the correction model. The server then replied with possible correction options and the corresponding probabilities. The notebook then selected the most likely option to update the correction. To avoid the correction happening too far away from the throwing endpoint, I constrained the x-coordinate of the correction to be within 250 pixels of the finger-lift endpoint.

For the *Magic Key* technique, the keyboard notified the notebook when the magic key was pressed or dragged. The notebook would treat the last word typed as the correction, and sent the last 1000 characters to the server. The server then split the text into groups of 60 characters with overlaps of 30 characters, and predicted a correction for each group. When the notebook received the prediction results, it first highlighted the nearest option, and then switched to further error options when the key was dragged left. For substitution corrections, it would highlight the whole word to be substituted; for insertion corrections, it would highlight the space where the correction was to be inserted.

The server running the correction model handled responses via HTTP requests. To increase the accuracy of the model for typos, I first calculated the matching score between each token of the input text and the correction using the Levenshtein algorithm [125]. The score equaled the number of matches divided by the total character number of the two words. If the score of a word in the sentence was above 0.75, I treated the word as the error to be corrected. Otherwise, I fed the text and correction into the aforementioned neural network model.

4.4 Experiment

We evaluated three aspects of the correction techniques: (1) timing and efficiency; (2) success rate of *Drag-n-Throw* and *Magic Key*; and (3) users' subjective preferences. We conducted an experiment containing two tasks: a correction task and a composition task.

The correction task purely evaluated the efficiency of the interactions, and the composition task evaluated the usability and success rate of the intelligent techniques in more realistic scenarios.

4.4.1 Participants

We recruited 20 participants (8 male, 12 female, aged 23-52) for the study. We used emails, social media and word-of-mouth for recruitment. All participants were familiar with entering and correcting text on mobile devices. The experiment lasted one hour, and participants were compensated $20 USD for their time.

4.4.2 Apparatus

A Google Pixel 2 XL was used for the study. The phone had a 6.0" screen with a 1440×2880 resolution. We added logging functions from the notebook application to record correction time. The server running the correction model had a single GTX 1080.

4.4.3 Phrases Used in the Correction Task

Both tasks utilized a within-subjects study design. For the correction task, we chose 30 phrases from the test dataset on which the correction model had been 100% correct because we wanted purely to evaluate the performance of the interaction technique, not of the predictive model. We split the phrases evenly into three categories: *typos, word changes,* and *insertions*. *Typos* required replacement of a few characters in a word; *word changes* required replacing a whole word in a phrase; and *insertions* required inserting a correction. For each category, we had five *near-error* phrases where the error positions were within the last three words; and five *far-error* phrases where the error positions were farther away. The reason was to see whether error positions would affect correction efficiency. Examples of phrases in each category are provided in the Appendix A.

4.4.4 Procedure

Participants were first introduced to the different interaction techniques, including the categories of errors that *Drag-n-Throw* and *Magic Key* were able to correct. Then participants practiced the three techniques with three practice phrases each. After practicing, the 30 phrases as well as their corresponding corrections were presented to the participants. Then they began to correct the phrases using four techniques: (1) today's *de facto* cursor-positioning and backspace-based method, (2) *Drag-n-Drop*, (3) *Drag-n-Throw*, and (4) *Magic Key*. The order of the four techniques was counterbalanced using a balanced Latin Square.

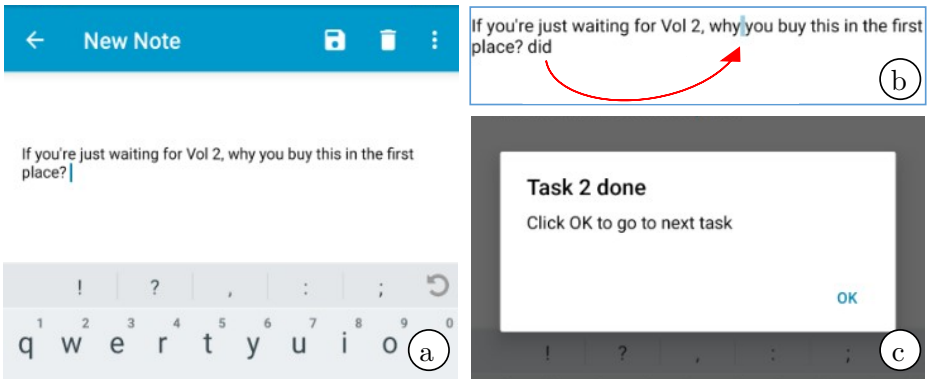

Figure 4.7: (a) The notebook application showing the test phrase. (b) The intended correction displayed on the computer screen. (c) After each correction, a dialog box appeared

When the correction task started, the participant would be shown the next phrase to be corrected on a desktop computer screen, as well as how to correct it. The notebook application would display the phrase with an error. After the participant corrected the error, a dialog box appeared asking the participants to enter the next phrase. The experimenter then showed the next phrase on the computer. When the participant was ready, she entered the next phrase by tapping the OK button. The interface is shown in Fig. 4.7. The rationale for showing how to correct the phrase on a computer screen was to filter out the learning effect and visual search time caused by the unfamiliarity of the phrases, and to isolate the interaction time.

After the correction task, the participants started composing messages freely. They were told to type for three minutes as they would in normal messaging situations. However, they were told not to correct any errors during typing. After finishing their compositions, they then corrected all errors with the four interaction techniques. This composition task endeavored to evaluate usability in a more realistic scenario. When participants were correcting errors with *Drag-n-Throw* and *Magic Key*, the experimenter recorded whether any failure happened in order to calculate the error rate.

When the two tasks ended, participants filled out a NASA-TLX survey [182] and a usability survey adapted from the SUS questionnaire [30] for each interaction.

4.5 Results

For the correction task, 2400 phrases were collected in total. For the correction task, I focus on task completion times; for the composition task, I focus on the success rate of the two intelligent interaction techniques and users' preferences.

4.5.1 Correction Time

Fig. 4.8 shows correction times for the four techniques. In addition to overall times, the correction times for *near-error* and *far-error* phrases are also shown. I log-transformed correction times to comply with the assumption of conditional normality, as is often done with time measures [129]. I used linear mixed model analyses of variance [71, 130], finding that there was no order effect on correction time ($F(3, 57) = 1.48, n.s.$), confirming that my counter-balancing worked. Furthermore, Technique had a significant effect on correction time for all phrases ($F(3, 57) = 26.49, p < .01$), *near-error* phrases ($F(3, 57) = 29.02, p < .01$), and *far-error* phrases ($F(3, 57) = 17.04, p < .01$), permitting me to investigate *post hoc* pairwise comparisons.

I performed six *post hoc* paired-samples t-tests with Holm's sequential Bonferroni procedure [94] to correct for Type I error rates, finding that for all phrases, the *de facto* cursor-based method was significantly slower than *Drag-n-Throw* ($t(19) = 6.66, p < .01$) and *Magic Key* ($t(19) = 4.79, p < .01$); *Drag-n-Drop* was also significantly slower than *Drag-n-Throw*

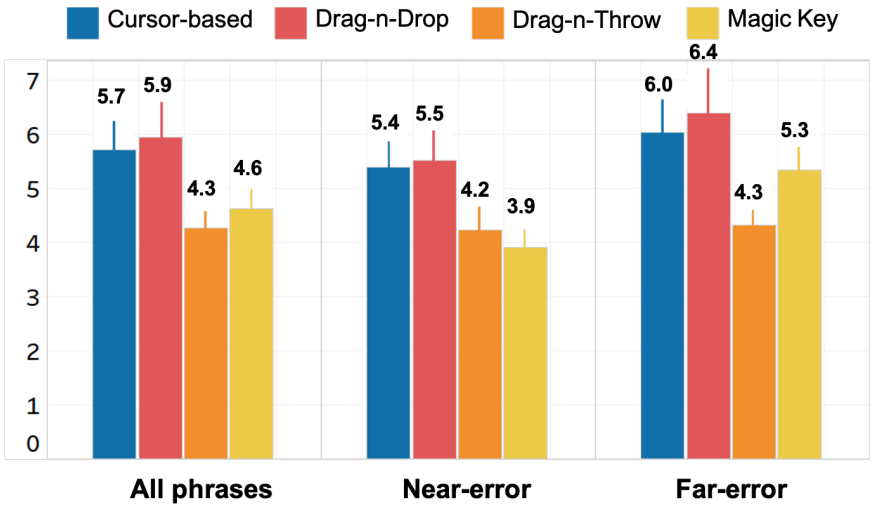

Figure 4.8: Average correction times in seconds for different interaction techniques (lower is better). *Drag-n-Throw* was the fastest for all phrases and far-error phrases, while *Magic Key* was the fastest for near-error phrases. Error bars are +1 SD

($t(19) = 7.49, p < .01$) and *Magic Key* ($t(19) = 5.62, p < .01$). For near-error phrases, the *de facto* method was significantly slower than *Drag-n-Throw* ($t(19) = 5.58, p < .01$) and *Magic Key* ($t(19) = 7.02, p < .01$); *Drag-n-Drop* was also significantly slower than *Drag-n-Throw* ($t(19) = 5.00, p < .01$) and *Magic Key* ($t(19) = 7.44, p < .01$). For far-error phrases, *Drag-n-Throw* was significantly faster than all other interactions: the *de facto* method ($t(19) = -5.64, p < .01$), *Drag-n-Drop* ($t(19) = -6.60, p < .01$), and *Magic Key* ($t(19) = -3.68, p < .01$).

I then looked at different correction types. Fig. 4.9 shows the average correction times for *typos*, *word changes*, and *insertions*. Again, I used linear mixed model analyses of variance [71, 130] on log correction time [129]. Technique had a statistically significant effect for all correction types: *typo* ($F(3, 57) = 5.11, p < .01$), *word change* ($F(3, 57) = 10.87, p < .01$) and *insertion* ($F(3, 57) = 55.55, p < .01$).

I then performed *post hoc* paired-samples t-tests with Holm's sequential Bonferroni procedure, finding that for *typos*, the *de facto* cursor-based method was significantly slower

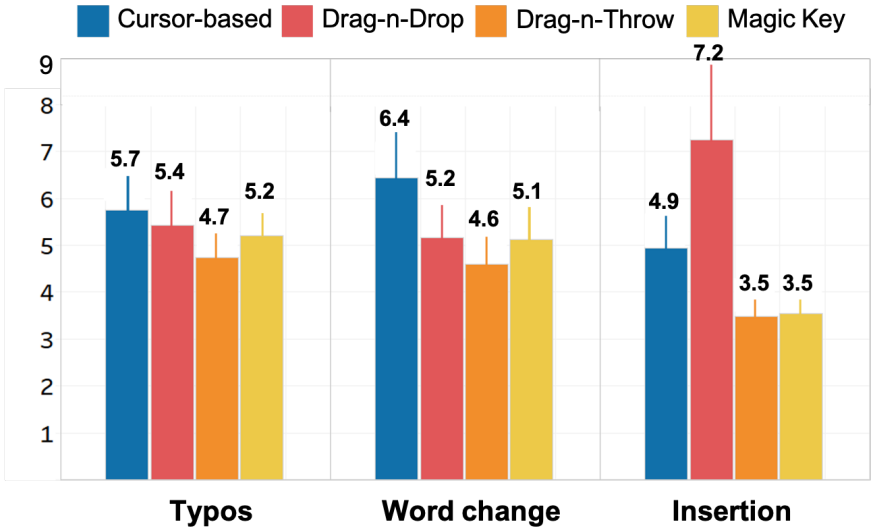

Figure 4.9: Average correction times in seconds for different correction types (lower is better). *Drag-n-Throw* was the fastest for all three types. Error bars are +1 SD

than *Drag-n-Throw* ($t(19) = 3.80, p < .01$); *Drag-n-Drop* was also significantly slower than *Drag-n-Throw* ($t(19) = 2.70, p < .05$). For *word change*, the *de facto* cursor-based method was significantly slower than all other techniques: *Drag-n-Drop* ($t(19) = 3.54, p < .01$), *Drag-n-Throw* ($t(19) = 5.58, p < .01$), and *Magic Key* ($t(19) = 3.74, p < .01$). For *insertion*, *Drag-n-Drop* was significantly slower than all other interactions: the *de facto* method ($t(19) = 5.72, p < .01$), *Drag-n-Throw* ($t(19) = 11.17, p < .01$), and *Magic Key* ($t(19) = 10.92, p < .01$); also, the *de facto* cursor-based method was significantly slower than *Drag-n-Throw* ($t(19) = 5.45, p < .01$) and *Magic Key* ($t(19) = 5.20, p < .01$).

4.5.2 Success Rate

In the text composition task, I recorded errors when participants were using *Drag-n-Throw* and *Magic Key*. With *Drag-n-Throw*, participants made 108 errors in all, and 95 of them were successfully corrected, a success rate of 87.9%. Among the successfully corrected errors, nine were attempted more than once because the corrections were not applied to expected error positions. With *Magic Key*, participants made 101 errors in all, and 98 of them were

successfully corrected, a success rate of 97.0%.

Figure 4.10: Composite usability (higher is better) and NASA TLX (lower is better) scores for different techniques. *Magic Key* was rated as the most usable and having the lowest workload

4.5.3 Subjective Preference

The composite scores of the SUS usability [30] and TLX [182] surveys for different interaction techniques are shown in Fig. 4.10. Participants generally enjoyed using *Magic Key* and *Drag-n-Throw* more than the *de facto* cursor-based method and *Drag-n-Drop*. Also, the two deep learning techniques were perceived to have lower workload than the other two.

4.6 Discussion

Drag-n-Throw performed fastest among different correction types. Moreover, its performance was not affected by whether the error was far away or not (Fig. 4.8). *Magic Key* also achieved reasonable speeds across different correction types. For *near-errors* within the last three words, it even surpassed *Drag-n-Throw*, because the errors would be highlighted and corrected with just two taps. For *far-errors*, participants had to drag atop the *Magic Key* a few times to highlight the desired error, leading to longer correction times. *Drag-n-Drop*

performed the slowest over all phrases, which was mainly caused by the insertion corrections. As Fig. 4.9 shows, it was faster than the de facto cursor-based method for typos and word changes, but significantly slower than other interactions for insertions. To insert a correction between two words, a user had to highlight the narrow space between those words. Many participants spent a lot of time adjusting their fingers in order to highlight the desired space. They also had to redo the correction if they accidentally made a substitution instead of an insertion. The undo key proved to be vital in such cases.

To evaluate the performance of the algorithm in more realistic scenarios, I analyzed the results from text composition tasks. *Drag-n-Throw* achieved a success rate of 87.9%. A failure was when two possible error candidates were too close to each other. For example, if the user wanted to insert "the" in the phrase "I left keys in room," there were two possible positions (before keys and before room), but only one of them would be corrected. *Magic Key* achieved a higher success rate of 97.0%, as it searched every possible error in the text.

As for participants' subjective preferences, 12 of 20 participants liked *Magic Key* the most. The major reason was convenience: all the actions were done on the keyboard. P1 commented, "Just one button handles everything. I don't need to touch the text anymore. It was also super intelligent. I am lazy, and that's why I enjoyed it so much." Another reason was that *Magic Key* provided feedback (highlights) before committing the correction, making the user confident about the target of their actions. As P4 pointed out, "It provides multiple choices, and the uncertain feeling is gone." The main critique of *Magic Key* was about the dragging interaction required to move among error candidates. P5 commented: "If the text is too long and the error is far away, I have to drag a lot to highlight the error. Also, the button is kinda small, and hard to drag."

Interestingly, I found that all three participants above age 40 had positive feedback about the two intelligent correction techniques, and negative feedback about the *de facto* cursor-based method. P14, aged 52, commented, "I dislike the cursor-based method most. I have a big finger, and it is hard to tap the text precisely. Throw is easy and works great. I also like *Magic Key*, because I don't need to interact with the text." Older adults are known to perform touchscreen interactions more slowly and with less precision than younger

adults [64], and the intelligent correction techniques might benefit them by removing the requirement of precise touch. Moreover, people walking on the street or holding the phone with one hand might also benefit from the interactions, because touching precisely is difficult in such situations.

4.6.1 Further Impact: JustCorrect

Type, Then Correct (TTC) inspired a follow-up work on mobile text correction interaction: *JustCorrect* [47], 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 4.11. The lab experiment showed that correction performed by JustCorrect were faster than TTC interactions.

4.7 Future Work

On the basis of my work here, I propose four possible future directions: (1) Punctuation handling: my current correction algorithm does not handle punctuation, so errors like “lets” (let’s) currently cannot be corrected. (2) Feedback for *Drag-n-Throw*: Participants felt unconfident when throwing corrections, as there was a lack of feedback as to where the corrections would be applied. Adding visual feedback such as highlighting surrounding the text of the throwing position might provide cues about where the correction will occur. (3) Better error-navigation interaction for *Magic Key*: The magic key itself was considered small and hard to drag. Better interactions to navigate through different error candidates should be explored, such as a swipe gesture on the keyboard itself. (4) Multilingual correction support: my interaction techniques could be applied to other languages as well, such as the Chinese language.

Figure 4.11: (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.

4.8 Conclusion

I presented three novel interaction techniques, *Drag-n-Drop*, *Drag-n-Throw*, and *Magic Key* for mobile text correction. The common concept in these three techniques was to type the correction and apply it to the error, without needing to reposition the text cursor or use backspace—maneuvers that together break the typing flow and slow touch-based text entry. I also developed deep learning algorithms for Drag-n-Throw and Magic Key using recurrent neural networks (RNN). my correction experiment showed that Drag-n-Throw and Magic Key were significantly faster than de facto cursor-based correction methods and garnered more positive user feedback. This work provides an example of how, by breaking from the desktop paradigm of arrow keys, backspacing, and mouse-based cursor positioning, I can rethink text entry on mobile touch devices and develop novel methods better suited to this paradigm.

Chapter 5

TYPEANYWHERE: A QWERTY-BASED TEXT ENTRY SOLUTION FOR UBIQUITOUS COMPUTING

In this chapter, I present a QWERTY-based text entry system, *TypeAnywhere*, for use in off-desktop computing environments. Using a wearable device that can detect finger taps, users can leverage their touch-typing skills from physical keyboards to perform text entry on any surface. TypeAnywhere decodes typing sequences based only on finger-tap sequences without relying on tap locations. To achieve optimal decoding performance, I trained a neural language model and achieved a 1.6% character error rate (CER) in an offline evaluation, compared to a 5.3% CER from a traditional n-gram language model. The user study showed that participants achieved an average performance of 70.6 WPM, or 80.4% of their physical keyboard speed, and 1.50% CER after 2.5 hours of practice over five days on a table surface. They also achieved 43.9 WPM and 1.37% CER when typing on their laps. The results demonstrate the strong potential of QWERTY typing as a ubiquitous text entry solution. [1]

5.1 Introduction

Computing is now almost ubiquitous [237], from wearables and Internet-of-Things devices to smartphones and virtual reality headsets, computers are woven into nearly every aspect of modern life. Since the earliest days of interactive computing, a fundamental method of communication with computers has been text entry. But as we move further off the desktop, off tablets, off smartphones, and indeed onto devices of almost every kind, satisfying text entry solutions remain elusive [268]. There is as-yet no unified text entry solution

[1] This chapter is adapted from: Mingrui "Ray" Zhang, Shumin Zhai, Jacob O. Wobbrock. (2022) TypeAnywhere: A QWERTY-Based Text Entry Solution for Ubiquitous Computing. Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '22). New Orleans, USA (April 30 - May 6, 2022). New York: ACM Press.

that is efficient, effective, and always available. Existing text entry solutions either are not mobile enough (e.g., a full-size hardware keyboard), or require the user to learn new typing skills (e.g., [121, 134, 207, 258]), or offer painfully low throughput (e.g., watch-based methods [37, 81, 247]). While speech-based input is sometimes an option, it is often not socially acceptable and can be a risk to privacy [152].

QWERTY-based touch-typing on physical keyboards remains the most common text input skill of computer users [54], utilizing multiple hands and fingers and relying on tacit knowledge of learned key positions. Although the QWERTY layout was not invented for optimal speed [50], the fact that people are so familiar with the layout has motivated text entry researchers to build upon QWERTY [26, 84, 226, 253, 254, 259]. We therefore want to leverage the physical QWERTY keyboard typing experience in my effort to discover a solution for ubiquitous computing text entry, but without presuming the availability of mechanical keys. What if we could type on an imaginary QWERTY keyboard on *any* surface, just like we type a physical keyboard?

To explore this question, my work developed *TypeAnywhere*, a QWERTY-based text entry solution for everyday use in ubiquitous computing contexts (Figure 5.1). TypeAnywhere employs wearable sensors on two hands that detect finger-tap actions, a decoder that converts tap sequences into text, and a corresponding interface for text editing. TypeAnywhere's hardware is the commercial *Tap Strap*[2] product, which uses accelerometers for tap detection. We feed a detected finger-tap sequence to a neural decoder modified from the BERT model [53], which then displays the output text on the typing interface. Similar to *Type, then Correct* [271], I also designed a text correction interaction for TypeAnywhere that avoids the need for cursor navigation.

An important feature of TypeAnywhere is that it does not rely on position information to decode the intended letters being typed by the user. Rather, TypeAnywhere relies only on the index of the finger being tapped and the context provided by the text entered thus far. Because TypeAnywhere only uses the index of the tapping finger without relying on (x, y)

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

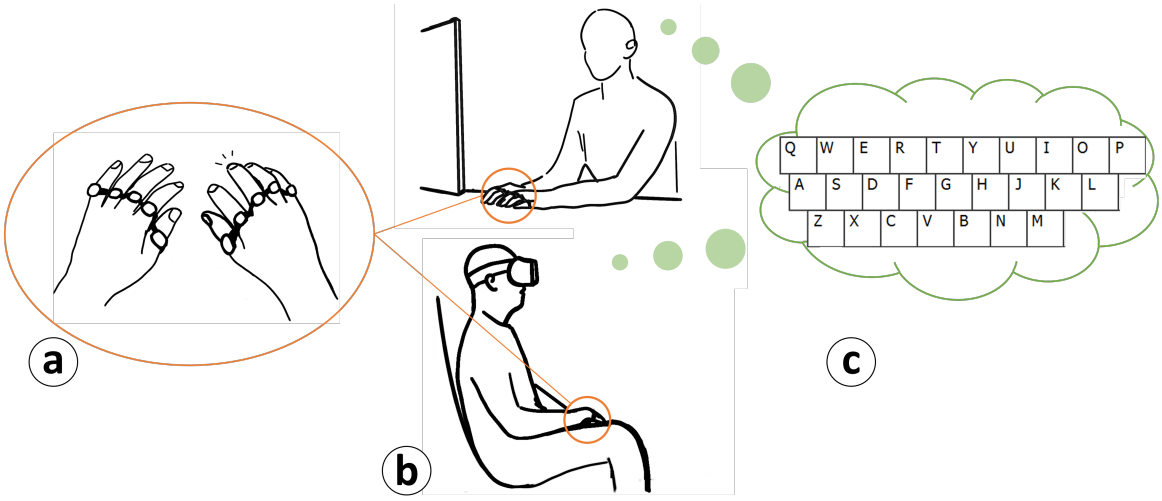

Figure 5.1: TypeAnywhere's core concepts. **(a)** Users wear lightweight devices on their fingers that detect finger taps. **(b)** With TypeAnywhere, users can type on any surface such as a tabletop or their lap. **(c)** Users type with their own QWERTY key-to-finger mappings, resembling physical keyboard typing.

position information, the user can perform the tap at any location, so long as they use the correct finger. This scheme simplifies the design space, enabling us to both generalize the decoder easily for different finger-to-key mappings, and to achieve high accuracy on text decoding. Unlike other QWERTY-based text entry research projects [65, 66, 257, 259, 280], which had to conduct user studies to collect spatial information for model-building, I was able to train the model without collecting data from user studies. Instead, I curated the training data by converting each letter in a phrase set to its corresponding finger ID. In this way, I was able to train the model on a large text corpus containing over 3.6 million samples [277] and adapt different finger-to-key mappings by altering the finger IDs in the training set. We performed computational evaluations on the neural decoder, achieving a 1.6% character error rate (CER) on the Cornell Movie-Dialogue Corpus [48], compared to a 5.3% CER using a conventional n-gram language model.

Without the tactile feedback provided by mechanical keys, whether users can effectively take advantage of TypeAnywhere's neural decoding power required us to conduct an empirical evaluation. So to evaluate TypeAnywhere, I performed a longitudinal user study with ten participants over five days. In my study, participants used TypeAnywhere on a table surface for a half-hour each day, achieving an average of 70.6 WPM and 1.50% CER. The best performer achieved 91.4 WPM and 3.15% CER after five days of use (or 2.5 total hours). To test the convenience of TypeAnywhere, participants also performed typing on their laps, reaching 43.9 WPM and 1.37% CER. my results demonstrate that users can quickly learn TypeAnywhere with minimal practice and achieve high performance after practicing for a total of 2.5 hours. As a result, I think TypeAnywhere has potential as a text entry solution for ubiquitous computing environments. To support TypeAnywhere's adoption, I provide an open-source implementation including my neural language model.[3]

5.2 Enabling the TypeAnywhere Experience

TypeAnywhere aims to provide users with a QWERTY-style typing experience similar to typing on a physical QWERTY keyboard, except that typing occurs on any surface without

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

the benefit of mechanical keys. The TypeAnywhere system contains the following building blocks: the hardware that detects finger taps, the neural language model that decodes the input sequence into text, and the interface that allows text composition and editing. In this section, I describe TypeAnywhere's interaction and user interface. In Section 5.3, I describe TypeAnywhere's neural language decoder.

5.2.1 Typing Interaction

Figure 5.2: The hardware used in TypeAnywhere. **(a)** The Tap Strap device. **(b)** The user wearing two Tap Strap devices.

TypeAnywhere's hardware interface comprises two wearable devices called *Tap Straps*,[4] which each have five connected rings that can be worn on each finger (Figure 5.2). Each ring contains a three-axis accelerometer that detects when a finger performs a tap action, the detection of which it is reported to have over 98% accuracy.[5] The intended purpose of Tap Strap is for one-handed text entry with a proprietary chording system.[6] As prior chording-based text entry systems have shown (e.g., [134]), chords can take considerable time to learn and master. The reported best performance using the default Tap Strap chording

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

[5]https://www.tapwithus.com/how-to-the-tap-strap-works/

[6]The default chording scheme of TapStrap is provided at `https://www.tapwithus.com/wp-content/uploads/2018/09/Tap-Alphabet-Glossary-2.pdf`

system is 62 WPM, which was achieved after three weeks' practice.[7] To the best of my knowledge, Tap Strap is the most affordable and accurate commodity device for finger-tap detection on the market. Because of Tap Strap's small form factor, users can wear the device comfortably. However, due to the requirement that users learn a proprietary one-handed chording scheme, Tap Strap's widespread adoption seems extremely unlikely absent efforts like ours to enable familiar QWERTY-style typing.

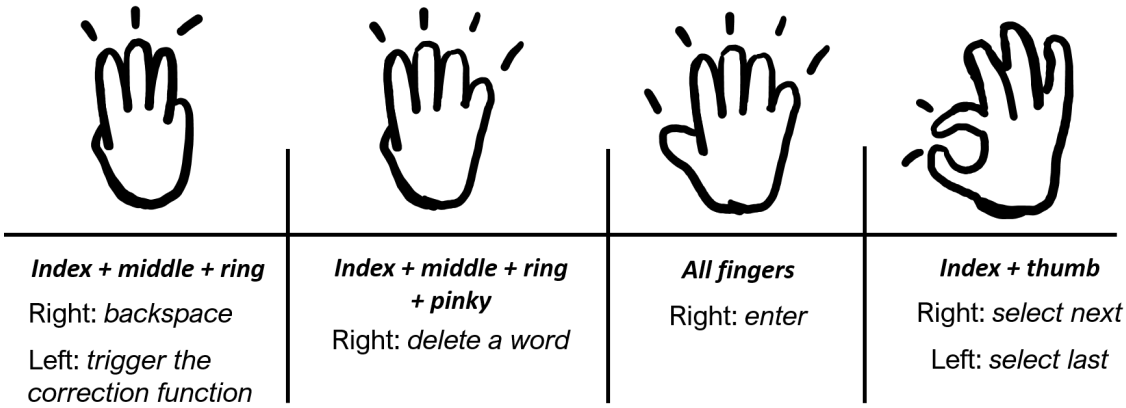

Figure 5.3: The tap-chords for text editing in TypeAnywhere. For the rightmost gesture, which performs candidate selection from a word list, the user can either tap the thumb and index finger, or perform a pinch gesture.

TypeAnywhere's typing interaction resembles the same typing interaction as one would use on a physical QWERTY keyboard. Specifically, users perform finger taps based on their own personal finger-to-key mappings, and the most likely character is produced. Users can perform finger taps on hard surfaces such as tabletops or walls, or on soft surfaces such as their stomachs or laps. The tap of the two thumbs is mapped to the spacebar, as most people use a thumb for spacebar pressing [66]. Essential text editing functions are also mapped to certain multi-finger chords, as shown in Figure 5.3. The backspace gesture was

[7]See `https://www.tapwithus.com/wpm-contest`. Note that this is advertised as a commercial contest and the reported performance is therefore the best achieved from among all customers.

the same as the default Tap Strap gesture, and I assigned symmetrical gestures to both hands to improve learnability [8]. We also assigned gestures only to neighboring fingers (for example, index + middle + ring) to make them easy to perform. We did not assign backspace to any specific finger because (1) each finger was already assigned with a set of letters, and (2) people were inconsistent with which finger they used to press backspace [62]. Assigning a backspace finger would add too much ambiguity to the typing process, as there was no clear indication of whether the user wanted to type a letter or backspace text.

5.2.2 Typing Interface

To evaluate TypeAnywhere's interaction, I designed a web app implemented in JavaScript as an evaluative user interface (Figure 5.4). There are four components of this interface: (1) the *test area*, which displays the target string for evaluation, used in lab studies only; (2) the *commit area*, which displays the committed text corresponding to the text view in apps; (3) the *compose area*, which contains the text that is being decoded; and (4) the *candidate list*, which displays the word candidates of the current typing sequence. Alternatively, the *compose area* could be integrated into the *commit area* by differentiating its content with underlines or different colors [273].

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

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

Figure 5.4: The TypeAnywhere web app. There are four main components in the interface. Note that the test area, redo button, and next button are for user study purposes only, and would not be present when just using TypeAnywhere.

example of the user's typing flow is depicted in Figure 5.5.

The user can also select a candidate for the currently typed word from a candidate selection list as shown in Figure 5.5d. If the user performs the selection gesture shown at the far right of Figure 5.3, the candidate selection function will be triggered and the candidate word will be highlighted in orange. The user can use the selection gesture to navigate through the candidates and then tap space to select the candidate, which copies it into the *compose area* (Figure 5.5e); or, the user can tap the backspace chord to cancel the candidate selection. (Candidate word lists raise the specter of out-of-vocabulary words. We describe how the user can enter such words in Section 5.2.4, below.)

5.2.3 Text Editing with "Type, then Correct"

As the user finger-taps with TypeAnywhere, the text in the *compose area* often changes, as the decoder continually processes entered finger-taps in light of surrounding textual context. At times, the text in the *compose area* will not be what the user intends, but by providing

Figure 5.5: The typing flow of TypeAnywhere. The user is typing the sentence "he is begging" with the standard finger-to-key mapping. **(a)** After typing the first word, the decoder outputs "me" as the most probable text. **(b)** With more input context, the decoder is able to decode the tap sequence to "he is". **(c)** The user first double-taps space to commit "he is" to the *commit area*, then types the last word decoded as "getting". **(d)** The user then performs a selection gesture to trigger candidate selection and navigate to "begging". **(e)** The word "begging" is now entered in the *compose area* after the user presses space.

more textual context, the resolved text might conform to the user's desires. Therefore, the user is encouraged to continue typing in TypeAnywhere, even when the currently decoded text is not (yet) what the user wants. However, in cases where further context does *not* resolve the decoded text to match the user's intentions, the user can edit the text; however, at this point, they will be well ahead of the error position. This situation can result in user uncertainty and editing inefficiencies that are not desirable in TypeAnywhere.

To address this issue—that is, letting users continue to type, which provides the decoder with

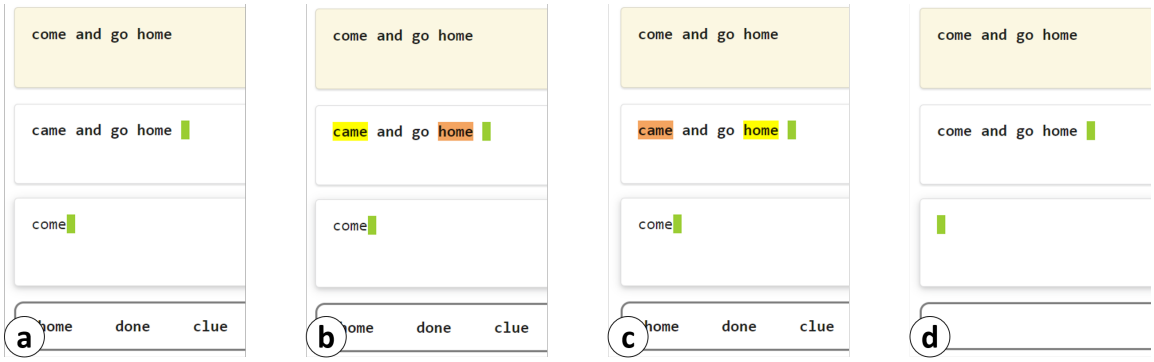

Figure 5.6: The correction interaction flow: **(a)** After committing the text containing the error word "came", the user types the correction "come" in the *compose area*. **(b)** The user then triggers the correction function. Possible errors are highlighted: both "came" and "home" have one finger-sequence edit distance with "come", hence the last word, "home", is highlighted in orange. **(c)** The user then employs the selection gesture to navigate between the candidates. **(d)** After the user presses the (virtual) spacebar, "came" is corrected to "come".

valuable additional context, but also avoids costly error correction situations—we employ the text correction scheme from *Type, then Correct* [47, 271], which lets the user type a correction at the *end* of their current text stream and then apply it directly and retroactively to a previous error without any cursor navigation. In the context of TypeAnywhere, if there are errors perhaps multiple words behind the current typing position, the user can commit all current *compose area* text to the *commit area*, and then type the corresponding correction in the *compose area*. The user can then tap the chord to trigger the correction function (Figure 5.3, left). The function compares the similarity of the correction word and the words in the *commit area* based on their finger-sequence edit distances, as described in the next paragraph. All words that are less than or equal to two edit-distance will be highlighted in yellow, with the shortest distance word being highlighted in orange. The user can then navigate through the possible errors and tap their virtual spacebar to commit the

correction. To cancel correction mode, the user simply taps their virtual backspace. The whole correction interaction flow is demonstrated in Figure 5.6.

Edit distance refers to the minimum number of character edit operations (substitute, insert, delete) to change one string into another [125]. In TypeAnywhere, because the text is decoded based on finger-tap sequences, I compare the finger-sequence edit distance to implement the correction function. For example, in the standard finger-to-key mapping, the word "far" (left index - left pinky - left index) has the same finger sequence as the word "rat", and has a finger-sequence edit distance of one with the word "get" (left index - left middle - left index). Note that in the finger-sequence edit distance calculation, along with substitutions, insertions, and deletions, I also added transpositions, thereby handling cases when finger-taps are interchanged, such as typing the sequence "information" as "infromation").

5.2.4 Out of Vocabulary Words

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

5.3 The Neural Language Decoder

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

5.3.1 Converting Text into Finger Sequences

Preparing the training data for the TypeAnywhere model is straightforward. As it only requires finger-tap sequences, I can reverse the decoding step to generate the training data: take the finger ID sequences corresponding to the character sequences in a corpus as the

network input data, and the character sequence itself as the label. I assigned a unique ID to each finger: 2 – 5 for pinky to index finger of the left hand, and 6 – 9 for index to pinky finger of the right, and 1 for the two thumbs. For example, if I am creating the training data for the standard finger-to-key mapping, the phrase "sunny day" would be converted to "366661426". Because the model does not need finger motion information, generalizing the data to other finger-to-key mappings is easy, as I just need to change the finger ID for certain characters. For people who type the same letter with multiple fingers (such as typing the letter "u" with their index or middle fingers [62]), I can generate multiple training samples for each combination of the mapping. (Usually, people type one key with no more than two fingers [62], meaning the number of combinations will not be too large.)

5.3.2 BERT-Based Neural Language Model

The neural language model used for decoding finger-tap sequences in TypeAnywhere is based on the BERT model [53]. To fine-tune the model for the task, I used the finger-tap sequence as the input to make the BERT model output the corresponding characters[8]. Specifically, I added a linear classification layer on top of the BERT base model to generate the character for each input code. I decided to decode on the character level to provide real-time feedback for each tap action, even if the user has not yet finished typing the entire word. Another important motivation for decoding on the character level is that such a model can be used to type out-of-vocabulary (OOV) words not in the training data (see Section 5.2.4). Our implementation is based on the open-sourced transformers library [249]. Specifically, I used the *bert-base-uncased* pre-trained model. The model employed the multi-layer bidirectional transformer structure [219] with 12 transformer blocks, 769 hidden layers and 12 self-attention heads. It was pre-trained with a masked language model (i.e., predicting a hidden word with its surrounding context) and next-sentence prediction tasks on 3.3 billion words [53]. BERT was reported to achieve state-of-the-art performance on many language understanding tasks, and became the basis for many subsequent language

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

models [122, 132, 255].

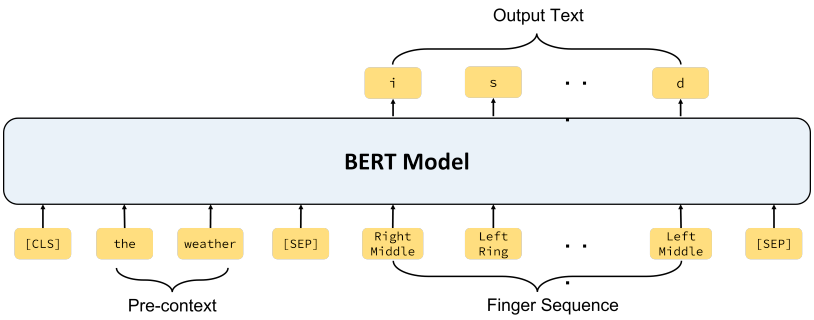

Figure 5.7: Input and output of the BERT decoder model.

The flow of the model input and output is illustrated in Figure 5.7. Given a finger-tap sequence, the model will classify each tap in the sequence as a corresponding character. Because the committed text also provides useful contextual information for the decoder, I also include the text in the *commit area* (we call this *pre-text*) in the typing sequence. For example, if the user taps "*right middle – left ring – thumb – left index – right ring – right ring – left middle*" after committing the text "the weather", then the input to the model will be:

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

Here "`[CLS]`" and "`[SEP]`" are special tokens representing the start of the input and the separator of different parts of the input. The *pre-text* is then tokenized by the model at a sub-word level [251] while the finger IDs are all special tokens, and the decoded text will be "the weather is good". In the TypeAnywhere interface (Figure 5.4), the text in the *commit area* is treated as the pre-text.

5.3.3 Data Collection

I used the Yelp and Amazon review dataset generated by Zhang et al. [277] containing over 3.6 million reviews. For each review in the dataset, I cropped consecutive sequences containing 5 – 25 words, and removed all numbers and special characters so that sequences

only contained the 26 Roman letters and the space character. For each sequence with n words, I then randomly selected m words as the *pre-text* ($0 \leq m < n$), with the rest of the text converted to a finger sequence. For the test dataset, I used the Cornell Movie-Dialogs Corpus [48], which contained conversational exchanges from movie scripts. The purpose was to test the performance in a conversational setting so that the evaluation could reflect real usage scenarios for ubiquitous text entry. I randomly selected 10,000 phrases containing fewer than 25 words from the corpus. In all, I gained 44 million training samples and 10,000 test samples.

5.3.4 Training Process

I fine-tuned the BERT-based model with a linear classification layer (similar to a token classification task in NLP [53]). I calculated the cross-entropy loss of the output characters versus the ground truth text from the finger sequence. I used the AdamW optimizer [133] with a learning rate of 3e-5, and trained the model for one epoch. I applied weight clipping of $[-1.0, +1.0]$, and the batch size was set to 12 due to hardware limitations.

5.3.5 Model Evaluation

The best performance of the model on the test dataset (Cornell Movie-Dialogs Corpus [48]) reached 1.6% character error rate (CER) for an average sentence. To compare its performance with traditional language models, I also trained a 5-gram word-level language model using the KenLM library [90] on the same training dataset, and I excluded the punctuation and special characters other than Roman letters and space. I also applied Kneser-Ney smoothing [160]. The final binary model was 12 GB, which was large compared to the 440 MB BERT model. For evaluation purposes, I pruned the KenLM model by filtering out the words that were not in the test dataset (the movie corpus) to speed computation. The pruned version contains 27,000 1-grams, 0.8 million 2-grams, 2.7 million 3-grams, 4.2 million 4-grams and 4.6 million 5-grams.

To test the 5-gram model, for each word in a test phrase, I first found all possible candidate words based on the word's finger sequence. Candidate words were searched from the Open

American National Corpus (OANC) word frequency dictionary [143], which contained over 22 million words of written American English. I then selected the best word candidate based on probabilities from the language model using the back-off strategy. Both neural and 5-gram decoders were given whole finger-tap sequences, and they both had access to all contextual information. The 5-gram model achieved 5.3% CER. The results showed that the neural language model was both lighter and more accurate than a traditional n-gram model.

5.3.6 Candidate Generation and Auto-Correction with Beam Search

Of course, TypeAnywhere's neural decoder is not 100% accurate, and users may also type an incorrect finger-tap sequence for their intended word. I devised and implemented a few simple algorithms to complete TypeAnyWhere in order to address these challenges. To provide alternative word candidates for a finger-tap sequence, I searched the sequence in the OANC word frequency dictionary [143] (e.g., search the words whose character sequence is the same as the finger sequence), and collected the five most frequent of these words as alternate candidates. I did not use the BERT model for candidates, as it was designed to output character-level results, which did not work well for generating alternate word candidates.

I also needed to deal with situations in which the user typed the word incorrectly, such as missing a tap or interchanging the tap sequence. I developed an auto-correction feature based on beam search for the neural language model. Each time the user performed a tap, I searched through all possible hypotheses for different error types [247]. Besides decoding the detected finger-tap sequence, I also substituted the last finger ID with other finger IDs to detect substitution errors, omitted the last finger ID to detect insertion errors, interchanged the last two finger IDs for transposition errors, and appended one extra finger ID for omission errors. In total, 17 extra sequences were decoded alongside the original sequence.

As an example (Figure 5.8), consider when the user taps the finger sequence "6489". (Recall that finger IDs ranged from '2' to '9' in the model, and I here excluded the thumb IDs of '1' for spacebar in beam search.) The possible combinations then include seven sequence

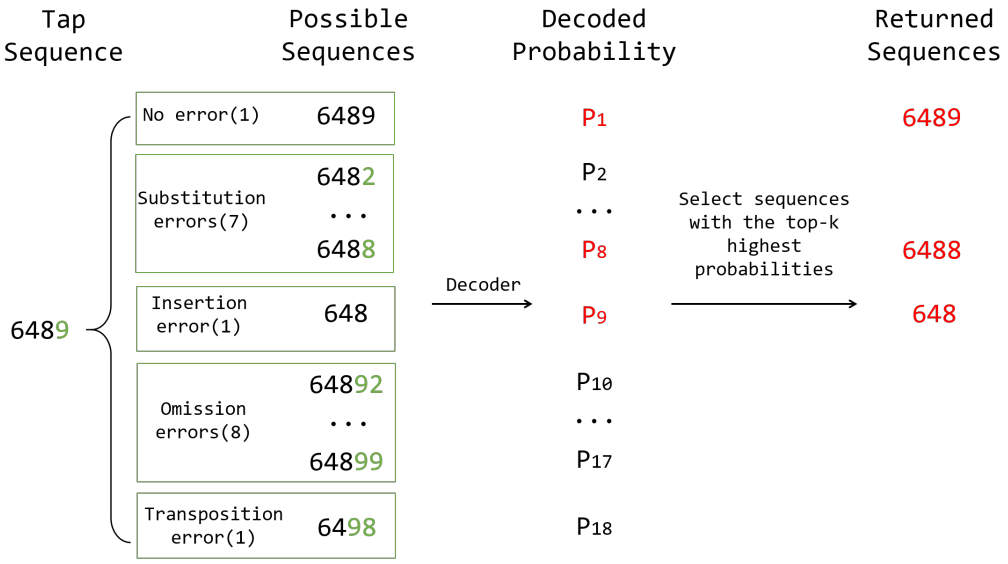

Figure 5.8: Finger-tap sequence beam search example. The original sequence is "6489" with the last finger being '9'. Seventeen extra combinations are then added to the sequence list, and each possible sequence is processed by the decoder. The decoder outputs the probability for each sequence, and beam search then selects the top-K (here, K=3) sequences. If the user taps a new finger, for example, finger '4', the new tap sequences would be "64894", "64884", and "6484". Each of the three sequences then go through the procedure again; this time, there will be $3 \times 18 = 54$ total possible sequences, and beam search will select the top-K sequences from among all candidates.

substitution errors ("648x" where 'x' is '2' to '8'), one sequence for insertion errors ("648"), eight sequences for omission errors ("6489x" where 'x' is '2' to '9'), and one sequence for transposition errors ("6498").

If we maintain all possible sequences each time a new finger-tap is detected, the number of the sequences will explode as tapping continues. I therefore applied beam search to reduce the number of sequences, similar to the surface touch-decoding algorithm [178], where I only kept the top K sequences with the highest probabilities for each step. When a new tap was detected, I generated new sequences for error corrections based on the K sequences. In this

project, K was set to 3, meaning for each tap, TypeAnywhere takes the top-3 sequences thus far. In this way, the model was able to detect most user errors caused by finger mis-taps. The best candidate of the current tap sequence was then displayed in the compose area of the interface in Figure 5.4.

5.4 Evaluation of TypeAnywhere

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

5.4.1 Participants

I recruited 10 participants (aged 23 – 28, all men[9]) for the study via word-of-mouth and online forums. The study was longitudinal over five days, resulting in 50 participant sessions and the ability to observe learning. I required that participants were able to perform desktop QWERTY typing without looking at the keyboard, and that they were consistent with their own personal finger-to-key mappings, i.e., they always used the same finger to type a given letter. Due to the COVID-19 pandemic, it was extremely difficult to recruit participants; I therefore included one participant ("P1") who was not consistent with his finger-to-key mappings, but was able to commit his time to the study. P1 could type on a physical keyboard without looking, but he used multiple fingers to press certain letters, such as "uio", for different words. Therefore, I can regard P1 as a novice learner for TypeAnywhere, as he needed to learn fixed finger-to-key mappings for the study.

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

---

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

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

5.4.2 Apparatus

The typing device used was the Tap Strap mentioned in Section 5.2.1. I built a server written in Python to handle the HTTP communication between the web interface, the hardware, and the decoder. The model ran on a machine with a GTX 1080 graphics card. The testing software was from the TextTest++ open-source text entry evaluation project [272, 274].

5.4.3 Procedure

The study was a five-day longitudinal study. After a participant signed an online consent form, the experimenter sent the hardware apparatus to the participant's home, and instructed the participant on the use of the device and the web interface via Zoom videoconferencing.[10] Specifically, the typing interaction, the text editing gestures, and the correction function were demonstrated. Participants then started to practice typing with the device in the TextTest++ web application. The phrase sets for both practicing and testing were from the MacKenzie and Soukoreff phrase set [140] and the Enron email phrase set [224]. I randomized and divided the phrases so that they were different for practicing and testing in any given session, and also different from one session to the next.

Separate sessions occurred over five consecutive days. In each session, participants would practice typing with TypeAnywhere for 30 minutes on their desk surfaces, and then start testing. During practice, the web interface also displayed a QWERTY layout for the participants to refer to. For testing, 20 different phrases were used each day. Participants were instructed to type as fast and accurately as possible. The "enter" gesture was used to submit a completed phrase. During the evaluation, the participants were told that they could rest before typing each phrase, but needed to finish without stopping a phrase once they began typing it. I logged participants' typing using the transcription sequence model [272], which

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

recorded all the intermediate text during the typing procedure to enable a thorough analysis of the input. After the evaluation, participants sent their log file to the experimenter.

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

5.5 Results

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

The average performance in each condition is shown in Table 5.1. I also report each participant's performance individually to reveal more insights. I refer to the three conditions, respectively, as *tabletop*, *on-lap*, and *keyboard* (i.e., typing on a mechanical desktop keyboard).

Table 5.1: The average words per minute (WPM) and character error rate (CER) and standard deviations (SD) on each day for all participants with TypeAnywhere. Performance with the mechanical desktop QWERTY keyboard and in the typing-on-lap conditions are also shown.

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

5.5.1 Text Entry Speed

Overall performance. Participants' average typing speed is shown in Table 5.1, with the corresponding learning curve shown in Figure 5.9. Their speed on Day 1 was 41.6 WPM, already comparable to mobile phone touch or gesture typing after five days of practice [177]. Their speeds increased each day with this trend still continuing by the end of the study. Compared to the mechanical desktop QWERTY keyboard (87.8 WPM), participants reached 70.6 WPM with TypeAnywhere on the last day. The 19.6% (17.2 WPM) difference between TypeAnyWhere and the desktop keyboard was smaller than the previously reported difference between touchscreen tabletop keyboards [66] and desktop keyboards.

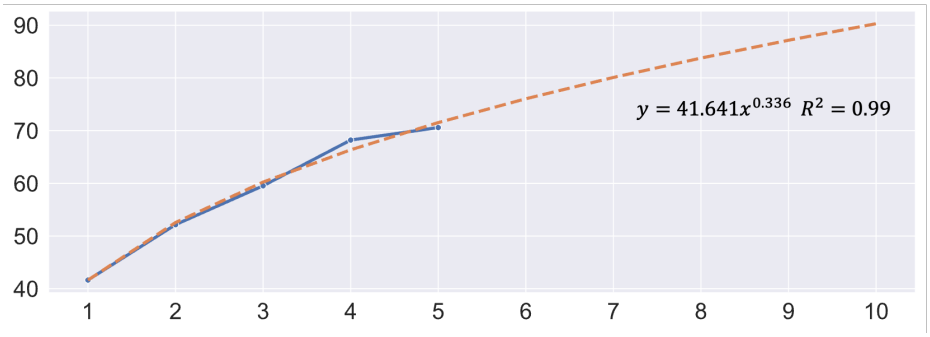

Figure 5.9: The average words per minute and its fitted learning curve for TypeAnywhere being used by participants finger-tapping on a desk. The extrapolated curve is limited to twice the number of sessions [142]

Typing-on-lap performance. The 43.9 WPM speed of the typing-on-lap condition also demonstrated the feasibility of TypeAnyWhere to be used in off-desktop conditions: As long as there is a surface, the user can perform QWERTY-style text entry with reasonable speed. However, as people's laps are softer than table surfaces, TypeAnywhere's finger-tap detection performed less accurately on laps, causing some false-negative non-detections. That said, typing on laps is an odd but potentially useful scenario compared to typing on firm table surfaces, so some degradation of performance is to be expected.

Individuals' performance. Figure 5.10 shows each participant's performance across the

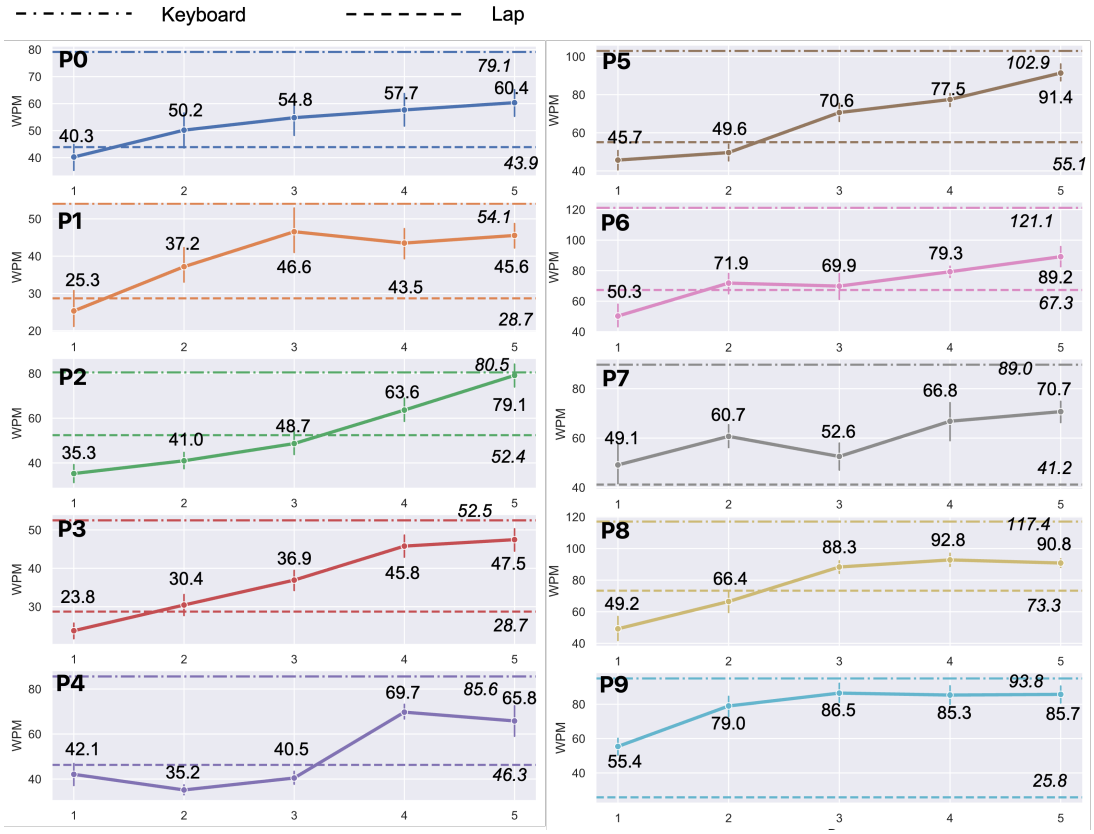

Figure 5.10: Words per minute (WPM) over five days for each participant. Higher is better. The upper horizontal line in each graph represents the desktop keyboard speed; the lower line in each graph represents the typing-on-lap speed. The vertical bar on each data point represents the 95% confidence interval.

five days. All participants increased their typing speeds over time, although there were some small drops due to the vagaries of the test phrases and human performance. With TypeAnywhere, participants reached between 73.7% and 98.3% (mean 80.4%) of their desktop QWERTY keyboard typing speeds on the last day. The fastest typist (P5) reached 91.4 WPM, while P2 even reached a typing speed with TypeAnywhere very close to his desktop keyboard typing speed on the fifth day, about 80 WPM. The novice learner P1 reached 84.2% of his desktop keyboard typing speed, although he had to learn both the interaction and the finger-to-key mappings in the study. The results suggest that people *can* learn Ty-

peAnywhere in a relatively short period of time, even if they do not follow fixed finger-to-key mappings.

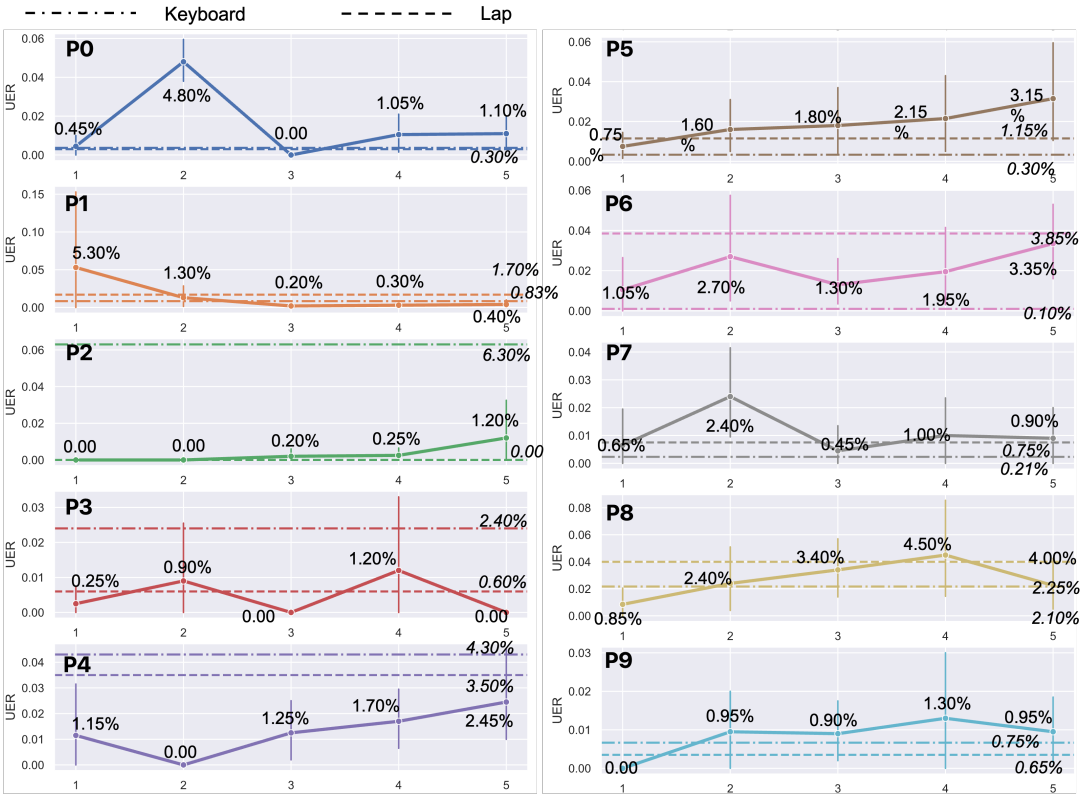

Figure 5.11: Character error rates (CER) over the five day study for each participant. Lower is better. Here I use Uncorrected Error Rate (UER) [201,272] as CER. The upper horizontal lines in each graph represent the desktop keyboard CER; the lower lines represent the typing-on-lap CER. The vertical bars on each data point represent 95% confidence intervals.

5.5.2 Error Rate

Overall performance. Participants' average character error rates (CER) are shown in Table 5.1. These error rates were operationalized as *uncorrected* errors in the final transcribed text, as such errors trade-off against speed [274]; in contrast, *corrected* errors were not counted in CER, since any errors made but fixed during entry take time and are thus

subsumed by speed. There is no obvious trend for the error rates, but the desktop keyboard condition had the biggest CER on average. From the log files, I observed that participants generally made more corrections (i.e., they pressed more backspaces) when typing with TypeAnywhere than in the desktop keyboard condition. However, since they left fewer errors in their final transcriptions, their resultant CER was lower.

Individual performance. Figure 5.11 shows each participant's error rate across the five days. The CER of all participants remained low for all five days. P2, P3, and P4 were even able to achieve lower error rates than the desktop keyboard condition for all sessions, despite the lack of physical keys for TypeAnywhere. Because of auto-correction, participants tended to leave few typos with TypeAnywhere.

Utilization of "Type, Then Correct". I logged the frequency of participants using the Type, Then Correct (TTC) [271] correction feature in the study, finding that only four participants (P0, P2, P4 and P8) used this interaction method for making corrections. Among the 1000 phrases entered, only 16 phrases were completed with this method. As the interaction was a new concept to the participants, it was not surprising that participants still heavily relied on backspace and word-delete functions for text correction. Although the TTC method can be cognitively demanding for new users, with more practice, I expect that participants might make greater use of TTC style corrections, since it is more efficient.

5.5.3 Error Behavior Analysis

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

From the text logs, I located 513 total typing errors made by all participants. I identified those typing errors by finding the mismatches between finger sequences, corresponding text sequences (the transcribed text), and the ground truth text sequences (the presented text).

Among the 513 errors that were typed incorrectly by participants, 224 errors were auto-corrected, with 112 (50%) of those fixed to become the correct text, indicating the usefulness of the auto-correction algorithm.

Table 5.2: Count of different error types each day for all participants.

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

The distribution of errors is shown in Table 5.2. The most common error was typing with the wrong finger (i.e., 299 substitution errors[11]), followed by omitting a finger tap (i.e., 127 omission errors). Although the latter error type was mostly caused by the device not recognizing finger-taps, omissions accounted for less than 50% of wrong-finger errors, which were caused by the user.

The decrease in omission errors also indicated that the participants learned how to make a recognizable tap during practice. Although Tap Strap was reported to have over 98% tap recognition accuracy, not all taps were recognized well during the initial practice session. The Tap Strap company also has a video tutorial on how to perform a proper tap,[12] where it mentions, "*tap gently, and you don't need a lot of force to tap.*" In the study, it took some time for the participants to learn how to tap properly. Specifically, in the first practice session, some participants tended to move their whole hands instead of their fingers to "tap", which caused non-recognitions, as the straps sense the movement of fingers, not the whole hands. Another failure case was when the participants performed "roll-over" actions when typing [62] (*e.g.*, pressing one finger before another finger lifts; this action was common among fast typists). The roll-over finger usually exerted a light touch on the surface instead

[11]The error type definitions can be found in [272].

[12]`https://www.youtube.com/watch?v=XCI7D3IkA6E`

of a tap. Over the five days of learning, participants successfully adjusted their typing style to make sure that taps were recognized by the straps.

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

5.5.4 Subjective Feedback

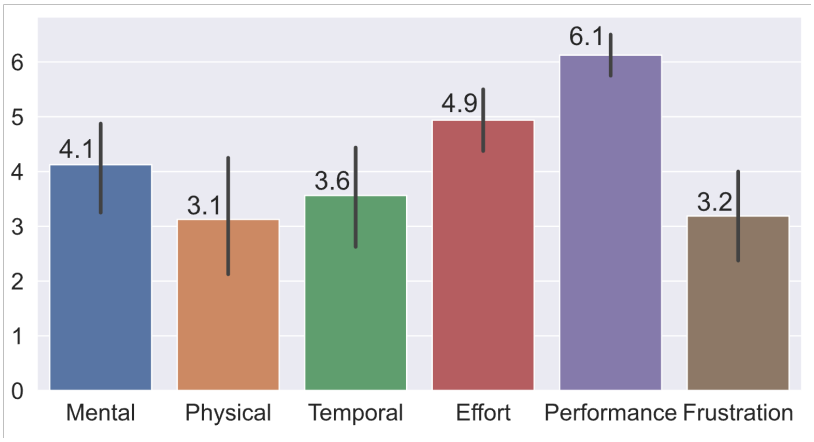

Figure 5.12: Average NASA TLX ratings of the participants (scale 1 - 7). For mental / physical / temporal / effort / frustration, high score means high task load; for performance, high score means participants are highly satisfied with their performance. The vertical bars on each data point represent 95% confidence intervals.

The NASA TLX workload results are shown in Figure 5.12. The mental load and the overall effort were perceived as high, as participants had to explicitly think about the keyboard layout during typing. (By contrast, when typing on a physical desktop QWERTY keyboard, participants can rely upon muscle memory without thinking about which finger is striking each key [46].) The temporal taskload (*i.e.*, time pressure) was also perceived as a bit high, as the participants tried to type as fast and accurately as possible during the study. However, their perceived performance was rated as very successful, indicating that participants were quite satisfied with their typing performance by the end of the study.

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

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

As for aspects of TypeAnywhere that participants wanted to improve, all participants felt they needed to think about their finger-to-key mappings explicitly while typing, which increased their cognitive load. Our novice learner, P1, commented that, *"I used to type without even knowing what my finger-to-key mapping was. It just happened naturally. Now I need to have a keyboard map in my mind during typing."* Participants also complained that the Tap Strap hardware device failed to detect taps sometimes, especially in the typing-on-lap condition. Finally, as the text dynamically changed in the composition area during typing, participants mentioned feeling unsure in the middle of a word, as the intermediate

results were not always as they expected. This uncertainty added to participants' occasional confusion, so sometimes they would just delete a word and retype it.

5.6 Discussion

Combining off-the-shelf wearable finger-tap sensors, the pre-trained deep learning language model BERT, and custom software to enable a novel interaction design, TypeAnywhere has achieved the highest reported performance among manual text input methods other than the desktop physical keyboard 5.1. Although TypeAnywhere does not have the benefit of tactile feedback from mechanical desktop QWERTY keyboards, the design of TypeAnywhere has the benefit of several factors: (1) Its QWERTY-style typing interaction provided for a relatively easy-to-learn system, enabling users to transfer their years of keyboard typing skills to a new system within a short period; (2) Unlike previous work [193,254,259,280], which also required position information, only finger-tap sequences are required in TypeAnywhere, enabling us to utilize an existing large corpus as the training data without the need of collecting real typing data; (3) The neural language decoder took advantage of current advances in natural language processing by incorporating more context during the decoding process and achieving better decoding accuracy compared to a traditional n-gram model, for $n = 5$; (4) To improve the usability of TypeAnywhere, I also implemented auto-correction and alternative candidates based on beam search and Type, then Correct [271] correction interactions, both of which reduce the penalties incurred with making mistakes.

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

adaptation to minimize their necessary movements was actually surprising to the authors, as participants not only transferred their existing typing skills, but also learned a more efficient way to operate their fingers for the interaction.

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

Another noticeable result was the performance degradation in the typing-on-lap condition. A reduction in performance was expected, as participants only performed on-lap typing on the last day, and the surface was softer and smaller compared to the tabletop. Participants also adjusted their sitting postures several times during the test and changed the part of the lap on which they tapped,. P3 observed a "lack of the feeling of taps as the laps and pants are soft." Due to the softness of the lap, it was also harder for the device to detect a "tap" action. Nevertheless, all participants were able to finish the on-the-lap typing and reached an average 43.9 WPM, which is still a considerable entry rate compared to most off-desktop text entry solutions [268]. The on-lap condition was included to demonstrate TypeAnywhere's ability to enable the user to type on most any surface, such as on a wall, a bicycle handlebar, or a user's leg or lap.

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

5.6.1 Limitations

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

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

I only conducted the study with ten participants, albeit longitudinally over five sessions. Although having multiple sessions helps to compensate for the small number of participants, the generalizability could be improved further by a larger study. Also, participants were still improving after five sessions; more sessions would illuminate their performance further. On the other hand, a five-day longitudinal study with five participants produces 50 study sessions, which for most text entry evaluations is a reasonable size. In fact, many text entry studies (e.g., [121, 165, 178, 180]) have smaller participant numbers, and most are not longitudinal.

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

I also did not explicitly measure the accuracy of the tap actions or the editing gestures. As I show in section 5.5.3, the accuracy of the gestures relies not only on how well the device was, but also on how participants performed their taps. And throughout the study, all participants adjusted their typing style for better performance.

Finally, the BERT-based neural decoding implementation was not optimized. Possible optimizations include adding weights to different errors during the beam-search auto-correction procedure; outputting words instead of characters so that the model could also provide al-

ternative candidates; condensing the model size so that it could incorporate larger context and run faster.

5.7 Future Work

Along with addressing the limitations mentioned above, I see several possible directions for future work. I could add comprehensive support for special characters, since currently, only letters and the space character are supported in TypeAnywhere. Numbers, punctuation, symbols, and modifier keys must also be included in a full solution capable of supporting everyday use. One possible solution could be assigning different symbols to different chords and users would perform a mode gesture to switch between typing and symbol-entry modes. Another possible solution could be query-style entry, such as for emojis: the user simply performs a chord to trigger the search mode, and then types the description of the symbol or emoji to search and enter it [270]. For example, typing a "q" in search mode could make a question mark ("?") immediately available for entry.

I also could utilize gestures beyond just finger-tapping. TypeAnywhere's current design only involves tapping, whether by a single finger or chorded for editing operations. Motion gestures, such as swipe, pinch, snap, or open/close hand could be used for various purposes in future iterations. Finally, the devices I used for the project were two finger-worn ring straps. In the future, I expect smartwatches and other wrist-worn devices, or a computer vision-based system, to detect finger taps to avoid the need to wear specialized devices on fingers for text entry. Even with alternate hardware, the finger-tap encoding scheme, neural language decoder, and interaction design all could be reused.

5.8 Conclusion

I have presented *TypeAnywhere*, a system that enables QWERTY-style typing on just about any surface as a general text entry solution for ubiquitous computing. TypeAnywhere applies a neural language decoder that accepts only finger-tap sequences as the input, which eases both the process of training and the generalization to different finger-to-key mappings. I trained the decoder with a large corpus and achieved 1.6% character error rate (CER) in

an offline evaluation. To enable interactive use, I also designed auto-correction and *Type, then Correct* [271] style interaction to lower the risk and cost of making and correcting errors. Our longitudinal study showed that participants were able to learn TypeAnywhere with relative ease, reaching 70.6 WPM on average after five sessions or about 2.5 hours of practice, with the fastest person reaching 91.4 WPM. I hope that TypeAnywhere can serve as a first step towards a usable text entry solution for wearable and ubiquitous computing devices, and will inspire future research to design faster yet intuitive text entry methods for the ubiquitous computing era.

Part II

ACCESSIBLE PICTORIAL INFORMATION INTERACTIONS

This part presents two projects for blind or low vision users to communicate with digital information, in particular, pictorial information. With user-centered methods such as co-design, interviews and online surveys, I built assistive systems for easily interacting with pictorial information, including emojis and animated GIFs.

Chapter 6

VOICEMOJI: EMOJI ENTRY USING VOICE FOR VISUALLY IMPAIRED PEOPLE

Keyboard-based emoji entry can be challenging for people with visual impairments: users have to sequentially navigate emoji lists using screen readers to find their desired emojis, which is a slow and tedious process. In this work, I explore the design and benefits of emoji entry with speech input, a popular text entry method among people with visual impairments. After conducting interviews to understand blind or low vision (BLV) users' current emoji input experiences, I developed *Voicemoji*, which (1) outputs relevant emojis in response to voice commands, and (2) provide context-sensitive emoji suggestions through speech output. I also conducted a multi-stage evaluation study with six BLV participants from the United States and six BLV participants from China, finding that Voicemoji significantly reduced entry time by 91.2% and was preferred by all participants over the Apple iOS keyboard. Based on our findings, I present Voicemoji as a feasible solution for voice-based emoji entry.[1]

6.1 Introduction

Emojis have become an essential element of online communication, with over 3,000 emojis available in the Unicode standard [33]. Facial expressions, emotions, activities, and objects are succinctly represented using emojis. Emojis are widely used in everyday social interactions including text messaging, posting on social media, contacting customer service, and appealing to online audiences through advertisements, making emojis undoubtedly a popular and important way of communicating in today's digital age [172, 172].

[1]This chapter is adapted from: Mingrui "Ray" Zhang, Ruolin Wang, Xuhai Xu, Qisheng Li, Ather Sharif and Jacob O. Wobbrock. (2021). Voicemoji: Emoji entry using voice for visually impaired people. Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '21). Yokohama, Japan (May 8-13, 2021). New York: ACM Press

Table 6.1: Summary of different emoji entry methods. Voicemoji aims to address several problems of current methods by providing features including voice input, fuzzy semantic-level search and emoji suggestions

Method	**Modality**	**Emoji search?**	**Emoji suggestions?**
Emoji keyboard	Touch	No	No
Emoji shortcut	Touch	No	No
Built-in emoji search	Touch	Keyword-level	No
Voicemoji	Voice	Keyword- and semantic-level search	Semantic-level

The prevalence of emojis in online communications means that blind or low vision (BLV) users encounter emojis often. According to a recent study by Tigwell *et al.* [215], 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 [215], current emoji entry methods, including *emoji keyboards*, *emoji shortcuts*, and *built-in emoji search*, are cognitively demanding and unreasonably time-consuming for BLV users. I compare current emoji entry methods and summarize their shortcomings, including *emoji keyboards*, *emoji shortcuts*, and *built-in emoji search*, in Table 6.1 and Figure 6.1.

Such limitations hinder BLV users from using emojis easily, causing social exclusion for BLV users, and reducing their communication efficacy [221]. Through our interviews with BLV users (N=12), we report 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.

Prior work has reported that BLV users employ voice commands more frequently, and are more satisfied with speech recognition, than sighted people [13]. Gboard has support for

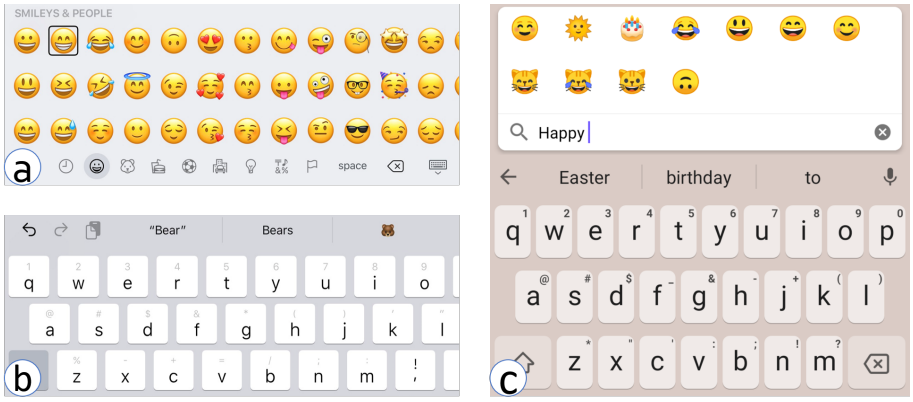

Figure 6.1: Current emoji input methods. (a) The Apple iOS emoji keyboard. Blind or low vision users can use a finger to move over the keyboard to navigate through the options one by one, and the screen reader will read out the name of each emoji. The process is tedious and slow. (b) The emoji shortcut. When the user types certain keywords, such as "bear," the corresponding emojis will appear in the suggestion list. Not all emojis have shortcuts, and the user has to memorize the shortcuts to find them. (c) Built-in emoji search. Some keyboards offer built-in search functions, where the user can input text to search for emojis. The search is only based on manually curated keywords. In our interviews, I found that our participants mainly used the emoji keyboard (a) to input emojis, despite its evident drawbacks.

dictating emojis, where commands like "fire emoji" would input the 🔥 emoji [83]. Apple voice control allows emojis with multi-word descriptions to be inputted by using similar commands [99]. However, both methods requires the user to know the exact name of the emoji.

I present *Voicemoji*, a voice emoji entry system that supports: (1) voice-based semantic-level emoji search, (2) emoji entry with keywords, (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.

Additionally, Voicemoji, at present, supports a rich emoji set accessible through two of the three most spoken languages in the world,[2] 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 code to support the research community and provide a platform for contributions from like-minded researchers and developers.[3]

I conducted a multi-stage study to evaluate Voicemoji with six BLV participants from the United States and six BLV participants from China. After learning the usage of Voicemoji in an initial training session, participants were encouraged to use the Voicemoji system in their daily chat conversations for three days. Then, they participated in a lab study to compare the performance of Voicemoji with their current keyboard-based emoji entry system.

The results show that participants entered emojis significantly faster with Voicemoji than with the Apple iOS keyboard, and the suggestion function of Voicemoji was perceived as relevant and helpful. Qualitative analysis shows evidence that Voicemoji not only improved the emoji entry experience, but also enriched participants' overall online communication experience.

I make three primary contributions in this work:

1. Through semi-structured interviews, I report on the current emoji input experiences and challenges faced by BLV users;

---

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

[3]`https://github.com/DrustZ/VoiceEmoji`

2. I developed *Voicemoji*, a speech-based emoji entry system that enables BLV users to input emojis. I contribute its interaction design, including its commands, functionality, and feedback, which support a multilingual system. Additionally, I provide the source code of our implementation;

3. Through a multi-stage user study, I evaluated the usability of Voicemoji and compared it to current emoji entry methods. The results show that Voicemoji significantly reduces input time for emoji entry by 91.2% and is highly preferred by users.

6.2 Understanding Current Emoji Usage by Blind Users

To design an emoji entry method for blind or low vision (BLV) users, I first need to understand the problems they face when they utilize current emoji systems. To gain this understanding, I conducted multiple semi-structured interviews[4] with the target users from both the United States and China. Specifically, we wanted to answer three questions: (1) How do BLV users currently input emojis? (2) How do BLV users discover and conceive of new emojis? (3) What are the main challenges of using the current emoji entry methods for BLV users?

I recruited 12 participants, 6 from the United States (5 men, 1 woman) aged 18 to 68 (M=35.5, SD=17.1), and 6 from China (4 men, 2 women) aged 25 to 27 (M=26.0, SD=0.9). I contacted the participants by sending emails to BLV community centers. The participants' demographic information is shown in Table 6.2. Seven participants identified as blind and five participants identified as low vision. All participants owned mobile phones and used them daily with screen readers. Due to the inconsistency of emoji descriptions on different platforms, we only recruited Apple iOS users, as the iOS system has more detailed descriptions for each emoji than Android in both English and Chinese. The interviews lasted 45-60 minutes and were audio-recorded for analysis. The interview protocol was guided by the research questions. Participants were compensated with $15 USD or 100 CNY for their time.

[4]Due to the COVID-19 pandemic, I conducted all interviews online.

Table 6.2: Demographic information of participants

ID	Sex	Age	Nation	Visual Impairment	Frequency of Using Emojis
P1	F	25	CN	Blind	Every day
P2	M	25	CN	Blind	Every day
P3	M	26	CN	Low Vision (Retinitis pigmentosa)	Every day
P4	F	26	CN	Low Vision (Retinitis pigmentosa)	Every day
P5	M	27	CN	Low Vision (Retinitis pigmentosa)	Rarely use them
P6	M	27	CN	Low Vision (Peripheral vision loss)	Every day
P7	M	30	US	Blind	Once a week
P8	F	68	US	Low Vision (Central vision loss)	Every day
P9	M	18	US	Blind	Once a week
P10	M	35	US	Blind	Every day
P11	M	35	US	Blind	Rarely use them
P12	M	27	US	Blind	Every week

For analysis, two authors independently coded all of the interview transcripts while discussing and modifying the codebook to reconcile ambiguities on an ongoing basis. The research team discussed any discrepancies until reaching consensus. I did not, however, calculate inter-rater reliability, as the primary goal of the coding process was not to achieve complete agreement, but to eventually yield overarching themes [147]. After coding all interviews, all authors conducted multiple sessions of thematic analysis of the interviews, using affinity diagramming [188] as a modified version of grounded theory [42] to uncover themes of various levels. I present the results in the following subsections. Some participant quotes have been edited slightly and shortened to improve readability without changing their substance.

6.2.1 Current Emoji Entry Practice

I briefly report participants' current emoji entry practices in this section.

Frequency and Motivation

Eight participants reported that they used emojis every day, two used emojis once a week, and two rarely used them (*i.e.*, less than once a week). This result aligns with previous work [215] indicating the popular usage of emojis despite their pictorial and visual nature. However, I found that most participants only used a limited number of emojis frequently (about 10), most of which were emotion-related ones such as smiling faces. When asked about motivations, all people mentioned *enriching the expressiveness* of their communications; one participant (P10) also mentioned using emojis as a quick response. Interestingly, two participants (P8, P10) also mentioned the sense of *belonging and connecting to their peers* when using emojis, which emphasized the social aspects of emoji usage.

Input Methods

For daily communication, six participants used speech input as their main text input method, three used an on-screen keyboard, and three used a braille keyboard. For those who did not use speech input as their main method, they all used speech input for certain situations, such as "quick stuff" (P12) or "when I'm lazy" (P7). All participants reported using the emoji keyboard as the main input method for emojis. For those who used emojis frequently, they would memorize the position of certain emojis. Five participants also utilized the *frequently used page* of the emoji keyboard to speed up their input process. Participants also mentioned using emoji shortcuts, but only P2 used them as the main way to input emojis, memorizing the keywords that brought up certain emoji suggestions. However, other participants felt that this approach was too unpredictable, and they often did not know what keyword could provide a desired emoji suggestion.

Learning New Emojis

To discover new emojis, seven participants mentioned that they got to know new emojis while they were swiping to input a known emoji on the emoji keyboard. Five participants occasionally scrolled through the whole emoji keyboard. Six participants mentioned discovering new emojis from the messages sent to them. Two people also read release notes,

such as Unicode specifications, to learn new emojis. Everyone learned new emojis by their emoji descriptions; however, P10 mentioned that a lot of these descriptions were "confusing and not detailed enough." Participant 8 mentioned that she would connect her phone with a television magnifier to see new emojis, and P12 would search on *Emojipedia.org* to learn new emojis. Five participants also mentioned that they would consult with their sighted friends about how to use certain emojis to feel confident using them. Although all of these learning methods exhibit the tenacity and cleverness of the participants, they amount to labor-intensive workarounds that should be avoidable with better designs.

6.2.2 Challenges of Current Emoji Entry Methods

Through the interviews, I identified several problems with current emoji entry methods for blind or low vision (BLV) users. Many of these problems reduce usability for non-BLV users as well, and unsurprisingly, improving emoji systems for BLV users is likely to improve emoji systems for all users.

C1. Time Consuming

All participants complained about the inconvenience of using the emoji keyboard. There are thousands of options in the Apple iOS emoji keyboard, and these options are even grouped by categories and similarity; but it is still time-consuming to listen to the description of each emoji one-by-one. Participant 7 said, "I actually have to read every single one on the page to know what's on that page. So it just it's time consuming." Participant 2 mentioned that when the procedure was too long she would just give up. The emoji shortcut method was faster compared to the emoji keyboard, but "typing and correcting the text still took time" (P2, P7). Participant 1 and P10 also mentioned that when using the shortcut method, they have to think about the keyword and might try different words to trigger an emoji suggestion.

C2. Inconsistent with Users' Expectations

The main challenge of the emoji shortcut method was the lack of consistency with users' expectations. There was no guarantee that every keyword would result in an emoji suggestion, and the user had to guess the right keywords. Participant 7 said, "When it works, it works really well. When it doesn't work, I have to guess several times and if all the keywords fail, I'm pretty confused." Participant 4 also expressed her confusion: "Sometimes I type exactly the description of the emoji but it does not show the suggestion. Then I feel it is stupid and do not know what to type." The timing of the emoji suggestions was also inconsistent. Some emoji suggestions appeared as auto-correction candidates, while others appeared as auto-prediction candidates. Participant 3 provided an example: typing "happy birthday" would lead to a partying face emoji suggestion in the list, but after the space bar was pressed, the emoji changed to a balloon emoji 🎈. There was also inconsistency in the emoji keyboard method. Many participants mentioned that the categories were ambiguous; for example, the category *Smileys & People* contained cat face emojis.

C3. Lack of Support for Discovering Emojis

There was also no convenient way for BLV users to discover new emojis. Most participants mentioned that they only used a limited number of common emojis. While the emoji list contains many options, only five participants mentioned that they would occasionally navigate among the keyboard to explore new emojis.

C4. Lack of Support for Finding the Right Emoji

Not knowing enough emojis limited the expression participants could convey through emojis. Participant 2 mentioned, "Sometimes I want to add some emojis, but I don't know which to add so I just give up." The keyword suggestion method can mitigate this challenge to some degree, but not always: "If the emoji is suggested by the keyword, I would pick it. However, it might not be the best one in my mind. I pick the suggested ones only because it was too tiring to pick the right one from the emoji keyboard" (P4). Even if users know the emoji exists, they usually ask a sighted friend to explain the context of the emoji. There

is no way for them to discover the proper usage context of new emojis with current input methods.

6.2.3 Features Emerged from the User Interviews

Based on the interview results, I summarize certain key features that *Voicemoji* needs to have to address these challenges.

F1. Support Direct Emoji Entry

To address challenge *C1*, when users have specific emojis in mind, they can directly and easily insert those emojis via speech. Ideally, users can speak both emojis and text in one utterance without explicitly switching modes.

F2. Enable Natural Language Queries

To address challenge *C2*, with Voicemoji, users should be able to ask for emojis in a natural way, rather than having to remember keywords or names. For example, when looking for the emoji *Man with Probing Cane*  , users can simply say, "a blind person emoji" instead of the whole, exact name.

F3. Offer Various Options Related to the Query

To address challenges *C3* and *C4*, Voicemoji should be able to scope relevant emojis if users are unsure which emojis to use. Scoping the results is recommended by the human-AI interaction guidelines [6], where it is suggested offering the user more options from which to choose, and opportunities to discover new options.

F4. Suggest Emojis Related to the Current Context

To address challenge *C3* and *C4*, when users do not know which emojis to use, Voicemoji needs to provide suggestions based on the current message content.

F5. Provide the Ability for Color or Skin Tone Modification

Besides the major challenges, two participants (P7, P10) explicitly mentioned the difficulty of choosing skin tones for certain emoji. With the Apple iOS emoji keyboard, users have to long-press an emoji to trigger the skin tone selector, and go through extra steps to modify the skin tone of an emoji. For a better user experience, users should be able to specify or modify the color of an emoji with speech directly and easily.

6.3 The Design and Implementation of Voicemoji

In this section, I describe the design and implementation of *Voicemoji*. Voicemoji contains several speech commands to trigger emoji search, emoji insertion, and emoji modification. The list of commands is shown in Table 6.3.

Table 6.3: Emoji input commands of Voicemoji and usage examples

Command	Result	Example command	Example result
"Emoji search: *Description* + emoji"	Return a list of emojis relevant to the description	"Emoji search: A blind person emoji"	🧑‍🦯 🦮
"Insert + *description* + emoji"	The most relevant emoji is added directly to the transcription	"Happy birthday insert birthday cake emoji"	"Happy birthday 🎂"
"Change the emoji to + *color/skin*"	The last inputted emoji is changed to the corresponding color/skin	(the input text: "that's great! 👍") "Change the emoji to dark skin"	"that's great! 👍🏿"
Emoji suggestion function	Five emoji suggestions relevant to the spoken content when no emoji command is received	"How about dinner tonight?"	🍴 😋 🍝 😏 🍽️

Voicemoji can be operated using only speech, or in conjunction with VoiceOver or other screen readers. I implemented Voicemoji as a web application for easy cross-device access without the need of app installation. When the user clicks the *speech* button, the screen reader prompts, "Please start speaking," and users can begin their speech input. Users click the button again when they finish speaking. I used the Google Cloud speech-to-text API[5]

[5]https://cloud.google.com/speech-to-text

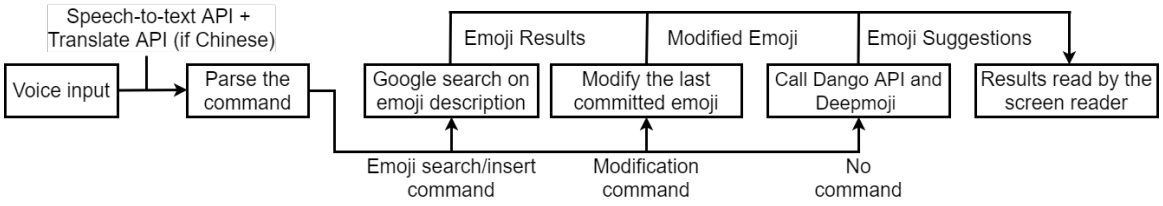

Figure 6.2: The usage flow of Voicemoji. After the transcript of the voice input is received, the server parses the input to check the type of the command it contains. The parsed input is then processed by different subsystems according to its command type. Finally, emoji results are returned and announced.

for speech recognition. To improve the usability of Voicemoji, I added a "copy button" so that users could copy the spoken content and paste it into other messaging apps; we also added a "help" button that announces the basic usage and commands of Voicemoji. To support the remote user study, described in the next section, we added a chat feature in Voicemoji: clicking the "Send" button would send the text to other the users on the website at the same time. The overall process of using Voicemoji is depicted in Figure 6.2. I next introduce each command for searching and inputting emojis.

6.3.1 The Emoji Search Command

I designed the command template, "Emoji search: *description* + emoji" to explicitly search for certain emojis. Voicemoji extracts the description between "Emoji search:" and "emoji" as the query, returning related emojis and announcing them with Apple VoiceOver. For example, the user can say, "Emoji search: a blind person emoji," and Voicemoji will return emojis including 🧑‍🦯 (*Man with Probing Cane*), 🦯 (*Probing Cane*), and 🦮 (*Guide Dog*). Upon receiving the results, Voicemoji triggers the screen reader to announce the names of the emojis one by one. The emoji results are shown as buttons, and the user can either tap an emoji, or select an emoji by its position by saying, for example, "Insert the second one." The usage flow is shown in Figure 6.3.

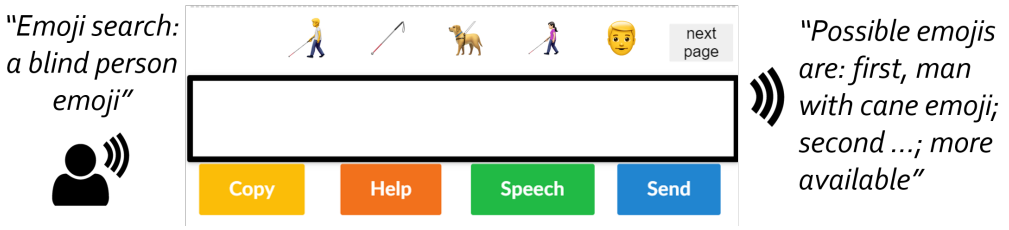

Figure 6.3: Emoji search command flow. When the user speaks the command, "Emoji search: *description* + emoji," Voicemoji will return related emojis above the text field, and the screen reader announces the name of each returned emoji. If there are more emojis available, the screen reader will read the first five emojis, and then say "more emojis available."

As specified in Section 6.2.3, the user's search description does not have to be a predefined emoji keyword or name. Voicemoji accepts any form of natural language as the emoji description (feature *F2*, above), such as "tropical fruit" or "cold weather," even though no specific emojis exist with these exact names.

To enable Voicemoji's search functionality, we utilized the Google search API[6] as the search engine to enable flexible search queries with natural language understanding [158]. After extracting the query, Voicemoji searches the query in *Emojipedia* via the API. Emojipedia is an emoji reference website that documents the names of emoji characters in the Unicode Standard. The Google search API finds the most relevant pages in Emojipedia based on the query. Google's search results may contain different types of websites such as blogs, news, and emoji definition pages.[7] Voicemoji then applies regex matching on the resulting pages to extract emoji definition pages, and adds the corresponding emojis into the list of emoji search results (feature *F3*, above). The results are then announced by the device's screen reader. If there are more than five results, a *next page* button will appear to facilitate page navigation.

---

[6]https://developers.google.com/custom-search/v1/overview

[7]For an example of an emoji definition page, see https://emojipedia.org/fire/ for the "fire" emoji 🔥.

For the Chinese language, we used similar search commands "给我一个 + *description* + 表情" (表情 stands for "emoji"). After extracting the query, Voicemoji translates it into English using Google's Cloud Translation API[8]. The rest of the procedure is the same as for English search queries.

One essential difference between Voicemoji and searching emojis directly on Google is that Voicemoji is a text-input interaction, and it is targeted at improving the communication efficiency. While the user could get the same result by searching on Google, the portion of the interaction for a visually-impaired user (open a browser, go to Google, search the emoji, go to the website, copy the emoji, switch the application, paste the emoji) is significantly higher than using a built-in Voicemoji-like function from the keyboard.

6.3.2 The Direct Insertion Command

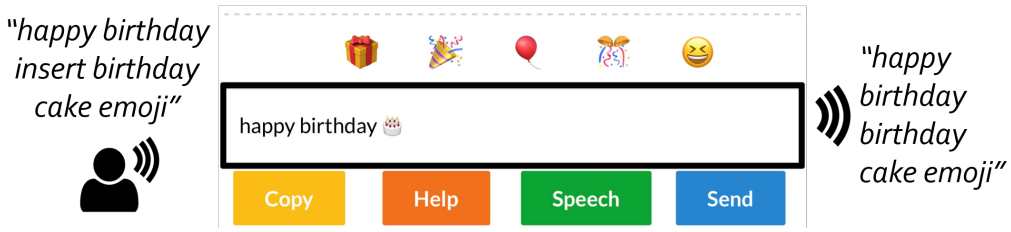

Figure 6.4: Emoji insertion command flow. When the user speaks the command, "Insert + *description* + emoji," or "*single word description* + emoji," Voicemoji will return the transcribed text with the emoji replacement.

I designed two commands to support direct emoji entry within text (feature *F1*, above). The first one is similar to a feature in Gboard: whenever the user speaks a word followed by the keyword "emoji," Voicemoji will replace the word with a corresponding emoji. For example, saying, "walking in the park with my dog emoji" results in "walking in the park with my 🦮." To avoid replacing words that are actually describing the word "emoji," such as, "I like to use emojis in my daily life," we only trigger the replacement of words that are

[8]https://cloud.google.com/translate

nouns or gerunds.

The other command is, "Insert + *description* + emoji," and Voicemoji will replace the whole command with the corresponding emoji. This command enables the direct entry of emojis with multi-word descriptions. For example, "Happy birthday insert birthday cake emoji" results in "Happy birthday 🎂." The usage flow is shown in Figure 6.4. The command for the Chinese language is "插入 + *description* + 表情".

Both the English and Chinese commands replace the description in place, thereby supporting fast emoji entry when the user has a specific emoji in mind. When the processing finishes, the screen reader speaks the transcribed text, including the emoji, to assure the user of the result. The query is processed with the Google search API described above, and the top emoji results are returned.

6.3.3 Emoji Suggestions for Spoken Content

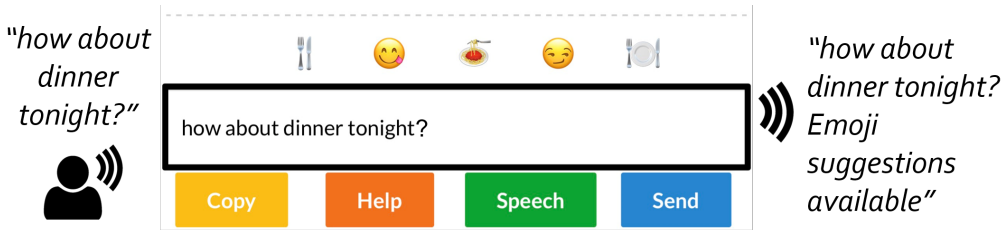

Figure 6.5: When no emoji command is received, five emoji suggestions are produced by Voicemoji based on the spoken word content. For example, for the phrase, "how about dinner tonight?", Voicemoji produces a fork and knife emoji, smiley face licking its lips emoji, plate of spaghetti emoji, smirking face emoji, and dinner plate with utensils emoji.

When the user does not explicitly ask for emojis during speech, Voicemoji suggests relevant emojis based on the current spoken word content (features *F3, F4*, above). For example, if the user says, "How about dinner tonight?", Voicemoji will return the transcription with suggested emojis including 🍴 (*Fork and Knife*), 🍽 (*Fork and Knife with Plate*), 🍝 (*Spaghetti*), 😋 (*Smiling Face Licking Lips*) and 😏 (*Smirking Face*). When emoji sugges-

tions are available, the screen reader says, “Emoji suggestions available” after speaking the transcribed result to remind the user. The suggestions are also shown as buttons, and the user could tap to insert them. The usage flow is shown in Figure 6.5.

This emoji suggestion feature was implemented using two methods: the Dango API [9] and the DeepMoji model [63]. Dango [100] is a mobile application that suggests emojis and stickers based on the message content. DeepMoji is a neural network model trained on 1.2 billion Tweets for emoji prediction. Both methods use neural networks to embed the text into a vector in a semantic space and search for its nearest emoji vectors in the space as the suggestions. For more technical details, the reader is directed to related articles on deep learning and emoji prediction [63, 197]. Voicemoji always returns five suggestions for the current spoken content to improve the emoji variety. After getting the results from the Dango API (usually three or four emojis), Voicemoji then runs the DeepMoji model[10] for further emoji predictions to fill in the remaining slots. I use the Dango API first as it is a commercial product which has a larger training dataset and produces more realistic suggestions than Deepmoji. The suggested emojis reflect both the semantics of a phrase (e.g., suggesting food emojis in the above example) and the affect of a phrase (e.g., suggesting facial expressions). For Chinese input, the content is translated into English and then similarly passed to the Dango API and the Deepmoji model.

6.3.4 Emoji Modification Commands

Voicemoji also supports modification of already inserted emojis (feature *F5*, above). The user can say the command, “Change the emoji to + *description*” to change an already inputted emoji to another one. For example, if the inputted text is, “Take a walk with my 🐕,” speaking “Change the emoji to cat” will modify the dog emoji into a cat emoji 🐈. The command can also modify the color/skin of the emoji if the description contains a color (“yellow,” “blue,” “green,” “brown,” “red,” etc.) or skin tone (“light,” “medium-light,” “medium,” “medium-dark,” “dark,” etc.). For example, to change the emoji *thumbs up* 👍

[9] https://getdango.com/API/

[10] https://github.com/bfelbo/DeepMoji

Figure 6.6: Color/skin modification usage flow. When the user speaks the command, "Change the emoji to + *skin/color modifier phrase*," Voicemoji will change the last inserted emoji to its corresponding color/skin variation.

into 👍🏿, the user can say, "Change the emoji to dark skin." The usage flow for this feature is demonstrated in Figure 6.6.

To implement this color/skin modification feature, I extract the description and decide whether the description contains a color or skin modifier word. If not, Voicemoji just searches the query using the Google search API as usual. If a color or skin modifier word *is* detected, Voicemoji forms a new query by combining the last inserted emoji and the description (for example, 👍 + "dark skin"), and feeds this new query to the Google search API. The first emoji result is then used to replace the inserted emoji in the text.

Voicemoji also supports other modification commands: removing an inserted emoji ("Remove the emoji" or "Delete the emoji"), inserting an emoji in the suggestion list ("Insert the first one"), and making the screen reader read the names of emoji options currently in the list ("Read emojis"). All such modification commands make changes to the last inserted emoji. In this way, Voicemoji can be operated entirely via speech if desired.

6.3.5 Voicemoji Implementation Details

Voicemoji is an web application written in JavaScript. To support its accessibility features, I used aira-labels [148] on the UI elements in order to make them recognizable by a screen reader. The speech output for announcing emoji search results was implemented with

the aria-live feature [148]. To facilitate the remote user study, described below, I also used WebSockets to support a real-time messaging function. The Voicemoji backend was implemented with the Tornado library [216]. Finally, the Google Speech, Google Translate, Dango, and DeepMoji services were all incorporated into Voicemoji as described above.

6.4 Voicemoji Evaluation

I conducted a user study to evaluate Voicemoji and understand its usage. Specifically, I focused on three questions: (1) What is the performance of Voicemoji for emoji entry compared to today's *de facto* entry methods? (2) Is Voicemoji's suggestion feature useful? What is the perceived relevance of the suggested emojis? (3) How does Voicemoji impact the online communication experience of its users? I describe the study to answer these and other questions below.

6.4.1 Participants

I invited the same 12 blind or low vision (BLV) participants shown in Table 6.2 from the formative study to this summative evaluation. All of the participants used a mobile phone running the Apple iOS system, and were familiar with the built-in screen reader VoiceOver. The study lasted for about 2 hours, and participants received $30 USD or 200 CNY for their time.

6.4.2 Apparatus

To optimize for network speed, the testing website was deployed on a university server for the United States participants, and on a commercial Virtual Private Server (VPS) in South Korea for the Chinese participants. I did not deploy the server in China due to the Great Firewall, which blocked the Google services on which Voicemoji relied.

6.4.3 Procedure

All study sessions were conducted remotely via Zoom or WeChat, and were audio recorded for further analysis. I asked the participants to turn up their VoiceOver volume so that the

experimenter could hear the output over their video link. The speech rate of Voice Over was set to 70% for Chinese participants and 65% for American participants, values differing slightly owing to the different output rates of the two languages. The study was conducted on two separate days for each participant. The first phase in the study was a tutorial during which I introduced Voicemoji, and instructed participants how to use different commands for emoji search and selection. Participants tried each available command and demonstrated their ability to use Voicemoji. I then scheduled the summative study three days after the tutorial, and encouraged participants to use Voicemoji for their daily communication needs during the intervening period.

The second phase of the study was a formal summative evaluation, which contained two sessions: the *emoji entry session* and the *emoji suggestion session*, described below.

Emoji Entry Session. I asked the participants to input 27 phrases containing emojis using each emoji entry method (Voicemoji and the current Apple iOS keyboard); the order was counter-balanced. Both methods used speech input for text. (In the iOS keyboard condition, participants were told to input the text with dictation.) When using the iOS keyboard, participants were told they could use their preferred emoji entry method for each phrase. For example, they could use the emoji shortcut method for some phrases and the emoji keyboard for others. The rationale was to regard the iOS keyboard condition as the *de facto* means of inputting emojis, which makes available a combination of different methods, just as in everyday life. Furthermore, participants could always fall back to the emoji keyboard if they could not find the emoji using the shortcuts.

To make the task reflective of real conversational situations, I designed four groups of phrases:

- Five phrases with emojis at the end of the text: "*Are you going to join us for lunch?* 🍔 *(hamburger)*"
- Five phrases with emojis replacing a word in the text: "*A* 🎁 *(present) isn't necessary.* "
- Five phrases with emojis at the beginning of the text: "👀 *(eyes) See you soon!* "

- Ten phrases with only emojis: "😂"

I also asked participants to *compose* an original phrase with text plus emojis and a phrase with only emojis, which added up to 27 test phrases in all. Phrase text was randomly selected from the Enron mobile phrase set [224], and the first author added relevant emojis to each phrase. All other authors then reviewed the phrase set and agreed on the relevance of each emoji. For emoji-only phrases, I selected the five most popular emojis at Emoji Stats[11]. The website shows the real-time emoji usage data from Apple iOS keyboards *emojiXpress*[12]. I randomly added five more emojis into the phrase set. The phrase set is provided in Appendix B.1. I translated the phrases into Chinese for the Chinese participants. Three of the authors translated the phrases and verified the correctness together.

The experimenter sent every phrase to each participant via Voicemoji's chat function. Participants then heard the content of each phrase on their devices, and began to input the phrases using either the Apple iOS keyboard or Voicemoji. They then pressed the *Send* button to send the finished phrase back to the experimenter for verification. After finishing all 27 phrases with one method, participants took a five minute break and then started to input the phrases with the other method.

Emoji Suggestion Session. I sent eight phrases without emojis to the participants, asking them to speak the phrases using Voicemoji and to consider the relevance of the emoji suggestions. Specifically, participants were asked to select the emojis that they would use with the spoken phrase from among the five suggestions produced by Voicemoji. The purpose was to evaluate the accuracy and the usefulness of the emoji suggestion feature. Eight phrases were selected from the *Sentiment 140* dataset [77], with four positive sentiment phrases and four negative sentiment phrases. The Sentiment 140 dataset contains 1.6 million Tweets annotated with positive or negative sentiment labels. The phrase set is provided in Appendix B.2.

Participants were also instructed to compose a phrase by themselves in order to get a sense

[11] http://www.emojistats.org/

[12] https://www.emojixpress.com/

of using the system in real-settings; therefore, every person evaluated nine phrases in the emoji suggestion session.

After both sessions were complete, participants were asked to rate different methods with the SUS questionnaire [29] and the NASA TLX survey [88], followed by a short debrief on their experiences using the Voicemoji method.

6.4.4 Metrics

For the emoji entry session, there was one independent variable, *entry method*, with two levels: Voicemoji and the *de facto* Apple iOS keyboard. I measured the entry time from the audio recording, including the *total entry time* and the *emoji entry time*. Because participants were using Apple's built-in VoiceOver screen reader, the audio feedback of all actions could be recorded, allowing me to measure timings from the audio record.

Total entry time started when participants pressed the *Speech* button in the Voicemoji condition, or when they started dictation in the Apple iOS keyboard condition[13]. The timing ended after participants finished each phrase and VoiceOver read it aloud.

Emoji entry time was part of the *total entry time*, and measured only the time for emoji entry. For Voicemoji, it included the time for speaking the commands, playing the results, and choosing the emojis from the results list; for the Apple iOS keyboard, emoji entry time included the time for switching from the alphabetic to the emoji keyboard, and visually searching for and selecting emojis in the list–or, if participants used an emoji shortcut, it included the time for typing the shortcuts and selecting them from the suggestion bar.

Due to network latency and processing time, I removed the waiting time before the results were read by Voiceover in the Voicemoji condition. I did this because I wanted to evaluate the interaction time, rather than the vagaries of Voicemoji's current implementation. Processing time was neither equal nor controlled for the two methods: Voicemoji required the network for API usage and hosting the web page, while the Apple iOS keyboard's functions

[13]All participants used Apple's so-called "Magic Tap" gesture to start dictation, which was a double tap with two fingers.

operated offline. Including the processing time would thus confound the interaction time: the average processing time was 2.2 ± 1.3 seconds for English, and 3.3 ± 1.7 seconds for Chinese; the average network latency was 3.9 ± 1.1 seconds for U.S. participants, and 5.2 ± 3.2 seconds for Chinese participants. The first author went through all of the audio recordings and manually calculated the entry time. These results were then validated by another author, whose results were in agreement.

For the emoji suggestion session, I recorded the number of emojis that were perceived as relevant suggestions. I then calculated the accuracy based on the number of chosen suggestions: *pick-1 accuracy* examined whether any of the five suggested emojis were perceived as relevant. As long as any emoji from the suggestions were picked, the *pick-1 accuracy* was 100% for the suggestion; *overall accuracy* examined how many emojis from the suggestions were perceived relevant on average. For instance, if three of five suggestions were chosen as relevant, then the *overall accuracy* was 60%.

In addition, I also logged the usage of the Voicemoji website during the three-day interval between phase one and phase two of the study. For privacy, I only logged the emoji results and the IP address of the query. The IP address was for differentiating users.

I also analyzed participants' qualitative feedback on Voicemoji. I analyzed the audio transcripts using an open coding method [42]. Two of the authors first translated the Chinese transcripts into English, coded the transcripts individually, and met to achieve consensus on the codes. The codes were then discussed by the research team using affinity diagramming [188]. The codes were created in a random order and iteratively arranged into a hierarchy of themes.

6.5 Quantitative Results

In this section, I present the results from our user study. Overall, I collected 648 phrases (27 phrases × 2 methods × 6 participants from each nation × 2 nations) for the *emoji entry task*, and 108 phrases (9 phrases × 6 participants from each nation × 2 nations) for the *emoji suggestion task*.

6.5.1 Emoji Input with the Apple iOS Keyboard

Before diving into comparative results between the Apple iOS keyboard and Voicemoji, it is useful to characterize emoji input with the Apple iOS keyboard. With this keyboard, participants entered emojis with either the emoji keyboard or the emoji shortcut method, described above. Three American participants only used the emoji keyboard method, three used both methods, and all six Chinese participants used both methods to enter emojis. On average, for the three American participants who used both methods, 13.3 of 27 phrases (49.3%) were completed with the emoji keyboard, and 13.7 of 27 phrases (50.7%) were completed with emoji shortcuts. Chinese participants completed 12.7 of 27 phrases (47.0%) with the emoji keyboard, and 14.3 of 27 phrases (53.0%) with the emoji shortcut.

There were also occasions when participants first tried the shortcut method, but failed to retrieve their desired emoji, and then switched to the emoji keyboard. On average, there were 4.3 of 27 phrases (15.9%) that participants first tried the shortcut method and then switched to the emoji keyboard in the U.S. group, and 5.3 of 27 phrases (19.6%) in the Chinese group.

6.5.2 Entry Time

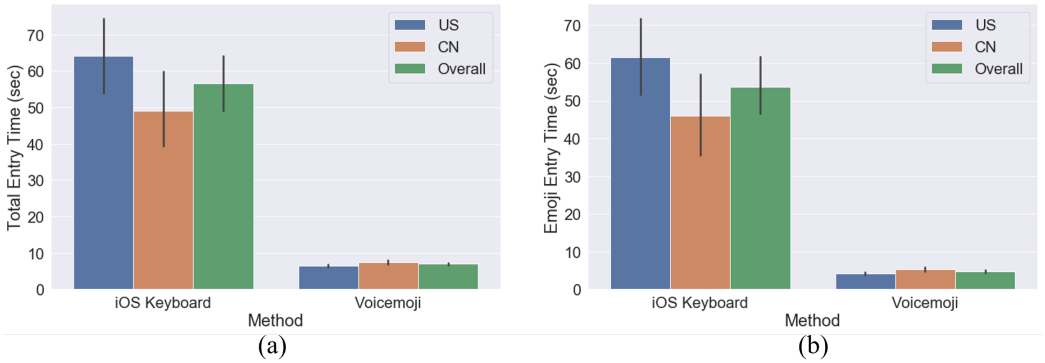

Figure 6.7: The average (a) *total entry time* and (b) *emoji entry time* of the two methods by nation (they were log-transformed in the analysis). Error bars represent 95% confidence intervals (CIs).

Figure 6.7 shows the *total entry time* and *emoji entry time* of the two methods. The average total entry time for Voicemoji was 6.9 seconds, which was 87.1% shorter than the 56.6 seconds for the Apple iOS keyboard. I log-transformed *total entry time* and *emoji entry time* to make both fit gamma distributions, as is common practice with time measures [129]. I performed analyses of variance using a generalized linear mixed model (GLMM) with gamma link function [146] on *total entry time* and *emoji entry time* separately, treating *entry method* and *nation* as fixed effects, and *participant* and *trial* as random effects. For *total entry time*, I found a significant main effect of *entry method* ($\chi^2_{(1,N=648)} = 486.29, p < .001$), indicating that Voicemoji was significantly faster than the Apple iOS keyboard. The effect of *nation* was not statistically significant ($\chi^2_{(1,N=648)} = 0.09, n.s.$). There was a significant interaction between *nation* and *entry method* ($\chi^2_{(1,N=648)} = 9.22, p < .01$), as shown in Figure 6.8(a).

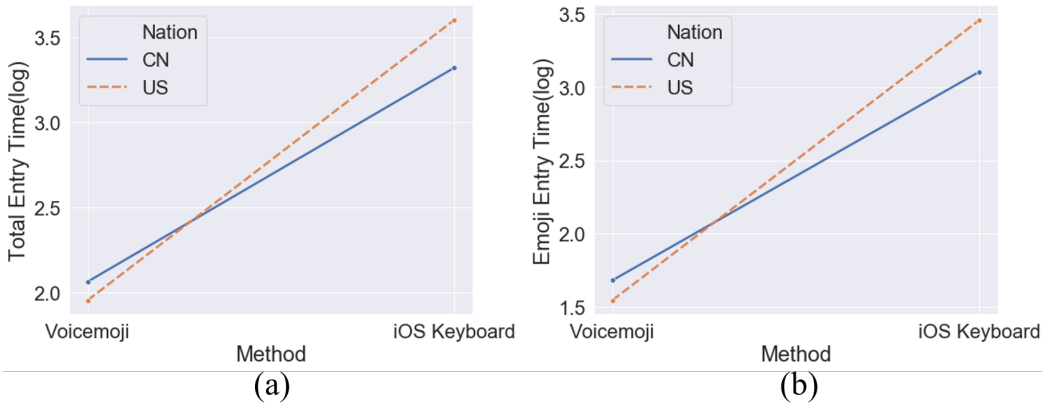

Figure 6.8: Interaction effect of *Method* × *Nation* on log entry time. Both interaction effects indicate that the time saved by Voicemoji in the U.S. group is more than in the Chinese group.

The average *emoji entry time* for Voicemoji was 4.7 seconds, which was 91.2% shorter than the 53.7 seconds for the Apple iOS keyboard. I found a significant main effect of *entry method*
($\chi^2_{(1,N=648)} = 513.55, p < .001$). There was no statistically significant effect of *nation* ($\chi^2_{(1,N=648)} = 0.06, n.s.$). There was also a significant interaction between *nation* and *entry*

method ($\chi^2_{(1,N=648)} = 16.11, p < .001$), as shown in Figure 6.8(b).

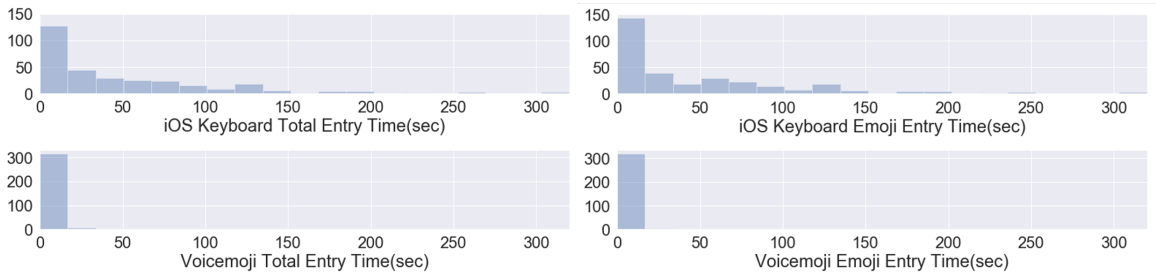

Figure 6.9: Frequency distribution of the *total entry time* and *emoji entry time* in seconds for the two methods. All the tasks were finished within 20 seconds for Voicemoji, while the distribution were heavy tailed for the Apple iOS keyboard.

From the time distribution (Figure 6.9), I noticed that the users also entered certain emojis quickly with the iOS keyboard. The could be explained by several observations in the study: 1) participants used certain emojis more frequently (*e.g.*, ❤️) than others (*e.g.*, 😻), so that they were familiar with the keywords of those emojis; 2) when using the emoji keyboard, some participants with residual vision entered the emojis faster as they could utilize the visual information to guide the search procedure.

Taken together, then, our results for *time* make it clear that Voicemoji was significantly faster than the Apple iOS keyboard for entering text with emojis.

6.5.3 Voicemoji Suggestion Accuracy

The *pick-1 accuracy* for Voicemoji was 100% for each phrase, as all participants picked at least one emoji suggestion for every phrase. The *overall accuracy* was 76.0% for the Chinese group (participants picked 3.80 ± 1.03 emojis as relevant suggestions for each phrase), and 72.2% for the American group (participants picked 3.61± 1.24 emojis as relevant suggestions for each phrase). These results indicate that the emojis suggested by Voicemoji are generally perceived as relevant by users from both countries.

6.5.4 Voicemoji Usage

I logged usage data on the Voicemoji website during the three-day interval before the second phase of the study to see whether participants started to use Voicemoji in their daily lives. There were 6 participants (2 from China, 4 from the U.S.) who used the website, resulting in 84 emoji-related commands. On average, participants used the website 4.7 times a day to input emojis. Given that I had told participants that using the Voicemoji tool was optional during this three-day interval, it is hard to draw firm conclusions from this limited usage. However, it does seem that some participants voluntarily began to use Voicemoji in their daily lives even apart from our formal study.

6.5.5 Subjective Ratings

Our subjective ratings data were ratings on the SUS usability instrument [29] and NASA TLX workload instrument [88], which enabled me to capture usability and workload ratings for both the Apple iOS keyboard and the Voicemoji system. For the Apple iOS keyboard, I separated the emoji keyboard method and the emoji shortcut method when capturing subjective ratings because these methods offered different interaction designs and user experiences. If the participant used both methods in the Apple iOS keyboard condition, then they rated both methods. I therefore had three American participants who only rated the emoji keyboard and Voicemoji, while the other nine participants rated all three methods (i.e., the emoji keyboard, emoji shorcuts, and Voicemoji). Both SUS and TLX results are graphed in Figure 6.10. I calculated the SUS score on a scale of 0-100, where higher indicates "more usable".[14] The average SUS score was 90.4 (SD=8.11) for Voicemoji, 46.5 (SD=16.8) for the emoji keyboard, and 60.8 (SD=21.5) for emoji shortcuts. Because of network latency issues, some participants mentioned lowering their SUS scores for Voicemoji.

I performed the nonparametric Aligned Rank Transform procedure [244] on the SUS scores and NASA TLX ratings to examine any effects of *Method*. I found that for SUS scores, *Method* had a statisitcally significant effect ($F_{2,32} = 27.70, p < .001$). Pairwise comparisons with the Bonferroni correction showed that the Voicemoji received higher scores than the

[14] https://measuringu.com/sus/

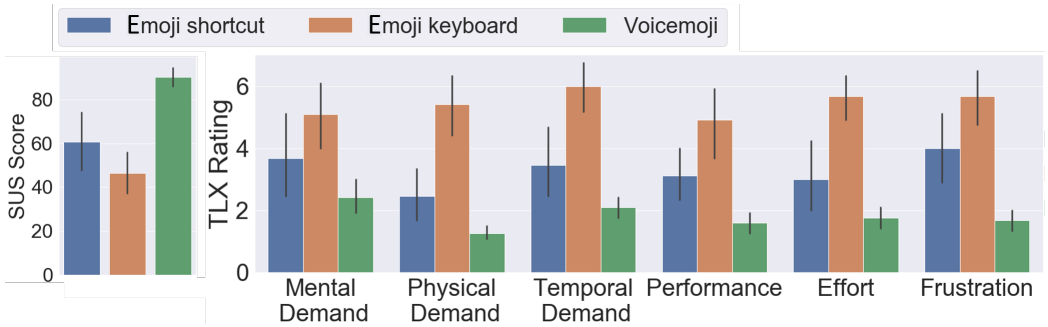

Figure 6.10: SUS usability scores and NASA TLX workload ratings of different emoji input methods. For SUS scores, higher indicates "more usable" and is better. For NASA TLX ratings, lower indicates "less workload" or "better performance" and is better.

emoji keyboard ($p < .01$) and emoji shortcuts ($p < .05$), while there was no significant difference between the emoji keyboard and emoji shortcuts. For NASA TLX ratings, I found that *Method* had a statistically significant effect on all dimensions: *mental demand* ($F_{2,32} = 8.35, p < .005$), *physical demand* ($F_{2,32} = 87.80, p < .001$), *temporal demand* ($F_{2,32} = 26.95, p < .001$), *performance* ($F_{2,32} = 16.64, p < .001$), *effort* ($F_{2,32} = 30.49, p < .001$) and *frustration* ($F_{2,32} = 27.13, p < .001$). Pairwise comparisons with the Bonferroni correction showed that Voicemoji had significantly lower *physical demand* and *frustration* than emoji shortcuts ($p < .05$); and had significantly lower (better) scores on all ratings than the emoji keyboard ($p < .05$).

6.6 Qualitative Results

Our affinity diagramming analysis revealed four main themes arising in participants' feedback about Voicemoji: *conveniently entering emojis*, *helpful suggestions*, *supporting emoji exploration*, and *enriching communication*. I describe each of these in turn.

Conveniently entering emojis

All participants mentioned that finding and entering emojis with Voicemoji was much easier than with the Apple iOS keyboard. As most participants were already familiar with speech

input generally, they quickly mastered Voicemoji after learning the commands. Participants consistently mentioned two benefits of Voicemoji: (1) *supporting both fuzzy and precise search*, and (2) *the ease of speech input*. Four participants appreciated the flexibility with which Voicemoji queries could be formulated, namely that they could either speak the name of the emoji, or describe emoji when unsure of their names. As P6 said, "It is good that Voicemoji can understand my description. I can use [the] 'emoji search' command if I'm not confident, and use the 'insert' command when I know the exact name, which is very efficient." Three participants mentioned feeling it was great that they could input emojis with voice. As P4 said, "[Voicemoji is] very convenient when I want to add some emojis when I'm speaking a long paragraph, I can just do it at once without switching the keyboard."

Helpful suggestions

Four participants mentioned that Voicemoji's suggestion feature was useful in helping them find the right emojis. As P12 said, "I think it was really helpful for just even quickly composing messages that have the right emoji in them, and there's really nothing like Voicemoji's suggestion function." Participant 2 also said, "some suggestions were even better than my intended emojis." Participants 1 and 6 mentioned that even when they had no intention of adding an emoji, the emoji suggestions provided them with appropriate emojis, making it easy to just tap one to insert it.

Supporting emoji exploration

Four participants mentioned that Voicemoji provided a new way to learn new emojis. The multiple options provided by Voicemoji sometimes exposed emojis that were previously unfamiliar to users. As P1 said, "It will suggest some emojis I've never imagined, such as an emoji about a particular object." Participant 2 agreed, saying that as a result, the unexpected emojis "add surprise and delight of using emojis, and offer a new way to learn them." Participant 2 also mentioned that finding rare emojis was a painful process in the Apple iOS keyboard, while Voicemoji's entry process was equally efficient for all emojis.

Enriching communication

Participants 6 and 8 explicitly talked about Voicemoji enriching their daily communications. Participant 6 did not use emojis much before the study, but he found that using Voicemoji lowered the effort of finding emojis, and he started to send emojis during his everyday messaging, which "improved the expressiveness and the interactivity." Participant 8 reported using emojis frequently before our study, and she found that using Voicemoji made the process even easier and faster, making her "feel as mainstream as my sighted friends."

6.7 Discussion

In this work, I have presented *Voicemoji*, a speech-based emoji input method. In creating Voicemoji, I sought to address four major challenges of existing emoji entry methods, namely that they are *time consuming*, *inconsistent with users' expectations*, *lacking support for discovering emojis*, and *lacking support for finding the right emojis*. To address these issues, I designed several features including speech-based search and automatic emoji suggestions. The user study showed that finding and entering emojis with Voicemoji was significantly faster than the current Apple iOS keyboard, and most of the suggested emojis were perceived as relevant to users' spoken content. Importantly, Voicemoji's apparent benefits were consistent for both the American and Chinese participants, suggesting that the design of Voicemoji is at least somewhat generalizable across different languages.

Many participants expressed that they would use the Voicemoji even after the study. Participant 10 commented that he was doubtful about the usability at first, but after the study, he commented that Voicemoji was, "practical as hell" and he "would immediately use the tool" in his everyday life. The feedback given by P6 and P8 about Voicemoji "enriching communication" indicated that the impact of Voicemoji was not only on the speed or effort of entering emojis, but also on a subjectively higher level, where Voicemoji provided support for people to express themselves better. Emojis improve expressiveness, and technologies like Voicemoji that facilitate their entry can improve this expressiveness.

One piece of feedback I did not anticipate was that participants mentioned the potential usefulness of Voicemoji beyond the blind community. Both P3 and P8 mentioned that

the design of speaking emojis could actually "benefit beyond blind people, as many of us use voice input sometimes" (P3). Adopting a perspective of ability-based design [245, 246], technologies designed for users with disabilities are often more usable to wide range of users [156, 232]. For example, when people are walking or driving, they might benefit from using speech more because of so-called "situational impairments" [243], and Voicemoji could facilitate emoji entry in such situations.

6.7.1 Limitations

As with any research project, there are several limitations of this work.

First, I only conducted studies with existing Apple iOS users because of the consistency of the system. Different Android phones have different voice-over descriptions for the same emojis, and not all emojis have descriptions on Android systems compared to iOS. However, some Android keyboards such as *Gboard* provide a built-in emoji search function, which might result in better performance than the Apple iOS keyboard.

Second, the study sessions were all conducted remotely, making them somewhat less controlled than a typical lab-based study. For example, participants used their own Apple iPhone models, which could be somewhat different from one another. Network latency was also not controlled during the study, where a lab-based study on the university campus could achieve shorter response times from the server. However, as the purpose of the comparison is to evaluate the timing of the two interactions (*i.e.* the search interaction, the command design, the voice feedback), rather than the current implementation performance, I did not include the latency in the analysis.

Third, the Google speech-to-text API I used would cut off the speech after a pause longer than two seconds, which meant that participants could not pause too long during their speech. Also, Voicemoji did not support real-time recognition due to network issues, which hindered the user experience.

Fourth, although I logged website usage during the three-day interval between study phases, conducting a long-term field deployment would reveal insights on how participants use

Voicemoji in their daily communications, if at all. The preliminary evidence suggests that some participants would indeed use Voicemoji in their daily lives, but this result is anecdotal at this point.

6.8 Future Work

Beyond remedying or addressing the limitations described above, there are exciting directions for future work based on this initial study. Six participants mentioned that it was cumbersome to switch between apps to use Voicemoji, as it was a web app, and so a priority would be to build Voicemoji into an actual keyboard, which is supported on Android devices. I already have open-sourced the implementation of Voicemoji, and keyboard developers could consider adding it to their keyboard projects. I also see value in adding explanations and example uses emojis. For example, P6 expressed that sometimes he could not understand an emoji just by its name, and many emojis have similar names. Therefore, it would be helpful if Voicemoji could also provide explanations of how to use emojis. Finally, I also think the style of interaction employed by Voicemoji could be extended to other forms of visual media besides emojis, such as stickers and memes [75].

6.9 Conclusion

In this work, I have presented *Voicemoji*, a speech-based emoji input method to make emoji input, search, and discovery easier and more accessible for blind or low vision (BLV) users. I conducted an interview study to understand the current emoji entry experience and challenges for BLV users, and designed multiple novel Voicemoji features, including direct and fuzzy emoji search, speech-only emoji insertion, color or skin tone emoji modification, and automatic emoji suggestions to address the challenges. Results from the formal evaluation with both American and Chinese participants demonstrated that Voicemoji provided significantly faster entry times, greater perceived usability, and lower perceived workload for both groups of participants than the current Apple iOS keyboard. I hope that by open-sourcing the implementation, Voicemoji will encourage keyboard developers to consider designing inclusive input methods for emojis. I also hope that this work will inspire future research in non-textual information entry methods.

Chapter 7

GA11Y: AN AUTOMATED GIF ANNOTATION SYSTEM FOR VISUALLY IMPAIRED USERS

Animated GIF images have become prevalent in internet culture, often used to express richer and more nuanced meanings than static images. But animated GIFs often lack adequate alternative text descriptions, and it is challenging to generate such descriptions automatically, resulting in inaccessible GIFs for blind or low-vision (BLV) users. To improve the accessibility of animated GIFs for BLV users, I provide a system called *Ga11y* (pronounced "galley"), for creating GIF annotations. Ga11y combines the power of machine intelligence and crowdsourcing and has three components: an Android client for submitting annotation requests, a backend server and database, and a web interface where volunteers can respond to annotation requests. I evaluated three human annotation interfaces and employ the one that yielded the best annotation quality. I also conducted a multi-stage evaluation with 12 BLV participants from the United States and China, receiving positive feedback. I anticipate Ga11y to be a practical solution for efforts to improve GIF accessibility.[1]

7.1 Introduction

Animated *Graphics Interchange Format* images, or "GIFs," are looped animations comprising a sequence of images, and are popular forms of content on the Web, messaging, and social media. Many GIFs are made from clips of videos (e.g., movies, TV shows, etc.) or animated cartoons (such as "stickers"), and because of their dynamic nature, GIFs can be used to express richer and more nuanced meanings than static images or text. People use GIFs on social media platforms (such as Twitter), in online messaging apps (such as Face-

[1]This chapter is adapted from: Mingrui "Ray" Zhang, Mingyuan Zhong, Jacob O. Wobbrock. (2022) Ga11y: an Automated GIF Annotation System for Visually Impaired Users. Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '22). New Orleans, USA (April 30 - May 6, 2022). New York: ACM Press.

book Messenger), and to make "memes." As of July 1, 2021, GIFs were used on 21.3% of all websites [228].

The prevalence of GIFs on the internet means that blind or low vision (BLV) users encounter them often. As I report below, in the study with 12 BLV users, all had encountered GIFs on social media platforms or received them in online massaging apps. However, most GIFs have no text annotations that would make them accessible to screen readers. For example, only 0.04% of GIFs on Twitter were annotated in February 2020 [74], and those that *were* annotated typically had short unhelpful descriptions of just one or two words.[2] Unlike static images, GIFs often contain a sequence of images with holistic meaning, which is challenging for current computer vision technologies to recognize and describe [128]. The fact that GIFs are inaccessible to screen readers creates a barrier for BLV users to fully participate in internet culture, causing social exclusion and a reduced richness of online experience.

Further exacerbating the challenge of making animated GIFs accessible to BLV users is that even when a GIF's visual content can be accurately recognized using machine intelligence, having only a GIF's visual information is usually insufficient for understanding it. Unlike emojis, whose descriptions are created and standardized by the Unicode Consortium[3], GIFs are mostly created by individual users, and their meanings can largely depend on context, such as the background of a movie character, the sarcasm of a meme, or the emotion contained in a facial expression. Therefore, it is necessary to have a person who understands this context to supply the meaning of an animated GIF.

To address these challenges, I present *Ga11y* (pronounced "galley"), a GIF annotation system that combines machine intelligence and crowdsourcing to supply annotations for animated GIFs on the internet (see Figure 7.1). Ga11y contains three components: (1) an Android client in which users can trigger annotation requests, (2) a server for GIF matching and annotation storage, and (3) a web interface for human annotation. The Android client runs an accessibility service on the phone, which detects animated GIF

[2] `https://blog.twitter.com/en_us/a/2016/introducing-gif-search-on-twitter`

[3] `https://unicode.org/consortium/consort.html`

and "sticker" elements on the screen. When the focus of the screen reader is on such an element, the user can press a "request for annotation" button to record the GIF and send it to the server. The server compares the similarity of the requested GIF with existing GIFs in the database. If the GIF is not in the database or has not yet been manually annotated, the server generates an automated description using computer vision; otherwise, a human-annotated description is returned. In the meantime, Ga11y's web interface enables human annotation for all requested GIFs, and once an annotation is manually updated by a volunteer, it is supplied to the server database for future retrieval. In this way, users get timely annotations even when GIFs are new to the server, and over time, the number of human-annotated GIFs increases. To enable others to contribute to and extend Ga11y, I open-source its entire implementation as part of this work.[4]

To increase the chances of receiving useful human annotations from Ga11y's web interface, I explored three annotation interface styles for volunteers annotating GIFs. These interface styles were: (1)*freeform*, where the volunteer was only asked to "provide a description" without any guidance; (2) *semi-structured*, where the volunteer was asked to "provide a description of the GIF," with specific guidance on important aspects of a GIF to mention; and (3) *structured*, where the volunteer was asked to answer a set of structured questions regarding a GIF's content. I collected and evaluated these GIF annotations from the three styles using the Amazon Mechanical Turk platform, and gathered feedback from BLV users ($N = 11$). The results showed that both sighted and BLV users preferred the *semi-structured* style for providing GIF annotations. I therefore implemented this style as the web annotation interface for Ga11y.

I then conducted a multi-stage user study with 12 BLV participants to evaluate Ga11y. Specifically, I were interested in how the annotation system affected participants' online communication experiences on social media and messaging platforms. I first conducted a one-hour remote usability test, where participants used Ga11y to request annotations of five GIFs that were already manually annotated on the server, and five new GIFs not in the server. Then, after the usability test, participants were encouraged to use Ga11y

[4] `https://github.com/DrustZ/Ga11y`

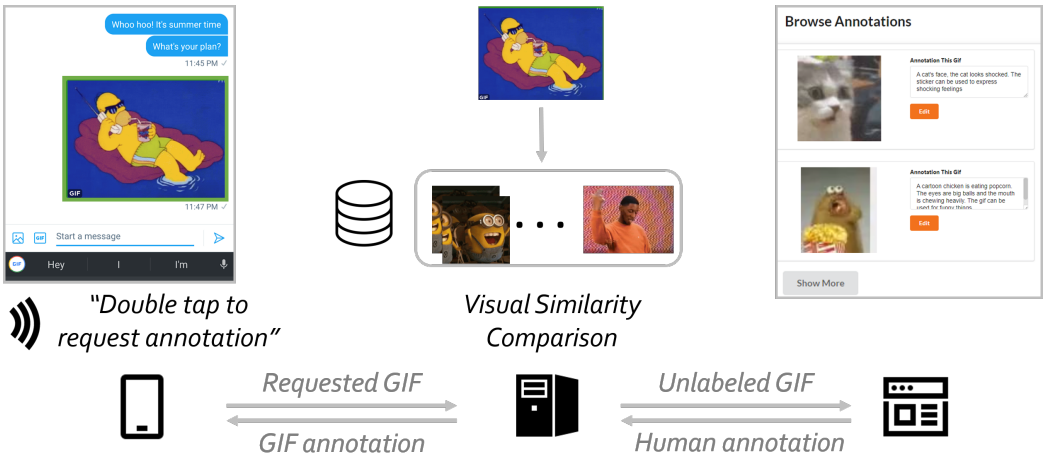

Figure 7.1: The components of Ga11y. The user requests the GIF annotation on the mobile client via the screen reader, and the requested GIF is searched for in the human annotation database on the server. If there is a visually similar GIF with a human annotation, that annotation will be returned; otherwise, a machine-generated annotation is returned, and the unlabeled GIF is then displayed in the Web-based human annotation interface. Once the GIF is annotated by volunteers on the website, the annotation is updated in the server's database for future retrieval.

for two days whenever they encountered GIFs or "stickers" on their phones and provide feedback. The results showed that users perceived Ga11y as a helpful tool for their online communication. They also rated Ga11y as having "high usability," with an average System Usability Scale (SUS) [29] score of 89.1 out of 100. Each participant also used the system 13.5 times per day on average "in the wild."

I make three primary contributions in this work:

1. I explored three interface styles for providing human annotation for GIFs, evaluated them with both sighted and BLV users, and identified the best style for generating high-quality GIF annotations;

2. I developed *Ga11y*, an end-to-end system providing annotations for GIFs and "stickers" on mobile devices. The contributions within the system include the interaction design for requesting a GIF description, animated GIF re-construction and similarity matching algorithms based on computer vision, and the successful combination of machine and human intelligence. Additionally, I provide the source code of Ga11y's implementation;

3. Through a multi-stage user study, I evaluated the usability and performance of Ga11y. The results show that Ga11y received positive feedback from users, is highly usable, and the human annotations were perceived as the most helpful feature within the system.

7.2 User Interaction in Ga11y

In this section, I describe user interaction scenarios when using Ga11y to request GIF annotations from a smartphone device, and provide human annotations for GIFs on Ga11y's web interface. I then present how each part of the system is designed and implemented.

7.2.1 Requesting a GIF annotation

Ga11y runs as a background service on an Android device. Blind or low vision (BLV) people can use TalkBack[5] to navigate through screen elements on an Android phone, and they can move the focus of TalkBack by swiping left or right. (They also can read screen elements by keeping a finger persistently on the screen.) When a BLV user moves the TalkBack focus onto a GIF element (Fig. 7.2a), the Ga11y service identifies that the focused element is a GIF based on properties supplied by system *AccessibilityEvents* and app-specific heuristics (details explained in section 7.4.1). If the element is recognized as a GIF, then the user will hear "double tap to request annotation" on their next swipe (Fig. 7.2b).[6] The user can either continue swiping to ignore the action, or double-tap to request the annotation of the GIF, which triggers the Ga11y service to record the GIF for five seconds[7] and send a request to the server. After the GIF is recorded, the client will make beep sounds until the annotation is successfully fetched to indicate the requesting status, and the annotation will then be read aloud; the user is free to move the focus during the fetching process. If the client fails to fetch the annotation due to connection issues, a double beep will be made to indicate the failure.

On the server side, the requested GIF will be compared to existing GIFs in the annotation database based on their visual similarity. If there is a similar GIF in the database, its human annotation will be sent back to the user's smartphone and read out by the screen reader; if not, the request will be added as an unlabelled GIF in the database and, in the meantime, a computer-generated annotation derived using computer vision will be sent to the user (Fig. 7.2c). In this way, the user always receives an annotation of the GIF quickly, although the machine-generated annotation will generally be less informative than the human-generated ones. Over time, human annotations will accrue as will the number of annotated GIFs,

---

[5]TalkBack is a screen reader service included in the Android operating system.

[6]I designed an extra swipe for annotation request as a double tap on the original GIF content usually has its own functionality (such as zoom in or open a GIF page view) which might conflict with the annotation function.

[7]An average GIF is about 3 seconds [128], I added 2 more seconds for redundancy. If the user moves the focus during the recording, the request is cancelled by default.

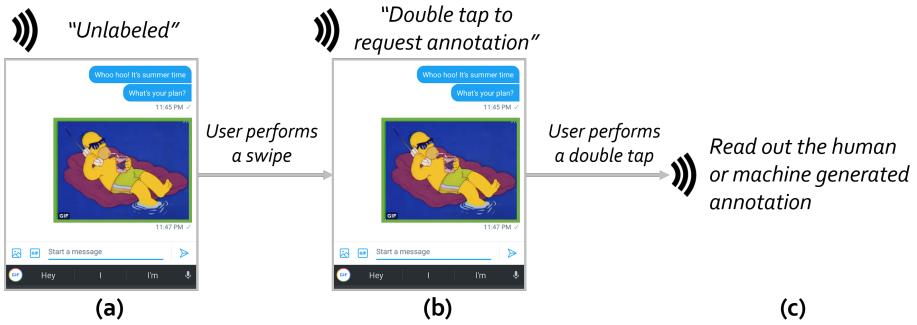

Figure 7.2: Requesting a GIF annotation on a smartphone: **(a)** an unlabeled GIF is focused by the screen reader; **(b)** on the next swipe that moves the screen reader focus, the user can double-tap to request the annotation; **(c)** after the request is processed by the server, the annotation will be returned and read out loud. A human annotation is returned if the requested GIF is already annotated in the database; otherwise, a machine-generated annotation is returned.

giving users the best annotations possible.

7.2.2 Annotating a GIF

The web interface of Ga11y is available to the public, allowing volunteers to annotate GIFs through the website. The website contains two pages as shown in Figure 7.3. One page displays all of the GIFs whose annotations are requested from users' smartphones but which are not yet annotated; website users can click a GIF to add an annotation. The other page displays GIFs that are already annotated by volunteers, and website users can click "edit" to revise and potentially improve the annotation.[8] Once an annotation is updated, the data is synced in the database.

[8]For now, I do not have quality control mechanism to validate that the revised version is better than the original, and leave it as a future work.

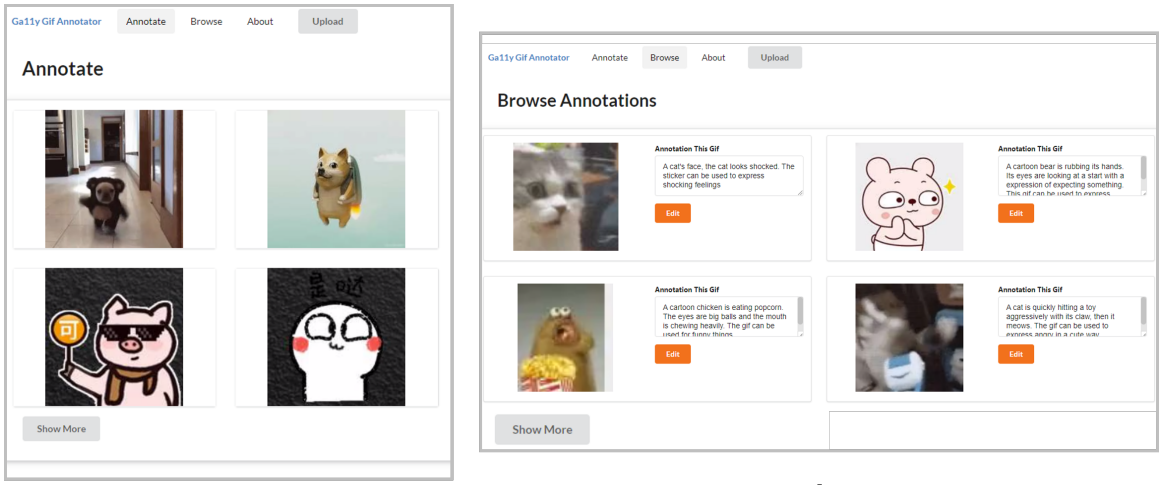

Figure 7.3: The annotation web interface: **(a)** the unlabelled GIF browsing page, and **(b)** the annotation browsing page.

7.3 Designing the GIF Annotation Task

As anyone can contribute annotations through the Ga11y web interface, I need to ensure that our task design leads to high quality annotations. For one thing, crowdsourcing tasks always face challenges of quality control, as the crowd is composed of people with diverse backgrounds and abilities [49]; for another, BLV users might have different needs than sighted users when trying to understand a GIF's content (e.g., sighted users might place more emphasis on visual attributes of the content). In order to design an annotation task that is easy for annotators to understand and that yields useful information for BLV users, I conducted an evaluation with three different annotation interface styles. I then ran a study on Amazon Mechanical Turk to obtain annotations from the three interface styles, and evaluated the results based on both sighted and BLV users' feedback.

7.3.1 Annotation Task Styles

Based on previous research on improving image annotation quality [49, 55, 115, 135, 153] and understanding BLV users' needs for visual content descriptions [74, 102, 154, 194, 208], I designed three GIF annotation interface styles: *freeform*, *semi-structured* and *structured*.

Freeform. In the *freeform* style, there is a GIF to be annotated and an instruction to "*Annotate the GIF in English. How would you describe this GIF and its context to someone, so that they can understand its content even without seeing the GIF?*" The only requirement is "*use a minimum of 8 words.*" The volunteer is then free to write the annotation.

Semi-structured. In the *semi-structured* style, beyond only providing the same instruction as the *freeform* style, I also include several tips for guidance. The tips are generated from previous research on describing GIFs and images generally [74, 128, 144, 208]:

- Please describe the visual content and the related information that is helpful for understanding the GIF;
- Please describe any actors (people, animals, etc.), their actions and expressions, the activities underway, and the environment in which those activities are taking place;
- If this GIF contains clips or actors from movies, television, or similar sources, please describe that information;
- If there is text in the GIF, please describe it;
- Please keep your description concise overall;
- Please use a minimum of 8 words.

Structured. In the *structured* style, I divided the tips from the *semi-structured* style into multiple questions. Instead of writing the annotation in one text field, the annotator provides answers to each question in a form (and they could write *N/A* for "no answer"). The answers were then concatenated into one complete annotation. The separate text fields are prompted with:

- What are the main actors (people, animals, etc.) in this GIF? (If none, write N/A.)

- What are the main actors (people, animals, etc.), if any, doing?
- What are the main actors (people, animals, etc.), if any, expressing (e.g., their emotions or expressions)?
- What is happening in the GIF (e.g., activities, events, actions)?
- If this GIF contains clips or actors from movies, television, or similar sources, provide their names, the source (if you know it), and any other relevant information.
- Is there text in this GIF? If so, what is the text?
- Is there any other information you would like to provide for describing this GIF? For example, if it is a meme, you can describe it.

This set of prompts was inspired by previous work [153] in which template-based tasks were easier for annotators and included more information compared to unstructured annotations.

In all three of the interface styles, I offered an example GIF and its annotation, as previous research suggested that providing high-quality examples could improve annotation quality [115, 194]. I also asked the annotator to "*describe when and how people might use the annotated GIF with an example*" to inform BLV users about the context of use.

7.3.2 Collecting Annotations

To evaluate the three interface styles, I collected annotations generated from each style on Amazon Mechanical Turk. I chose 34 GIFs that exhibited a range of features, including whether cartoon or live action, memes, clips from movies, TV programs, or illustrations, whether characters were present, whether a storyline was conveyed, and whether text was present. Example GIFs are shown in Fig. 7.4.

I crowdsourced three annotations for each interface style for each GIF, meaning each of 34 GIFs had nine total annotations, for $34 \times 9 = 306$ annotations in all. To recruit a diverse participant pool, I limited the number of annotations one Turker could provide to two. GIF orders were randomized to avoid learning. I paid participants $1 USD for each annotation they provided.

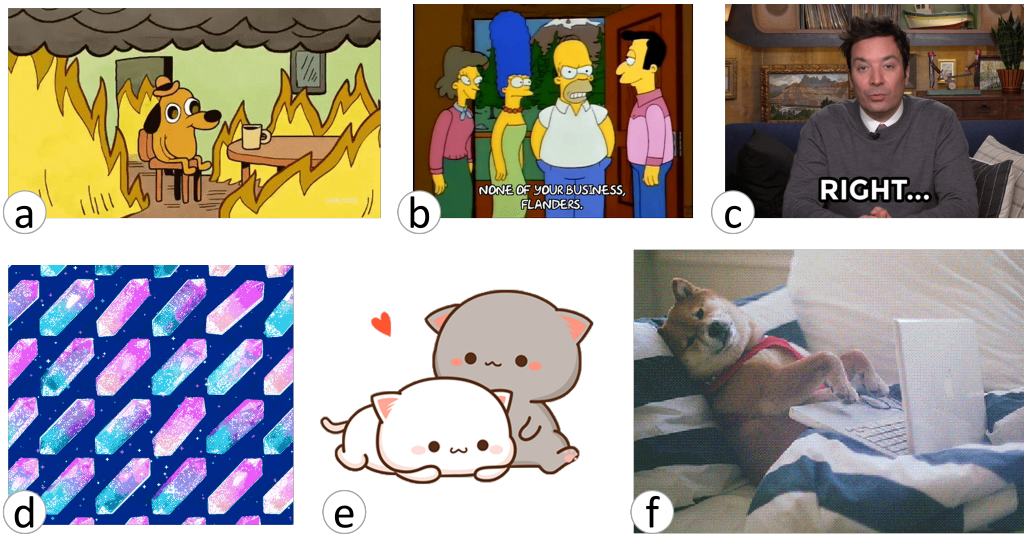

Figure 7.4: Example GIFs collected for the annotation tasks. **(a)** *This is fine* meme, including a cartoon dog drinking coffee in a room on fire, saying, "this is fine" (words not shown); **(b)** a clip from the cartoon *The Simpsons*; **(c)** a clip from *The Tonight Show* with Jimmy Fallon that contains the text "*right...*"; **(d)** patterened illustration of blue and purple diamonds; **(e)** a "sticker" with two cartoon cats; and **(f)** a video clip showing a dog typing rapidly on a laptop computer.

During data collection, I removed answers that were obviously unrelated to the annotation task (e.g., pasting unrelated text from other sources), and finally collected 306 annotations from 246 Turkers. Example annotations are provided in the appendixC.1. The descriptive results for each task design are shown in Table 7.1. I performed a one-way ANOVA on the *annotation length* and the *task completion time*, both of which were log-transformed to comply with the assumption of conditional normality [129]. I found that *Interface Style* had a significant effect on both annotation length ($F(2, 304) = 8.04, p < .001$) and task completion time ($F(2, 304) = 53.5, p < .001$). *Post hoc* pairwise comparisons, corrected with Holm's sequential Bonferroni procedure [94], indicated that the *semi-structured* style yielded significantly more words than both the *freeform* style ($t(34) = 3.79, p < .001$) and the *structured* ($t(34) = 3.03, p < .005$) style, with no significant difference between the

Table 7.1: Means (and standard deviations) for annotation length and completion time (for one annotation) for the three interface styles.

Task Design	**Annotation Length (words)**	**Task Completion Time (seconds)**
Freeform	56.8 (25.7)	1358.9 (885.3)
Semi-structured	60.1 (24.6)	998.9 (853.7)
Structured	57.2 (25.1)	1280.7 (921.5)

freeform and *structured* styles ($t(34) = 0.73$, *n.s.*). As for task completion time, the *semi-structured* style took significantly less time than both the *freeform* ($t(34) = 10.13, p < .001$) and the *structured* ($t(34) = 6.83, p < .001$) styles, and the *structured* task took significantly less time than the *freeform* style ($t(34) = 3.24, p < .005$). On the whole then, it seemed the *semi-structured* interface style produced the most content in the least amount of time.

7.3.3 Evaluating Annotations with Sighted Users

I evaluated the collected GIF annotations by having both sighted users and BLV users rate the annotations and provide feedback. Our evaluation with sighted users was conducted on the Amazon Mechanical Turk platform. The task description was, "*Rate each description of the same GIF, as I wanted to use one of them to make people understand the content without seeing the GIF.*" For each GIF, I displayed one annotation from each interface style in a single page, and let the user rate each annotation in four respects:

- Informative (*does the annotation contain enough information and detail?*)
- Clear (*is the language style clear?*)
- Accurate (*does the annotation accurately describe the GIF content?*)
- Understandable (*is the annotation easy to understand?*)

For each GIF, there were $3 \times 3 \times 3 = 27$ annotation combinations, and I collected 9 ratings

for each annotation. Each Turker was allowed to only rate at most five tasks to increase the number of distinct participants.

To evaluate the annotations for one GIF, participants first read its three annotations. Then, the associated GIF appeared after one minute. I delayed the unveiling of GIFs in this way to ensure that GIFs' visual depictions would not influence participants' judgments of their text annotations. Participants then provided their ratings for each annotation on a scale from 1 to 10, with "1" being "extremely negative" to "10" being "extremely positive." Participants could also write down their reasons for their ratings in an optional text box. The user interface for rating annotations is shown in Fig. 7.5.

Overall, I collected $9 \times 9 \times 34 = 2754$ ratings for each annotation, the results for which are presented in Table 7.2. I performed analyses of variance based on mixed ordinal logistic regression [1, 91], treating each GIF id as a random factor to account for repeated measures. I found that *Interface Style* (freeform, semi-structured, structured) had a significant effect on how informative annotations were ($\chi^2(2, N = 2754) = 17.08, p < .001$), how clear annotations were ($\chi^2(2, N = 2754) = 30.95, p < .001$), how accurate annotations were ($\chi^2(2, N = 2754) = 17.34, p < .001$), and how understandable annotations were ($\chi^2(2, N = 2754) = 34.76, p < .001$).

Table 7.2: Means and standard deviations for ratings from sighted users of GIF annotations. The scale is 1-10, with "10" being the most positive.

	Freeform	**Semi-structured**	**Structured**
Informative	7.1 (2.4)	7.5 (2.3)	6.8 (2.5)
Clear	7.2 (2.3)	7.6 (2.2)	6.6 (2.5)
Accurate	7.4 (2.3)	7.7 (2.2)	7.1 (2.4)
Understandable	7.4 (2.4)	7.8 (2.2)	6.8 (2.6)

Post hoc pairwise comparisons corrected with Holm's sequential Bonferroni procedure [94]

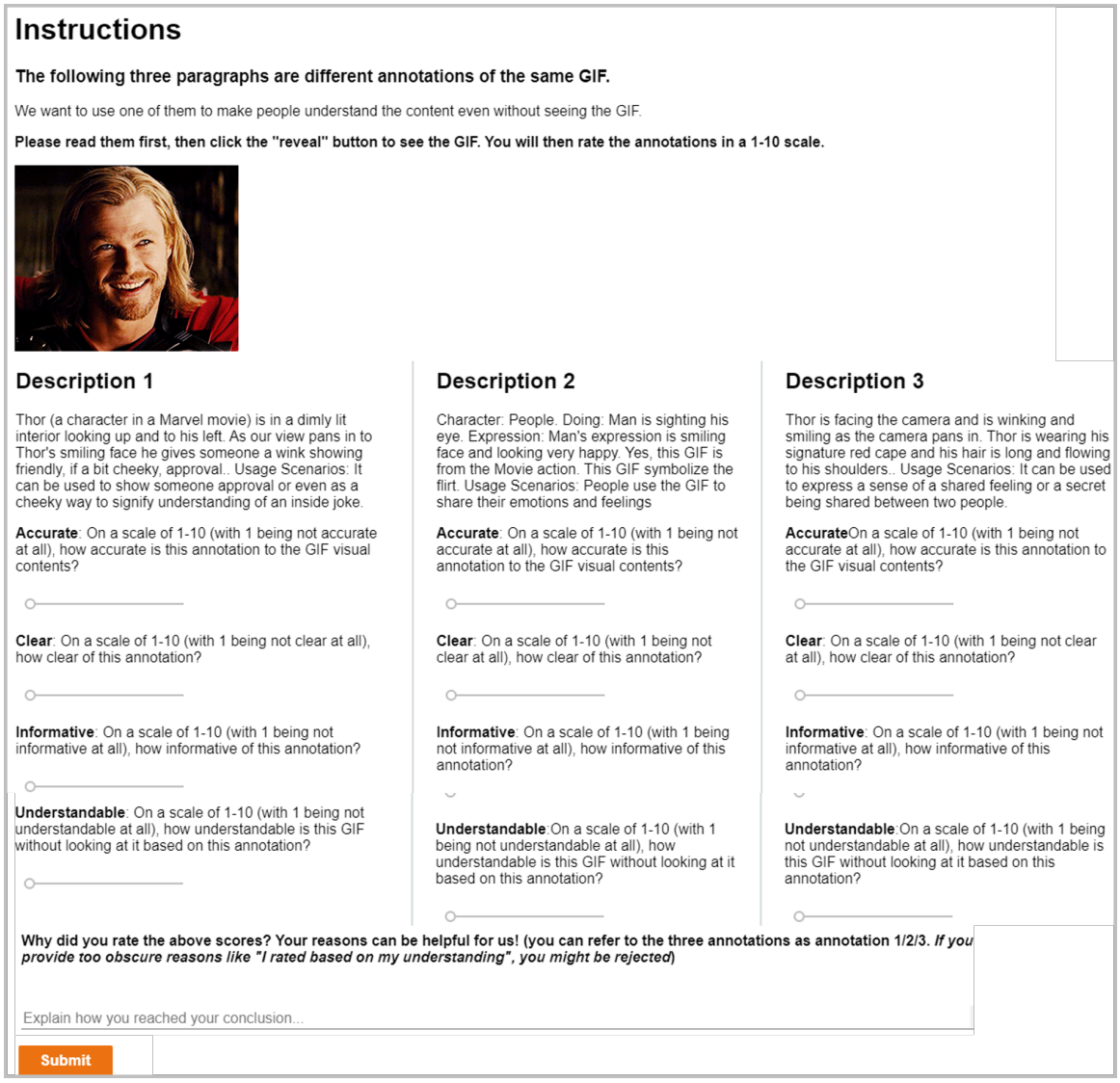

Figure 7.5: The annotation rating interface on Amazon Mechanical Turk. The GIF is revealed after one minute to encourage the rater to read the three descriptions first without being influenced by the GIF's appearance. Annotations from the three annotation interface styles are shown left-to-right as *semi-structured*, *structured*, and *freeform*.

indicated that for the informative rating, semi-structured annotations were significantly more informative than both *structured* ones ($Z = 4.15, p < .001$) and *freeform* ones ($Z = 2.54, p < .05$). For the clarity rating, *semi-structured* annotations were significantly clearer than both *structured* ones ($Z = 5.69, p < .001$) and *freeform* ones ($Z = 2.64, p < .05$), and *freeform* ones were significantly clearer than *structured* ones ($Z = 3.08, p < .01$). For the accuracy rating, *semi-structured* annotations were significantly more accurate than both *structured* ones ($Z = 4.30, p < .001$) and *freeform* ones ($Z = 2.44, p < .05$). For the understandability rating, *semi-structured* annotations were significantly more understandable than both *structured* ones ($Z = 6.04, p < .001$) and *freeform* ones ($Z = 2.82, p < .05$), and *freeform* annotations were significantly more understandable than *structured* ones ($Z = 3.26, p < .005$). Thus, annotations from the *semi-structured* interface style seemed to outperform other annotations in terms of their informativeness, clarity, accuracy, and understandability for sighted users.

7.3.4 Evaluating Annotations with Blind and Low-Vision Users

I also conducted a study with BLV users to evaluate our annotations from the three different interface styles. Specifically, I wanted to discover whether there were particular needs or preferences of BLV users when listening to GIF annotations, and whether their perceptions differed from those of sighted users.

I recruited 11 BLV users (4 women, 7 men, mean age = 32.7) via social media platforms and word-of-mouth. Participants' demographic information is shown in Table 7.3. Nine participants self-identified as fully blind and two identified as having low vision. All participants owned at least one smartphone and were familiar with screen readers. All participants had encountered animated GIFs on their smartphones, and understood what a GIF was. The study was conducted remotely via Zoom, and each participant was compensated $15 USD for the study, which took less than one hour to complete.

As it was infeasible for each participant to read all 2754 annotations for the 34 GIFs, I manually selected five GIFs comprising 45 annotations for rating.[9] The five GIFs are

[9]I selected various GIFs with different features as described in Section 7.3.2.

Table 7.3: Demographic information of BLV participants.

ID	Age	Gender	Visual Impairment	Phone Platform(s)
P1	25	Man	Fully blind	iOS
P2	35	Man	Fully blind	iOS + Android
P3	28	Woman	Fully blind	iOS
P4	28	Man	Fully blind	iOS + Android
P5	32	Man	Fully blind	iOS
P6	24	Woman	Fully blind	iOS + Android
P7	68	Woman	Low vision (central vision loss)	iOS
P8	32	Woman	Fully blind	iOS
P9	23	Man	Low vision (glaucoma)	iOS + Android
P10	36	Man	Fully blind	Android
P11	24	Man	Fully blind	iOS

provided in Appendix C.1. I sent a Google Sheet to the participant ahead of the study session, which contained all annotations, with their orders for each GIF randomized to avoid order effects. During the study, the researcher asked the participants to first listen to the nine annotations of a GIF, and then rate them one-by-one in four respects (they could re-listen to the annotation when rating it): *informative*, *clear*, *understandable* and *overall preference*. (I removed *accurate*, as participants could not see the GIFs to verify accuracy.) I explained the meaning of each rating category, and the rating was from 1-10, as described above for sighted users. After listening to all nine annotations of a GIF, the participants then typed their ratings into the Google Sheet.

I collected $45 \times 11 = 495$ ratings over all annotations, the results from which are shown in Table 7.4. I performed analyses of variance based on mixed ordinal logistic regression [1,91],

Table 7.4: Means and standard deviations for annotation ratings by BLV participants. The scale is 1-10, with "10" meaning the most positive.

	Freeform	**Semi-structured**	**Structured**
Informative	6.9 (2.5)	7.5 (2.3)	6.4 (2.5)
Clear	6.9 (2.6)	7.7 (2.2)	6.1 (2.5)
Understandable	7.0 (2.5)	7.6 (2.2)	6.3 (2.5)
Overall Preference	7.0 (2.5)	7.6 (2.2)	6.2 (2.5)

treating the GIF ID as a random factor to account for repeated measures. I found that *Interface Style* had a significant effect on the informative ($\chi^2(2, N = 495) = 25.66, p < .001$), clear ($\chi^2(2, N = 495) = 41.27, p < .001$), understandable ($\chi^2(2, N = 495) = 28.95, p < .001$) and preference ($\chi^2(2, N = 495) = 30.39, p < .001$) ratings.

Post hoc pairwise comparisons corrected with Holm's sequential Bonferroni procedure [94] indicated that for the informative rating, *semi-structured* annotations were significantly more informative than both *structured* ones ($Z = 5.01, p < .001$) and *freeform* ones ($Z = 2.79, p < .05$). For the clarity rating, *semi-structured* annotations were significantly clearer than both *structured* ones ($Z = 6.31, p < .001$) and *freeform* ones ($Z = 2.87, p < .05$), and *freeform* ones were significantly clearer than *structured* ones ($Z = 3.56, p < .005$). For the understandability rating, *semi-structured* annotations were significantly more understandable than *structured* ones ($Z = 5.29, p < .001$), and the *freeform* ones were significantly more understandable than *structured* ones ($Z = 3.09, p < .01$). For participants' overall preference, *semi-structured* annotations were significantly preferred to both *structured* ones ($Z = 2.59, p < .05$) and *freeform* ones ($Z = 5.45, p < .001$), and *freeform* ones were significantly preferred to *structured* ones ($Z = 2.93, p < .01$). Thus, again I see the superiority of annotations coming from the semi-structured interface style, this time for BLV users.

During participant debriefing, I asked BLV participants for their rationale behind their ratings. Most participants did not enjoy the structured annotations. Although the structured descriptions contained important information listed as bullet points, the language

style felt "too robotic" (P1) and "just like a collection of keywords" (P2). By contrast, the *semi-structured* annotations had a more natural style, and because there were guidelines provided, the quality of these annotations was perceived as higher than the *freeform* annotations. Participants generally appreciated that contextual or cultural information (e.g., the background of a movie character) was provided in certain annotations, and they also liked the description of usage scenarios for the GIFs, commenting that while the usage scenarios could be "subjective" (P8), knowing how GIFs could be used was helpful for knowing when to send it to others or post it online. These findings were generally consistent with findings from prior work on image annotation [74].

Both evaluations with sighted and BLV users showed similar results: the annotations from the *semi-structured* interface style were better than the *freeform* and *structured* ones. I therefore incorporated the *semi-structured* interface style into Ga11y as its web-based human annotation interface for annotating animated GIFs.

7.4 Implementation of Ga11y

In this section, I present the implementations of the three major components of the Ga11y system: the Android client, the annotation web interface, and the backend server. I provide these details for completeness and reproducibility, and because realizing Ga11y required solving certain technical problems that constitute their own contribution.

7.4.1 The Android Client

To enable end users request GIF annotations, I developed an Android client that could (1) detect on-screen GIF elements, and (2) record animated GIFs and communicate with the Ga11y server.

In order to monitor on-screen contents for GIFs and insert buttons for requesting annotations, I adopted an "interaction proxy" [278] built on the Android Accessibility API. Specifically, I implemented an accessibility service that listens to screen updates signified by system *AccessibilityEvents* and captures the view hierarchy of the current app. By comparing the elements within the view hierarchy to a predetermined set of heuristics, I was

able to identify GIF elements contained within supported apps[10] because of the app-specific UI structures. The heuristics mainly drew upon the *ClassName*, *ViewIdResourceName*, and *ContentDescription* attributes of an element and its child elements, if any. These heuristics were determined by monitoring unique properties of GIF elements within each app using the Android Accessibility API.

For example, in Facebook Messenger, an element is considered to be a GIF when its bounding box contains one *ViewGroup* with the *ContentDescription* of "Sent photo message" and an *ImageView* with the *ContentDescription* of "Forward button."

Once a GIF element is identified, I insert a virtual request button immediately after it in the focus order. The inserted button would normally be announced by Android's TalkBack feature as a regular button (i.e., "Get GIF annotation, button"), but the button is controlled and monitored by the Android client instead of the current app. As a result, the user is prompted to "double-tap to request annotation." The user either double-taps to initiate a request for the corresponding GIF element, or continues left or right swiping to ignore the prompt, which would switch the focus back to the foreground app so that TalkBack can properly return to the app's element tree.

When the request button is triggered via double-tap, Ga11y initiates continuous screen capture using the Android MediaProjection API. This API captures the entire screen and sends it to the client; I then crop out only the GIF within the bounds of the identified element so as to preserve user privacy and reduce memory usage. I wait a fixed amount of time before ending the capture, determined to be five seconds in the studies. The captured image sequence is then compressed and sent to Ga11y's Annotation Server for further processing and analysis, as described below in Section 7.4.3. To prevent accidental actions that might move or occlude the GIF during the recording, the client produces an intermittent beep sound at twice a second to indicate that capture and analysis is in progress. If the user interacts with the device during this period of time, I consider the annotation request to be canceled and discard the captured images. Otherwise, the client will announce when the

[10]It currently support Facebook Messenger, Twitter, WeChat, Telegram, and Discord.

server returns its annotation to the user, which takes around 5 - 20 seconds depending on the network connection.

To speed up processing for previously annotated GIFs, I implemented a local cache in the client. After a successful annotation is returned by the server, I store the image hash values for each GIF image sequence, calculated using an average perceptual hash function [120].[11] I also store the returned annotation along with the hash values. Every time the user initiates an annotation request and the image capture starts, each captured frame is hashed using the same hashing function and the result is compared against all encountered image hashes. Once I find at least three successful matches, I consider the two sequences to be from the same GIF. The client then stops the screen capture and announces the cached annotation. For a previously encountered GIF, this reduces the annotation time to around one second.

The Ga11y client utilizes two special permissions: the accessibility service permission and screen capture permission. The user is asked to grant these two permissions after the Ga11y app is launched. If the user chooses not to grant either permission, the app will stay on the permission request screen and will not be able to provide GIF annotations. To support both English and Chinese annotations as used in the studies, the Ga11y client adapts to the system language based on the device's language and locale settings.

7.4.2 The Annotation Website

I implemented the annotation website using the React framework.[12] The annotation website contains two main pages (Fig. 7.3): the *annotate* page, for browsing the GIFs that are requested by users but have not been annotated yet, and the *browse* page, for browsing and editing existing GIF annotations. One can click a GIF on the *annotate* page to go to the annotation interface, which displays the *semi-structured* interface style with an open text box accompanied by annotation instructions, together with an example annotation (Fig. 7.6). The user can also upload their own GIFs with accompanying annotations via the

[11] A perceptual hash function generates the same hash values for images that look similar, which is suitable for comparing the visual similarity among a set of images. I used the average hash function because it was fast and was not sensitive to pixel noise generated during screen recordings.

[12] React: `https://reactjs.org/`

website by clicking the "upload" button.

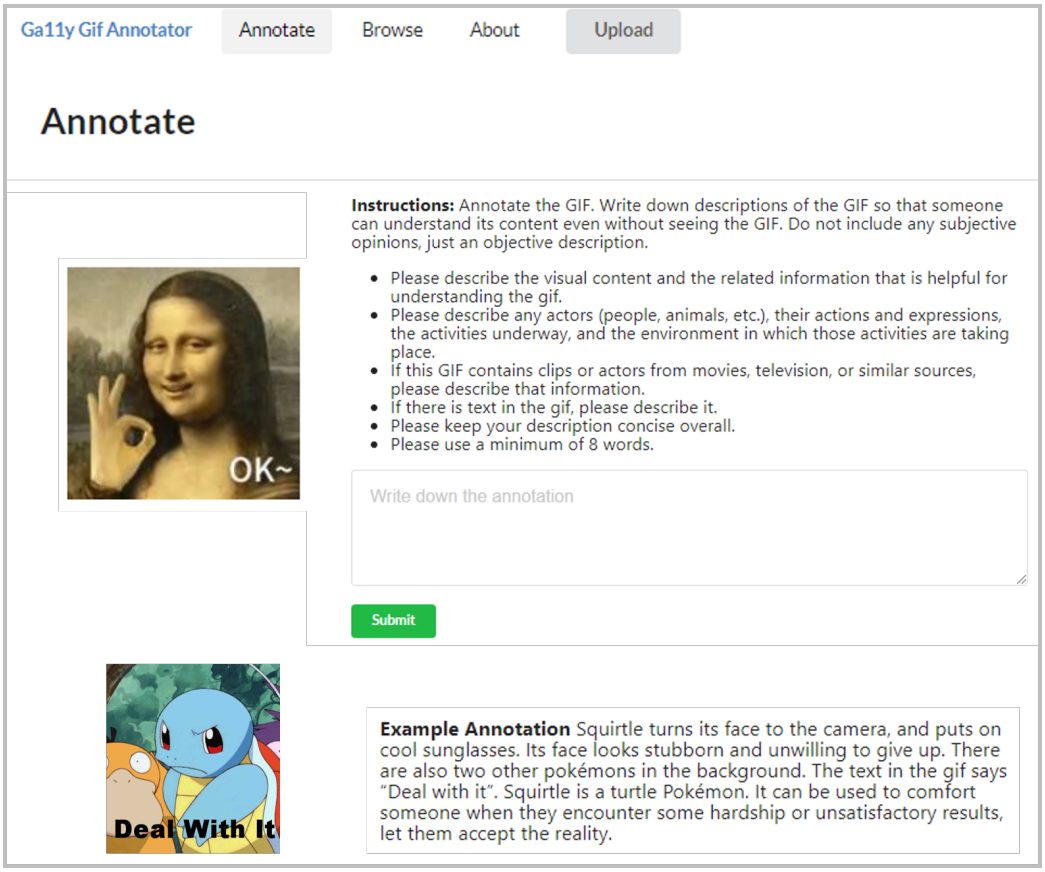

Figure 7.6: Ga11y's web-based annotation interface, which applies the *semi-structured* interface style based on the evaluation study described in Section 7.3.

7.4.3 The Annotation Server

Ga11y's backend server was implemented using the Tornado framework.[13] The server is responsible for two main tasks: (1) matching the requested GIF in the annotation database, and (2) requesting automated annotations for unmatched GIFs. I describe the various functions that the server performs.

[13]Tornado Web Server: `https://www.tornadoweb.org/en/stable/`

GIF Reconstruction. To handle an annotation request from an Android client, Ga11y's server first assembles captured screenshots into an animated video. As the GIF is captured directly from the screen, there is no indication about timing (e.g., the start and ending frame of the GIF). I therefore designed Algorithm 1 to identify the loop based on a sequence of GIF snapshots. For each snapshot of the GIF, I calculate its hash value by using the average perceptual hash function. I then put frames with the same hash value into the same bin, and derive the loop interval of the GIF based on the bins. Identifying the loop is important for the crowdworkers to annotate the reconstructed GIFs in the web client.

Algorithm 1 Identifying the loop from a sequence of GIF snapshots

```
frame_hash_dict ← {}
for frame in GIF.Snapshots do
    hash ← PerceptionHash(frame)
    if hash not in frame_hash_dict then
        frame_hash_dict[hash] ← [index_of_frame]
    else
        frame_hash_dict[hash].append(index_of_frame)
    end if
end for
duplicate_frames ← all items that have more than two values in frame_hash_dict
if duplicate_frames is empty then
    no loop in the GIF
else
    cnt ← the most frequent count of the item in duplicate_frames
    loop_frames ← items whose count equals cnt in duplicate_frames
    intervals ← time difference between consecutive pairs in each item of loop_frames
    loop_interval ← max(intervals)
    (loop_start_frame, loop_end_frame) ← the frame pair whose time difference is loop_-
    interval
end if
```

After the loop is identified, I further interpolate the GIF with snapshots that are outside the loop timespan to minimize any effects of lag from screen capture on the mobile client.

GIF Comparison. Because of the loop detection algorithm, the same GIF content might yield two loops starting at different frames. I thus used multiple frames within a GIF loop for comparison to increase robustness. I extract several keyframes of the GIF after reconstructing it by splitting the GIF loop into equal-lengthed subclips and taking the first frame of each clip as the keyframe. To compare whether two GIFs are equal, I compare the perceptual hashes of the two GIFs' keyframes: if any pair of the frame hash are identical, I treat the two GIFs as visually similar. I decided to have four keyframes for each GIF, as this number was empirically sufficient to identify similar GIFs and distinguish different ones.

GIF Annotation Database. After extracting the keyframes from the requested GIF, the server tries to match the keyframes with existing GIFs in the database. For each requested GIF, I store the perceptual hashes of its keyframes in a local Elasticsearch server.[14] If any of the keyframe's hash match an existing item in the database, the server will request the corresponding annotation; if not, the server will first store the keyframe hashes in the database, and request automated annotations for the GIF. The reconstructed GIFs are stored as videos in an AWS S3 database; all annotations are stored on the AWS DynamoDB database, which can be directly updated via the web annotation interface.

Requesting Automated Annotation. If a requested GIF has not yet been manually annotated, the server will request automated annotations by sending one of the keyframes to the Google Vision service[15] and the Microsoft Azure Computer Vision service.[16] The former service provides the objects and text in the image, while the latter service generates a caption of the image. The server then combines the two recognition results into a single description as Ga11y's automated annotation. I used the Google Translation API [17] to

[14] `https://www.elastic.co/`

[15] `https://cloud.google.com/vision`

[16] `https://azure.microsoft.com/en-us/services/cognitive-services/computer-vision/`

[17] `https://cloud.google.com/translate`

translate the annotation to other languages, namely Chinese in the user study.

7.5 Ga11y Evaluation

I conducted a user study to evaluate Ga11y. Specifically, I was interested in three questions: (1) How do BLV users experience Ga11y's usability? (2) How do BLV users perceive the quality of human-labelled and machine-generated annotations? (3) How might Ga11y affect BLV users' online communication experiences? I describe the study to answer these questions below.

7.5.1 Participants

I recruited 12 BLV participants, with four from the United States (P2, P4, P9, and P10 from the first study in Table 7.3) and eight from China, whose demographic information is listed in Table 7.5. Participants' average age was 29.1 years ($SD = 5.6$). All participants were familiar with GIF images generally and with using a mobile screen reader. I recruited Chinese participants to understand how Ga11y performs beyond the English language, including for pictographic languages. Participants received $25 USD or 150 CNY for the one hour study.

7.5.2 Apparatus

I hosted the Ga11y service on a server from a research institute, which had a public IP address to which participants' clients could send requests. All participants used the Ga11y service on their own Android devices. To send GIF content, for American participants, I used Twitter as the communication platform; for Chinese participants, I used WeChat.

7.5.3 Procedure

Study sessions were conducted remotely via Zoom, and were audio-recorded for further analysis. I asked participants to turn up their screen reader volume so that the experimenter could hear the output. I first sent participants the Android client application package (APK) and instructed them how to install and configure it. I then asked participants several

Table 7.5: Demographic information for the study participants. All participants owned an Android device.

ID	Age	Gender	Visual Impairment
CN_P1	23	Man	Fully blind
CN_P2	24	Man	Low vision (cataract)
CN_P3	30	Man	Low vision (traumatic visual loss)
CN_P4	26	Man	Low vision (cataract)
CN_P5	29	Man	Low vision (amaurosis)
CN_P6	40	Man	Fully blind
CN_P7	31	Woman	Fully blind
CN_P8	24	Man	Low vision (optic atrophy)
US_P1	35	Man	Fully blind
US_P2	28	Man	Fully blind
US_P3	23	Man	Low vision (glaucoma)
US_P4	36	Man	Fully blind

questions regarding their existing experiences and difficulties with online GIFs. After set-up was complete, I sent a test GIF to participants and let them try making an annotation request. After participants confirmed that they were familiar with the Android client, I began the formal study session.

During the formal part of the study, I sent participants two groups of GIFs, five that were already annotated by Turkers from the first study, and five that were not manually described and would therefore elicit machine-generated annotations. For context, the machine-generated annotations provided for items (f) - (j) in Figure 7.7 always began with "automatically generated description"; their detailed annotation is provided in Appendix C.2. All GIFs in Figure 7.7 were chosen by the experimenters so that they exhibited different features as specified in Section 7.3.2. The order of the GIFs were randomized for each participant. For each GIF sent to a participant, the participant was told to operate their

Figure 7.7: Ten GIFs used for the evaluation task. GIFs **(a)** – **(e)** had human annotations already on the server, whereas **(f)** – **(j)** did not. **(a)** Cartoon cat sticker with text; **(b)** the infinite dog; **(c)** Jimmy Fallon with text; **(d)** "This is fine" meme; **(e)** rapidly typing dog; **(f)** vibing cat; **(g)** telescope sticker with text; **(h)** smiling *SpongeBob SquarePants*; **(i)** plants growing; **(j)** cartoon rabbit sticker with text.

screen reader to perform an annotation request. After all the annotations were requested, participants rated Ga11y on the System Usability Scale (SUS) [29]. Participants were also interviewed for their feedback on using the Ga11y system. Participants were encouraged to continue using the system after the study, and to provide their feedback when using it, although this was optional.

7.6 Results

In this section, I present the study results. In general, all participants were positive about Ga11y and said they would like to have it on their phone. I calculated the SUS score on a $0 - 100$ scale, where higher scores indicate "more usable."[18] The average SUS score was 89.1 ($SD = 9.0$), indicating high usability of the system. I present the detailed results in this section.

[18] `https://measuringu.com/sus/`

7.6.1 Current Experience with GIFs

All participants said that they had encountered GIFs online, and most GIFs were not understandable because of the lack of annotations. US_P2 mentioned that he had encountered many GIFs on Twitter, and although some of them had descriptions, most of the descriptions were automatically generated and only contained one or two keywords, which was not helpful. US_P3 mentioned that with keyboards like Google's *Gboard*, he could select and send GIFs. However, the problem was that while the GIFs had keyword annotations within the keyboard, their annotations were lost once the GIFs were sent to a chat. All participants from China mentioned that they encountered GIFs and stickers every day as people in group chats liked to send funny GIFs and compete with each other, which has been referred to as "sticker competitions" [279]. However, because of the lack of annotations, the participants have not been able to participate in such activities. CN_P2 mentioned that he only used certain GIFs for simple expressions that he was familiar with, such as "thank you" and "okay," and CN_P1 mentioned that he only used emojis and not GIFs, as emojis are better annotated.

Participants tried to ask a sighted friend for help when they encountered unfamiliar GIF contents. As CN_P4 said, "I will just ask others the meaning of the GIF they sent if I don't understand." However, most of the time participants reported just ignoring the GIFs and stickers they encountered.

7.6.2 Ga11y Usage "In the Wild"

To see how Ga11y might help participants in their daily internet usage, I logged each participant's usage of Ga11y for three days after the formal study, with their knowledge and permission. As noted above, this part of the study was optional, but I encouraged participants to continue to use Ga11y after the study session. In total, the Ga11y server received 548 annotation requests, with 458 requests from Chinese participants and 90 from American participants. Among them, 203 requests included new GIFs that were not annotated in the server, and 345 requests included GIFs that were already in the database. For the newly requested GIFs, after the machine-provided annotations were sent, we, the researchers, pro-

vided the subsequent human annotations through Ga11y's web interface. I also assigned a unique ID for each participant in the request log, finding that each user sent about 13.5 requests per day on average ($SD = 16.6$), indicating that Ga11y was frequently used and could potentially play an important role in participants' online communications.

7.6.3 Qualitative Feedback

I collected participants' feedback after the formal study and during the three-day "in the wild" usage period, and coded the feedback using affinity diagramming [188]. Three main topics emerged: *Perceptions of human- and machine-generated annotations*, *usability suggestions*, and *the effects of Ga11y on online communication*. I take each of these in turn.

Perceptions of Human- and Machine-Generated Annotations

All participants felt that the human-generated annotations were helpful in understanding animated GIFs, especially when context, such as the background of a movie or cartoon character, was provided. Participants also appreciated the provision of machine-generated annotations, although these descriptions were thought to lack a "human touch" (US_P1). Two participants also appreciated that there was always a prefix "automatically generated description" for the machine-generated annotations, allowing them to adjust their expectations of the annotation quality.

However, while the four participants from the U.S. were all comfortable with the detailed level of the annotations, three Chinese participants commented that the translated annotations were too long and contained too much detail. For example, CN_P8 said that some of the detail could be omitted, and the language style of the transcribed annotations was not very natural. On the other hand, all participants appreciated having human volunteers providing annotations. They also appreciated the explicit usage scenarios offered in the annotations, even though the usage scenarios were clearly "subjective" (CN_P8).

Usability Suggestions

All participants found the interaction with Ga11y's Android client interface for requesting annotations to be intuitive, where they could easily choose whether to request an annotation or not. CN_P1 said, "The interaction [of pressing a button alongside a GIF] is far easier than other accessibility apps," referring to other image recognition apps in which he had to "take a screenshot and switch to the app for recognition results." Two participants (CN_P3, CN_P7) also suggested that the Ga11y service could automatically request the annotations, and attach annotated labels to the GIFs without the user having to request them interactively. I leave implementing this option for future work.

As two participants mentioned that the annotations were too lengthy, they suggested adding functions to adjust the detail level of the annotations. CN_P6 suggested that the system could first read out a brief version of the annotation, including only the most important content such as the main characters and the meaning of the GIF. Subsequently, the user could click a button to listen to the full annotation if they wanted. CN_P8 suggested a similar idea, where the user could adjust the level of annotation detail in the app settings.

The Effects of Ga11y on Online Communication

All participants appreciated Ga11y for enabling them understand unlabelled GIFs. As US_P3 mentioned, "Although sometimes the annotations are automatically generated, it offers information which is not accessible at all before." US_P1 said that he was a user of many social media platforms, and "it is important to speak the language others are speaking." CN_P4 also talked about the helpfulness of being able to understand GIFs: "I was very careful about using and sending new GIFs, as I am worried that I might send something inappropriate. And each time I find others are sending stickers and GIFs in the group chat, I feel left behind. Having the annotations can definitely help me understand what they are saying, and give me the sense of belonging." Taken together, the results indicate that although Ga11y could certainly be improved, it had a positive impact on the accessibility of online GIFs and participants' online communication generally.

7.7 Discussion

In this work, I presented Ga11y, a GIF annotation system combining the power of crowdsouring with machine intelligence. With the three components of the Ga11y system, I was able to not only provide on-demand GIF annotations to BLV users on their mobile devices, but also to provide a *semi-structured* annotation interface style that yields high quality GIF annotations. On the client side, I utilized the accessibility framework on the Android platform, and created an interaction proxy that allowed users to request on-screen contents without switching the application; on the server side, I designed a comparison algorithm for handling screen-recorded GIFs. By conducting a study with both English and Chinese language speakers, I validated the usefulness and usability of Ga11y for enabling GIF accessibility.

From the study results, I found that the current computer vision based techniques usually only generated high-level annotations without enough details, and many of the characters/objects were misrecognized, which confused participants. This also indicated that although for static images, auto-generated descriptions could be of high quality and were already used on commercial platforms (such as iOS and Chrome), they were not capable to handle the GIF content yet. On the other hand, participants appreciated the timeliness of the machine-generated annotation, and found it was especially useful for GIFs containing text, as the text could convey important information even if the visual content was misrecognized.

As for the human-generated annotations, I found it interesting that two participants complained that the annotations were too lengthy and detailed, as they were more interested in what the GIF tried to convey, rather than the content itself. Two participants also commented that they needed to listen to the same annotation multiple times, as the text was too long to remember. As a future step, it is worthwhile to investigate how users' annotation preferences vary for different usage scenarios (e.g. online messaging, social media post, blog/article, etc).

We also found that the semi-structured prompt for annotating GIFs was recognized as

providing the highest description quality by both sighted and BLV people, compared to structured and freeform prompts. This finding contradicts the claim of previous work [153], where the authors found structured prompts yielded better annotations for scientific figures. One possible explanation is that structured annotations sound less ”human”, and contain many redundant information [135] in comparison to the semi-structured annotations. In addition, the participants commented that since GIFs often contained nuanced expressions, or culture-related background information, it was important to have a ”human touch” in the explanation. Hence the annotation design can be deeply situated in the task context: for contents that has objective descriptions such as scientific figures or charts, structured annotation might provide ease to both annotators and readers; for contents that are subjective and require personalized explanations, the semi-structured annotation design might be the best.

The technical solution provided by this work did not fully investigate the privacy issues, and we would expect a more sophisticated way to handle the user data in the future. For now, the screenshots are sent to the remote server and displayed in the web client. This approach might leak the users' browsing data, or the source app they are using to capture the GIFs. In the future, a local cache containing hash + descriptions of most popular GIFs might mitigate the problem, as most of the requests can be processed offline. A better GIF recognizing algorithm can also be applied to crop unrelated portions of the screenshots.

The popularity of the GIF image format has created a unique aspect of internet culture, and the ability to understand GIFs well is a key to participating in that culture. As GIFs often contain subtle emotions and rich expressions, it is necessary to generate human annotations to convey human feelings. By providing a publicly accessible annotation service, I can also raise awareness of the need for GIF accessibility. Similarly, GIF platforms such as GIPHY[19] and Tenor[20] could also provide ways for users to annotate a GIF when uploading it or selecting it for use. These platforms already contain user-generated metadata about GIFs such as tags, which could be utilized to train machine learning models that generate better

[19] `https://giphy.com/`

[20] `https://tenor.com/`

annotations.

Although the annotations in Ga11y are provided as text, there are richer ways to represent a GIF, such as with audio. Cole et al. [74] suggested that including an audio description, such as any sound accompanying the source clip of a GIF, could enrich emotive understanding. Future versions of Ga11y could also employ automated methods to match GIFs with existing videos and extract the audio as part of the annotation. Of course, the use of audio also poses certain accessibility barriers, and would need to be addressed, perhaps again leveraging text like in Ga11y.

7.8 Limitations

As with any research project, there are several limitations of this work.

First, because I used the Android accessibility framework, participants found the setup of the app to be complex, as they needed to go through multiple permission-granting steps. Furthermore, the screen capture service I used for GIF recording was not entirely stable and could be shut down by the system unexpectedly. Capturing the whole screen also raised some privacy concerns by participants. The experience of Ga11y could have been smoother if the operating system supported annotation requests natively.

Second, Ga11y only supported GIF recognition in a limited set of mobile apps by recognizing their user interface structures. This limitation could be improved if there were a universal interaction that could be employed, such as a hard button or a gesture, to trigger Ga11y's screen recording. In this way, the user could perform annotation requests on any screen element, and there would be no need to recognize on-screen GIFs.

Third, although I logged the annotation service usage during the "in the wild" period of three days after the study sessions, conducting a long-term field deployment would reveal more insights on how participants use Ga11y in their daily lives. Fourth, as Ga11y's annotation web interface is not yet publicized, it remains an open question as to whether people are willing to contribute annotations, and the annotation quality on a large scale remains an open question.

Finally, the client was implemented on the Android system, thus I did not gather the feedback from iOS users. However, I am aware of the huge BLV population using VoiceOver as their main screen readers, and the experience might be different from the Android TalkBack. That said, the iOS system is very strict on the accessibility frameworks and third party apps, hence I chose Android as the main platform.

7.9 Conclusion

In this work, I presented *Ga11y*, a GIF annotation system that utilizes both crowdsroucing and machine intelligence to help blind and low vision (BLV) users understand the content and meaning of animated GIFs. Ga11y contains three components, including a mobile client, a data processing and storage server, and a GIF annotation website. In order to have high quality annotations, I evaluated different annotation interface styles with both sighted and BLV users, and applied the best style—one utilizing an open text field with guiding prompts—on Ga11y's annotation website. We also implemented GIF processing for the Android client, including GIF recording, reconstruction, and comparison, to support annotation requests. The user study with both American and Chinese participants demonstrated that Ga11y was perceived as highly usable, and the combination of human- and machine-generated annotations was an effective solution for aiding in GIF understanding. We hope that by open-sourcing the implementation, Ga11y will encourage GIF platforms and system providers to consider designing annotation supports for GIFs and stickers, and maybe for other forms of dynamic content. Everyone should be able to fully participate in the online culture enabled by animated GIFs.

Part III

EVALUATING INTELLIGENT COMMUNICATION SYSTEMS

Besides building systems to support intelligent communication experience, we need to understand their performance and impacts before deploying them on a large scale. In this part, I present models and metrics for the evaluation of text entry systems, and an empirical study to measure the effect on user behavior of different emoji suggestion mechanisms.

Chapter 8

BEYOND THE INPUT STREAM: MAKING TEXT ENTRY EVALUATIONS MORE FLEXIBLE WITH TRANSCRIPTION SEQUENCES

Method-independent text entry evaluation tools are often used to conduct text entry experiments and compute performance metrics, like words per minute and error rates. The input stream paradigm of Soukoreff & MacKenzie [138, 201] remains prevalent, which presents a string for transcription and uses a strictly serial character representation for encoding the text entry process. Although an advance over prior paradigms, the input stream paradigm is unable to support many modern text entry features. To address these limitations, I present transcription sequences: for each new input, a snapshot of the entire transcribed string unto that point is captured. By comparing adjacent strings within a transcription sequence, I can compute all prior metrics, reduce artificial constraints on text entry evaluations, and introduce new metrics. I conducted a study with 18 participants who typed 1620 phrases using a laptop keyboard, on-screen keyboard, and smartphone keyboard using features such as auto-correction, word prediction, and copy/paste. I also evaluated non-keyboard methods Dasher, gesture typing, and T9. The results show that modern text entry methods and features can be accommodated, prior metrics can be correctly computed, and new metrics can reveal insights. I validated the algorithms using ground truth based on cursor positioning, confirming 100% accuracy. I also provide a new tool, TextTest++, to facilitate web-based evaluations.[1]

[1]This chapter is adapted from: Mingrui "Ray" Zhang, Jacob O. Wobbrock. (2019). Beyond the Input Stream: Making Text Entry Evaluations More Flexible with Transcription Sequences. Proceedings of the 32nd Annual ACM Symposium on User Interface Software & Technology (UIST '19). New York: ACM Press.

8.1 Introduction

Text entry remains fundamental on most computing platforms, from desktops to tablets to game consoles to smartphones. Increasingly, the need for text entry extends to new platforms, such as interactive tables [66], smartwatches [258], and augmented reality [259]. As a result, researchers, developers, and product innovators still regularly create new text entry methods.

When seeking to quantify the performance of their new methods, 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 [136], various error rates [201], and more) without knowing the mechanisms by which that text is produced. (We refer to this as the "black box" consideration, and discuss it below).

To compute method-independent metrics, evaluation testbeds must artificially constrain the text entry evaluation process. Measuring text entry error rates presents a particular challenge, and is a major reason for artificial constraints, because we must infer a user's intention in order to detect deviations from it [59, 138, 247]. The prevalent evaluation paradigm addressing this need is that of Soukoreff & MacKenzie [200, 201], 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.

In this work, we present a new underlying model that supersedes the IS model for general-purpose method-independent character-level text entry evaluation. Specifically, we present

an approach that replaces the input stream with transcription sequences, or "T-sequences" for short. In brief, a T-sequence is a sequence of snapshots of the entire transcribed string after each text-changing action is taken by the user. Every pair of successive snapshots are then compared to compute character-level text entry metrics. We show that, unlike the IS paradigm, the T-sequence paradigm can handle the modern text entry features that have been disallowed thus far. We validated our measurement algorithms using ground truth cursor-position information, confirming 100% accuracy. We also show that T-sequences not only accommodate prior metrics from the IS paradigm, but also enable new metrics. Finally, we offer a web-based successor to the TextTest desktop application [247] called TextTest++[2], which encapsulates our approach and enables text entry innovators to study their inventions on any platform or device capable of running a web browser.

The contributions of this work are: (1) The detailed elucidation of a new general-purpose method-independent model of the text entry process based on T-sequences, superseding the IS model; (2) New algorithms for computing text entry metrics, both extant and novel; (3) Empirical results from a study of 18 participants confirming the ability of our model to handle modern text entry behaviors; (4) An evaluation of T-sequences with non-keyboard techniques Dasher [234], gesture typing [121,267], and T9[3]; and (5) The web-based TextTest++ evaluation testbed capable of conducting and analyzing text entry studies. This work will be useful to text entry method creators wishing to evaluate their new methods without imposing undue constraints.

8.2 A Method-Independent Abstraction of Text Entry Evaluation

As discussed above, any pre-existing text entry evaluation testbed must, by definition, operate without any method-specific knowledge of the text entry method it evaluates. As the specific features of the text entry method are unknown, the method can be considered a black box (Figure 8.1). An abstraction containing three parts is therefore implied: (1) the user's actions on the text entry method, (2) the black box text entry method itself, and (3)

[2]Available at `http://depts.washington.edu/acelab/proj/texttestpp/`

[3]`https://en.wikipedia.org/wiki/T9_(predictive_text)`

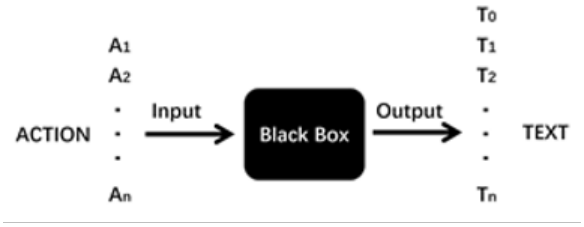

Figure 8.1: An abstraction of the method-independent text entry evaluation process. A user takes actions on a black box text entry method, which generates text as output, which is received by an evaluation testbed for logging and analysis.

the resulting text output that enters the method-independent evaluation testbed. In short, (1) acts on (2) to produce (3).

After each input action that alters the current transcription, rather than appending a character (or BACKSPACE) to the end of an input stream, the entire transcription at that moment is captured. This transcription, denoted Ti, is one entry in a sequence of transcriptions, or "T-sequence," captured after each action. Below is an example for entering "computer":

Transcription (T_i)	**Action** (A_i)
$T_0 :< null >$	$A_0 :< begin >$
T_1 : `c`	A_1 : insert c
T_2 : `co`	A_2 : insert o
...	...
T_8 : `computer`	A_8 : insert r
T_9 : `computerr`	A_9 : insert r
T_{10} : `computer`	A_{10} : delete r

In the example above, the last "r" is typed twice by mistake and corrected. Each transcrip-

tion T_i for i > 0 beyond the null transcription (T_0) is the result of a corresponding action A_i. An action should be thought of more broadly than just a concrete physical movement like "typing a key" or "making a stroke gesture;" rather, it is any action that transforms T_{i-1} into T_i. The next subsection provides an abstraction of actions. We build on this abstraction when analyzing T-sequences to extract text entry metrics.

8.2.1 An Abstraction of Actions

In the era of ubiquitous computing, there are any number of actions that might change text: typing a key, touching a screen, making a gesture, rotating a watch bezel, or typing CTRL + BACKSPACE on Windows to delete an entire word. The black box abstraction only considers before-and-after text transcriptions resulting from each action. It only knows what the method-independent changes to text are, not the method-specific mechanisms by which the user brought about those changes. (Of course, the same was true of the input stream paradigm, but it could only handle single character serial entry and removal actions.) Formally, we classify each action A_i that turns transcription T_{i-1} into T_i as one of three classic types [138, 247], based only on the changes to successive strings, not from any method-specific insights.

Insertion. An insertion is an action that adds one or more characters anywhere within or to either end of the current text, without removing any of that text. An insertion action is parameterized with two values: a zero-based start index where the insertion occurs (index zero is before the first character), and the string to be inserted. For example:

T_{i-1}: `All oads`
T_i: `All roads`

In this example, action Ai would be an insertion annotated with (4, "r"), meaning it starts at zero-based index 4 and the inserted string is "r".

Deletion. A deletion is an action that removes a substring anywhere within or at either end of the current text, without modifying other parts of that text. A deletion action is parameterized with two values: a zero-based start index at which the deletion occurs, and

the number of characters forward from that point that are deleted. For example:

T_{i-1}: `All roads`

T_i: `All ads`

Above, action A_i would be a deletion annotated with (4, 2), meaning it starts at zero-based index 4, and two characters are deleted ("ro").

Substitution. A substitution is an action that composes a deletion and an insertion into one action. In a substitution, a substring is removed simultaneously as a new string, not necessarily of the same length, is inserted. A substitution action is parameterized with three values: a zero-based start index at which the deletion occurs, the number of characters forward from that point that are deleted, and the new string to be inserted. For example:

T_{i-1}: `All road lead to Rome`

T_i: `All paths lead to Rome`

In this example, action A_i would be a substitution annotated with (4, 4, "paths"), meaning it starts at zero-based index 4, deletes four characters ("road"), and inserts "paths".

For insertions or deletions, changes must happen in only one contiguous place in the text. If changes happen in multiple non-contiguous places simultaneously, we consider the change a substitution. For example, if T_{i-1} = "all roads" becomes T_i = "ball broads" by inserting a "b" before both "all" and "roads", we consider the change a single substitution. Other special cases like transpositions, where chunks of text are moved to another place (e.g., via drag-and-drop, a method-specific action), are also substitutions.

With this level of abstraction established, we are now ready to describe the transcription sequence paradigm and the algorithms for extracting text entry metrics from it.

8.3 The Transcription Sequence Paradigm

We formalize the transcription sequence, or "T-sequence," paradigm using examples, and show how we infer actions from T-sequence changes. We also show how we can extract the old input stream (IS) from T-sequences that permit it.

8.3.1 Character Classes

Within the T-sequence paradigm, we reuse three of the four Soukoreff & MacKenzie character classes [201]: Correct (C), Incorrect and Not Fixed (INF), and Incorrect and Fixed (IF). We discard Fixes (F) because our black box abstraction and action definitions neither need nor permit method-specific information on the nature of fixes. To recap:

C – Correct characters in the final transcribed text, T.
INF – Minimum string distance (MSD) between P and T.
IF – All deleted characters in the entire sequence, T_0 to T.
By using these three classes, we can continue to calculate the uncorrected (UER), corrected (CER), and total (TER) error rates:

$$\begin{aligned} C &= Max(|P|, |T|) - MSD(P, T) \\ INF &= MSD(P, T) \\ UER &= INF/(C + INF + IF) \\ CER &= IF/(C + INF + IF) \\ TER &= (INF + IF)/(C + INF + IF) \end{aligned} \tag{8.1}$$

8.3.2 An Example of a Complete Transcription Sequence

Consider the following T-sequence representing the entry of P = "All roads lead to Rome". Note that whatever hypothetical text entry method is being used here, it has capabilities that enable it to go beyond just single-letter entry. In the Action column, "I" is insert, "D" is delete, and "S" is substitute.

Transcription (T_i)	**Action** (A_i)
$T_0 :< null >$	$A_0 :< begin >$
$T_1 :$ `A`	$A_1 : I(0, A)$
$T_2 :$ `All`	$A_2 : I(1, ll)$

T_3 : `All Rome`	$A_3 : I(3, \_Rome)$
T_4 : `All roads`	$A_4 : S(4, 4, roads)$
T_5 : `All roads l`	$A_5 : I(9, \_l)$
T_6 : `All roads let`	$A_6 : I(11, et)$
T_7 : `All roads le`	$A_7 : D(12, 1)$
T_8 : `All roads lead to`	$A_8 : I(12, adto)$
T_9 : `All roads lead to rome`	$A_9 : I(17, \_rome)$
T_{10} : `All roads lead to Rome`	$A_{10} : S(18, 4, Rome)$

Note that all of "rome" was replaced by "Rome" with the final action (A10), perhaps by a whole-word paste operation, auto-correction, or a spell-checker menu selection. The above example yields character classes as follows:

- C = {"All roads lead to Rome"}
- INF = { }
- IF = {"Rome", "t", "rome"}

Using Eqs. (3), the uncorrected error rate (UER) is 0.00%, the corrected error rate (CER) is 29.0%, and the total error rate (TER) is therefore 29.0%.

8.3.3 Inferring Actions from a Transcription Sequence

Consistent with our black box abstraction and our goal to improve method-independent approaches to text entry evaluation, we do not have ground truth information as to the inputs performed by the user or the actions that transform one transcription (T_{i-1}) into the next (T_i). We simply see the sequence of transcribed strings, and infer actions from successive transcriptions. For example:

T_{i-1} : `Thai`

T_i : `Thanks` A_i :???

In this example, the user might have used auto-correction, in which case A_i is S(0, 4, "Thanks"). Or, the user might have selected the "i" and pasted "nks", which would be S(3, 1, "nks"). To know the truth of A_i, one would have to build a textbox capable of receiving method-specific signals and know how to interpret them.

Instead, it is possible to infer actions from changes between consecutive T_{i-1} and T_i. The rationale is that in most character-level input methods, characters only change over a contiguous range, not at multiple simultaneous disjoint indices; thus, it is sufficient to get the character change information. Doing so works on any platform, as all textbox widgets provide a property to inspect their text. We can then build a testbed for text entry evaluation that remains independent of any specific text entry method's features.

To infer the most likely actions taken given T_{i-1} and T_i, we created an algorithm called INFER-ACTION. The algorithm finds the minimum modification necessary to turn transcription T_{i-1} into T_i. In the example above, the algorithm would favor S(3, 1, "nks") over S(0, 4, "Thanks") because only one character, the "i", is changed in the former.

Our INFER-ACTION algorithm generally works as follows: Given the two strings T_{i-1} and T_i, it first compares them from their beginnings, stopping when it finds a mismatch at index p1. Then it compares the strings from the end, again stopping when it finds a mismatch, now at p2. Based on the relationship of p1 and p2, the algorithm determines whether the change is an insertion, deletion, or substitution. For example, if p1 equals the length of T_{i-1}, p2 equals the length of T_i, and T_i is longer than T_{i-1}, the action A_i inserted (p2 – p1) characters at the end of T_{i-1}.

Note that our INFER-ACTION algorithm is triggered only when there are changes in the text, i.e., when T_{i-1} becomes T_i. If a user presses the CAPS LOCK key, drag-selects text with the mouse, or moves the text cursor with the arrow keys (all of which are method-

specific actions), our algorithm would not consider a change, as the entered text remains unchanged. Evaluators wishing to go beyond quantifying general text entry performance to understanding method-specific behaviors (e.g., the number of times CAPS LOCK was pressed on a keyboard) would need to build custom testbeds to capture such metrics, as they have always had to do.

To verify the correctness of our INFER-ACTION algorithm, we obtained ground truth in our study (explained below) by monitoring the text cursor movements with the JavaScript textbox properties *selectionStart* and *selectionEnd*. This information is sufficient for ground truth because when text changes, the text cursor appears at the end of the latest change. Our results show that INFER-ACTION, working only with T-sequence string pairs as described above, correctly inferred 100% of all actions in our study. But INFER-ACTION is not perfect. For example, the replacement of "rome" with "Rome" in the example above would be inferred as a substitution of only one letter, i.e., S(18, 1, "R").

8.3.4 Recovering the Input Stream From Transcription Sequences

Although in the new T-sequence paradigm, method-specific keystrokes like BACKSPACE are no longer separately distinguished, the input stream (IS) can still be recovered if it is known that users behaved as they did in the old IS paradigm, i.e., if users entered text sequentially at the end of the IS and BACKSPACE was their only form of error correction. Specifically, when there is only one character changed at the end of the IS with each action, we can simply recover the action by examining the difference between adjacent transcriptions. Note that substitution actions cannot directly be performed in the IS paradigm, only insertions and deletions at the end of the currently entered text. The entire IS can be rebuilt, as in the following example:

Transcription (T_i)	**Action** (A_i)
$T0 :< null >$	$A0 :< begin >$
$T1$: `t`	$A1 : I(0, t)$
$T2$: `th`	$A2 : I(1, h)$
$T3$: `thw`	$A3 : I(2, w)$
$T4$: `th`	$A4 : D(2, 1)$
$T5$: `the`	$A5 : I(2, e)$
IS : `thw<e`	

Now that we have presented T-sequences, we show how they can be used to formulate the Incorrect and Fixed (IF) class and its separation into two subclasses, IF_c and IF_e.

8.4 Extensions to the Incorrect-and-fixed Class

In text entry transcription studies, a common error correction behavior is to extensively use BACKSPACE, which often leads to deleting already-correct characters [202]. Consider this input stream:

quack $<<<<$ **uick**

Above, "uack" is erased by four BACKSPACEs, and "uick" is added. However, the "u" was correct despite being backspaced. Similarly, the "ck" were correct despite being backspaced. Should they be counted as errors? To address this question, Soukoreff & MacKenzie [202] created new IF_c and IF_e subclasses of their Incorrect and Fixed (IF) class. IF_c and IF_e respectively stood for "incorrect-and-fixed correct characters" and "incorrect-and-fixed errors." This separation of IF into these two new subclasses enabled more accurate error rate calculations.

However, because IF_c and IF_e were based on the IS paradigm, prior work [202] assumed BACKSPACE was the only way to correct errors. In the more flexible T-sequence paradigm,

a new algorithm is needed to separate IF into IF_c and IF_e. Therefore, I use a modified version of the Needleman-Wunsch algorithm [159]. As before, I assume no method-specific knowledge of the actions performing the correction of text.

8.4.1 Modified Needleman-Wunsch Algorithm for IF_c and IF_e

The Needleman-Wunsch algorithm [159] is a dynamic programming algorithm used to align biological sequences. The algorithm is more flexible than the algorithm used in the calculation of the minimum string distance (MSD) [200], i.e., the Levensthein string alignment algorithm [125].

In the Needleman-Wunsch algorithm, there are three types of character comparisons: match, mismatch, and gap, each of which are assigned scores during the alignment. Match means two characters are the same; mismatch means they are different; gap means one letter in one string lines up with a gap in the other string. For example, consider the alignment of "dynamic" and "plastic":

```
dyna-mic
-plastic
```

In this example, a '–' means a gap, of which there are two. There are also three matches and three mismatches.

I modified the original Needleman-Wunsch algorithm to not favor aligning string beginnings and endings by penalizing start and end gaps that occur after matches have been made while there are still characters left to match. This modification was necessary because in text entry transcription tasks, users try to align with the presented string (P). For example, if P = "true treasure" and T = "treasure", the unmodified Needleman-Wunsch algorithm would produce this:

```
true treasure
tr-e----asure
```

However, the modified algorithm produces this result:

```
true treasure
-----treasure
```

I also added a gap penalty to the algorithm, which assigns different penalties to the opening or extending of a gap. The purpose of this penalty is to promote the formation of connected gaps in the alignment, i.e., favoring long contiguous gaps over disjointed short gaps. Consider aligning "Massachusetts" and "Massetts". If the penalty is the same for opening a new gap as it is for extending an existing one, then the result will be:

```
Massachusetts
Ma-s----setts
```

But if the penalty for opening a new gap is higher than for extending an existing one, the result will be:

```
Massachusetts
Mas-----setts
```

In the version of the Needleman-Wunsch algorithm, I set the score for match as +3, mismatch as -2, opening a new gap as -2, and extending an existing gap as -1.

8.4.2 Detecting IF_c and IF_e Characters

The solution to finding correct (IF_c) and erroneous (IF_e) characters within the Incorrect and Fixed (IF) class [202] uses the modified Needleman-Wunsch algorithm [159]. Specifically, given a T-sequence, for each action Ai that is delete or substitute, I perform an alignment between Ti-1 and P, the presented string; I then find the corresponding part Δ in P that aligns with the deleted characters in Ti-1. In the deleted substring, the characters that match Δ are in IF_c, and other characters are in IF_e. An example illustrates:

P: `All roads lead to Rome`

T_{i-1}: `All toads`

T_i: `All` A_i: D(4, 5)

The optimal alignment of P and Ti-1 is:

```
All roads lead to Rome
All toads-------------
```

Now, the part of P that aligns with the deleted characters "toads" is Δ = "roads". The "r" and "t" do not match, and are classified as errors in IF_e. The "oads" suffix matches and is therefore classified as having correct characters in IF_c.

8.5 New Metrics Based On Transcription Sequences

To recap: thus far, I have shown how T-sequences can enable the calculation of error rates from the input stream (IS) paradigm of Soukoreff & MacKenzie [200, 201]. These error rates are defined in Eqs. 8.1, above. I have also shown how to separate the Incorrect and Fixed (IF) class into two parts, one for characters that were erased and correct (IF_c), and one for characters that were erased and erroneous (IF_e) [202]. It should be clear by now that the T-sequence paradigm can produce the traditional error rate metrics from the IS paradigm. It should also be clear that the T-sequence paradigm can handle text input behaviors that occur outside the IS paradigm, e.g., insertions, deletions, and substitutions within the transcribed text, the simultaneous entry or removal of multiple characters, and more.

Going further, the T-sequence paradigm can produce more than just the traditional error rates; it also gives rise to new metrics not formerly obtainable from the IS paradigm.

The new metrics that follow pertain to text entry transcription tasks with a presented string P and transcribed string T, just like for the traditional error rate metrics. Note that |T| indicates the length of the final transcribed string. I begin with some basic count metrics, upon which I build.

8.5.1 Basic Count Metrics

Total Changed Characters (TCC) refers to the number of characters that change during the text entry process, including all characters that are inserted or deleted.

$$TCC = |T| + 2{\times}IF$$

IF is added twice because any deleted characters were first inserted, constituting two changes per IF character.

Action Count (AC) refers to the number of actions taken during the text entry process. As described above, these actions are inferred from a T-sequence using INFER-ACTION. I also define counts for specific actions: the Insertion Action Count (IAC), Deletion Action Count (DAC), and Substitution Action Count (SAC).

More specifically, Correction Action Count (CAC) gives the number of corrective actions, which are delete and substitute actions (i.e., CAC = DAC + SAC). Similarly, Entry Action Count (EAC) gives the number of insert and substitute actions (i.e., EAC = IAC + SAC), as both make new entries.

8.5.2 New Metrics

Characters per Action (CPA) shows the average number of characters changed per action:

$$CPA = TCC/AC$$

Characters per Correction (CPC) and Characters per Entry (CPE) convey how many characters are changed, on average, per correction or entry, respectively:

$$CPC = IF/CAC$$

$$CPE = (|T| + IF)/EAC$$

Action Efficiency (AE) conveys the number of characters one action can change in a given time period, e.g., per second. It is therefore the "text-changing speed" of actions.

$$AE = TCC/Total\ time$$

More specifically, Correction Efficiency (CE) and Entry Efficiency (EE) indicate the "text-correcting speed" of actions and the "text-entering speed" of actions, respectively. Correction time refers to the total time of delete and substitute actions, and Entry time refers to the total time of insert and substitute actions.

$$CE = IF/Correction\ time$$

$$EE = (|T| + IF)/Entry\ time$$

Using the above metrics, I can categorize text entry methods into four types based on the effort to enter and correct text: (i) Easy entry, easy correction; (ii) Easy entry, hard correction; (iii) Hard entry, easy correction; and (iv) Hard entry, hard correction. Entry and correction difficulty is indicated by how much text can be added or deleted in one action, and by how fast it is to add or delete text. Therefore, the above CPE and EE metrics together are entry difficulty metrics; the CPC and CE metrics together are correction difficulty metrics. Following those, CPA and AE are overall difficulty metrics.

8.6 The New Evaluation Testbed Texttest++

The original TextTest tool [247] has been used to conduct numerous text entry studies (e.g., [117, 162, 240]) and produce measures based on the input stream (IS) paradigm of Soukoreff & MacKenzie [200, 201]. Inspired by TextTest, a Windows-based desktop application, I implemented TextTest++, a web-based testbed capable of running, logging, and analyzing text entry studies (Figure 8.2). By being web-based, TextTest++ is platform-independent, capable of being utilized on any device that offers a web browser. TextTest++ computes traditional metrics, including words per minute (WPM) and the error rates in Eqs. 8.1. It contains the algorithms described in this work, and produces all of the new metrics.

Figure 8.2 shows the main user interface for TextTest++. The program is written in JavaScript and logs each test in JSON format, which contains all of the T-sequences and inferred actions. A CSV file is also generated containing all traditional and new metrics described in this work.

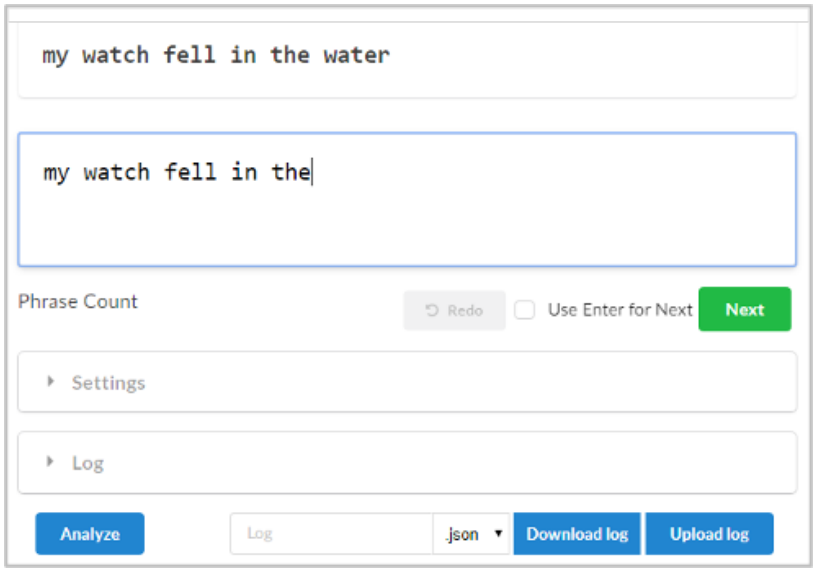

Figure 8.2: TextTest++ is a new web-based text entry evaluation testbed that produces the traditional metrics from the IS paradigm and the new metrics from the T-sequence paradigm.

8.7 Exercising Our Alogorithms, Metrics and Tool

I conducted an experiment to evaluate our algorithms, metrics, and the TextTest++ testbed. To put these through their paces, I first tested three keyboard-based text entry methods: a laptop keyboard, an on-screen accessibility keyboard, and a smartphone keyboard. Our questions were:

- Will the T-sequence paradigm yield the same results as the IS paradigm for transcriptions where the latter's experimental constraints happen to be upheld?
- How well does the T-sequence paradigm handle modern text entry behaviors that might arise? How often do such behaviors arise?
- What insights can our new metrics from the T-sequence paradigm provide?

8.7.1 Participants

I recruited 18 participants (15 female, 3 male) from our local university to partake in our study. Recruiting was done via flyers, emails, word-of-mouth, snowball sampling, and convenience sampling. Participants' ages ranged from 21 – 27. All participants were right-handed. They all indicated many years of experience typing on laptop keyboards and texting on smartphones. Participants were compensated $10 USD for 30 minutes.

8.7.2 Apparatus

I compared three keyboards: a laptop keyboard ("Laptop"), an on-screen desktop accessibility keyboard ("On-Screen"), and a smartphone touch keyboard ("Phone").

The Laptop keyboard was a Microsoft Surface Pro 4 typecover[4] measuring 11.60" × 8.54" × 0.20". Typing on such a keyboard is usually done serially, making it suitable for analysis with the IS paradigm—provided no arrow keys, no mouse, no copy/paste, no undo, and no word predictions are used.[5] Unlike in prior studies based on the IS paradigm, participants were free to employ such features.

The On-Screen desktop accessibility keyboard was operated using a Microsoft Wireless Mobile Mouse 4000 and the built-in Windows 10 On-Screen Keyboard. The keyboard measured 6.00" × 1.75". I chose this keyboard because by entering all text with the mouse, participants might use the mouse for other text entry behaviors, e.g., to reposition the text cursor, or to drag-select over multiple characters. The keyboard did not have word prediction enabled.

The Phone touch keyboard ran on a Google Pixel smartphone measuring 2.74" × 5.66" × 0.33". The keyboard was SwiftKey[6], a third-party smartphone keyboard equipped with opt-out auto-correction, word completion, a word prediction list, and, of course, the ability to reposition the text cursor with a tap or drag on the screen. The size of the SwiftKey

[4] `https://bit.ly/2LQCC7Q`

[5] That's a lot of restrictions! Alleviating all of them (and more) is the precise benefit achieved by moving from the input stream model to the transcription sequence model.

[6] https://swiftkey.com/en/keyboard/android/

keyboard itself was 2.74" × 2.60".

Our new TextTest++ tool was used throughout the experiment (Figure 8.2). The textbox in TextTest++ did not offer spell-check underlining. Timing for each phrase was from the first entered character to the last [136].

8.7.3 Procedure

For each participant, I randomly selected 30 phrases from a published text entry phrase set [140]. The ordering of phrases was randomized for each participant and method. Participants used each method in a fully counterbalanced order. At the start of using each text entry method, participants were given five warm-up phrases not included in the 30, which were the same for all participants and methods. Before the test phrases began, participants were told to proceed "as naturally as you can, while at the same time, keeping up your speed and accuracy." They were allowed to use the mouse in the Laptop condition, but not allowed to use the physical keyboard in the On-Screen keyboard condition. In the Phone condition, participants were told to use two-thumb typing. Participants were given 5 minutes of rest between each text entry method.

8.7.4 Design & Analysis

This experiment was a single-factor within-subjects design with three levels of text entry method: Laptop, On-Screen, and Phone. These levels were fully counterbalanced with 3! = 6 orders spread over 18 participants. Therefore, in all, I collected 18 participants × 3 methods × 30 phrases = 1620 text entry phrases.

Unlike most text entry experiments in which comparing the text entry methods is of primary interest, in this study, the methods were simply vehicles by which I could put our algorithms, metrics, and testbed through their paces. Therefore, I favor descriptive statistics over inferential statistics. (Indeed, I expect statistically significant differences among our text entry methods, but focusing on those differences would distract from our purposes here.)

8.8 Results

In this section, I report the results of our study, validating our work's ability to produce traditional speed and error rate metrics, while going further to produce our new metrics. Please refer to Table 8.1 for the numeric results.

Table 8.1: Means and standard deviations for speed, error rates, IFc and IFe counts, and our new metrics arising in the T-sequence paradigm. CPA, CPC, and CPE are counts. AE, CE, and EE are counts per second.

Input Method	Words per Minute (WPM)	Uncorrected Error Rate (UER)	Corrected Error Rate (CER)	Total Error Rate (TER)	Incorrected and Fixed - Correct (IFc)	Incorrect and Fixed - Errors (IFe)
Laptop	65.35 ±28.19	0.003 ±0.003	0.046 ±0.031	0.049 ±0.305	1.17 ±0.83	0.39 ±0.41
On-Screen	17.85 ±2.22	0.004 ±0.003	0.033 ±0.015	0.037 ±0.016	0.33 ±0.19	0.73 ±0.36
Phone	33.71 ±7.92	0.013 ±0.008	0.388 ±0.059	0.401 ±0.056	16.38 ±3.25	3.34 ±2.78

Input Method	Characters per Action (CPA)	Characters per Correction (CPC)	Characters per Entry (CPE)	Action Efficiency (AE)	Correction Efficiency (CE)	Entry Efficiency (EE)
Laptop	1.00 ±0	1.003 ±0	1.00 ±0	5.89 ±2.47	2.98 ±0.91	6.11 ±2.60
On-Screen	1.00 ±0	1.00 ±0	1.00 ±0	1.59 ±0.20	0.90 ±0.14	1.63 ±0.21
Phone	1.89 ±0.07	2.96 ±0.44	1.70 ±0.14	6.55 ±2.08	6.53 ±4.19	7.22 ±1.80

8.8.1 Words per Minute

I calculate words per minute (WPM) by subtracting the first character timestamp from the last character timestamp, being careful to define the length of the final transcribed string as one less than its character count (i.e., |T| – 1) [136].

8.8.2 Uncorrected, Corrected and Total Error Rates

All uncorrected error rates (UER) were below 2%, indicating high accuracy of the final transcribed strings. Corrected error rates (CER) were clearly highest for the Phone, which was the most error-prone during entry.

8.8.3 Comparison to the Input Stream Paradigm

To compare our T-sequence paradigm to the IS paradigm, I extracted 50 random trials from the Laptop condition that happened to exhibit the strictly sequential editing process required by the IS paradigm. To find these compliant trials, I looked through T-sequences to find where only the last character in each step of the sequence was modified. This requirement was met by 98% of Laptop trials, 94% of On-Screen keyboard trials, but only 2% of Phone trials.

By translating TextTest++'s JSON files into the XML log file format required by the original TextTest program [247], I could use the log file analysis feature in the latter to produce the C, INF, and IF character classes. The counts produced by TextTest were identical to those produced by TextTest++, showing that TextTest++ and the T-sequence paradigm can subsume the IS paradigm correctly.

8.8.4 Separating IF into IF_c and IF_e

Recall that I also want to separate Incorrect and Fixed (IF) characters, i.e., all erased characters, into IF_c and IF_e—those that were initially correct and initially erroneous, respectively [202]. Interestingly, for the Laptop and Phone methods, $IF_c > IF_e$, indicating that more initially correct characters were erased than erroneous ones. Such a result is consistent with observations of "pathologic error correction" [202], and also follows the use of the auto-correction feature on the Phone.

8.9 Validating the Correctness of INFER-ACTION

Recall that I logged text cursor position changes to obtain ground truth actions for each text entry method. I can compare these ground truth actions to those inferred from our INFER-ACTION algorithm to see how well our algorithm performed (i.e., when comparing T_{i-1} to T_i). Over our study's entire 1620 phrases, I found no differences between the results of INFER-ACTION and the ground truth information gleaned from text cursor position changes.

Furthermore, to ensure that INFER-ACTION correctly handles a variety of text entry methods, I conducted a follow-on study of three more text entry methods, none of which were keyboard-based: Dasher [234], gesture typing [121,267], and T9. Dasher is a pointing-based continuous-motion zooming interface. Gesture-typing enables stroke-gestures with a finger or stylus atop a virtual keyboard, with gestures corresponding to entire words based on their shapes and the letter arrangements beneath them. T9 is a predictive text method for 12-key numeric keypads, where sequences of key-presses are progressively disambiguated to form the most likely words.

For Dasher, I used Dasher 5.0 with cursor speed set at 3.2. For gesture typing, I used SwiftKey in swiping mode. For T9, I used Smart Keyboard Pro. Dasher ran on the Microsoft Surface Pro laptop; the other two methods ran on the Google Pixel smartphone. Under the same configuration as the main experiment, 6 participants (2 female, 4 male, ages 23 – 26) each transcribed text for 10 minutes with each method. The condition order was fully counterbalanced. Before the formal study, each participant learned about each method and took 20 minutes total to practice. Participants were paid $15 USD.

I collected 58 phrases with Dasher, 202 phrases with gesture typing, and 111 phrases with T9. Among the total 371 phrases, results generated from INFER-ACTION and results using ground truth cursor movement were exactly the same. This result indicates that INFER-ACTION works well across quite different text entry methods, including those that enter entire words at once.

8.10 Discussion

The experiment produced several findings. As we suspected, the different text entry methods resulted in different user behaviors. For example, with the Laptop keyboard, BACKSPACE was almost exclusively used for error correction, but with the On-Screen keyboard, mouse-based error correction with cursor repositioning was often used, a behavior formerly prohibited by the input stream (IS) paradigm. Even further, one participant (P6) used copy/paste in two phrases in the On-Screen keyboard condition. When P6 typed the phrase “the dreamers of dreams,” she drag-selected the first “dream”, copied it, and then pasted it at end of

the phrase, finally typing an "s". This behavior, too, was prohibited in the IS paradigm, but now can be supported.

Examining Characters per Action (CPA) and Action Efficiency (AE) together creates a picture of input difficulty. The Laptop and On-Screen keyboards had CPAs of 1.000 and 1.002, respectively, but the Laptop's AE is much higher than that of the mouse-driven On-Screen keyboard: 5.894 vs. 1.586 actions per second. This difference explains the faster entry speed of the Laptop keyboard. The Phone had the largest CPA and AE, indicating that it is even easier to act upon text than with the Laptop keyboard. This ease arises, for example, when one types only the initial characters of a word, and then word completion finishes the rest.

I now turn to entry and correction difficulty. Characters per Entry (CPE) and Characters per Correction (CPC) are nearly 1.00 for the Laptop and On-Screen keyboards. But interestingly, Entry Efficiency (EE) is about twice as high as Correction Efficiency (CE) for both keyboards, indicating that deleting a character takes about twice as long as entering it in the first place.

Of note is that participants deleted more correct characters than erroneous ones with the Laptop keyboard ($IF_c > IF_e$). With auto-correct active on the Phone keyboard, it is no surprise that this occurred, but that it occurred on the Laptop keyboard supports informal observations of "pathologic error correction" from prior work [20]. The EE of the Laptop keyboard was quite high (6.111), so perhaps participants did not mind if they had to re-type correct characters after backspacing through them to reach an incorrect one. The situation was different with the On-Screen keyboard, whose EE was only 1.629, and whose $IF_c < IF_e$ (0.33 vs. 0.73).

8.10.1 Limitations

Indeed, the T-sequence model has limitations: (1) Because T-sequences focus on providing general metrics across different text entry methods, gaining insights about how a text input method does its work is not possible with the model, just as it was not possible with the

IS model. Evaluators wishing to examine particular features of a text entry method must still build a custom evaluation tool. (2) Some text entry methods such as T9 produce temporary characters during input, with those temporary characters often appearing in-place. If such characters are actually committed into the textbox, then additional characters will be counted in IF. This problem, however, does not arise if these temporary characters are not actually committed until they are resolved; for example, temporary characters might appear in a separate list from which the user makes a selection. (3) The metrics associated with the T-sequence model are still character-level metrics, even though the model itself and its associated actions accommodate word-level behaviors such as those used in gesture typing. For word- or phrase-level input methods such as voice typing or gesture typing, INFER-ACTION still produces correct character-level metrics, as was validated in the second study, but the relevance of these metrics to word-level methods might be less. (4) In the experiment, INFER-ACTION worked flawlessly, but the text entry behaviors of participants in real life might differ from participants' behaviors in a lab-based transcription study. And we know INFER-ACTION is not perfect. (Recall the "rome" and "Rome" example, above).

8.11 Conclusion

Although the input stream (IS) paradigm of text entry evaluation has been highly successful, modern text entry methods require more flexible evaluation paradigms. To achieve this, we presented a method-independent paradigm based on transcription sequences, or T-sequences, which contain transcription snapshots after every text-changing action occurs. I built an abstraction of actions and showed how to infer these actions from transcription string pairs. I also showed how the traditional character classes used in error rate measurement [201, 202] can be calculated. Furthermore, we presented new metrics arising from this paradigm. The study demonstrated that T-sequences can supersede the IS paradigm and offer new insights not formerly possible, while greatly lessening the constraints on the evaluation and supporting many modern text entry behaviors. I offer the new method-independent web-based tool, TextTest++, to researchers and practitioners in the hope that text entry evaluations will be made easier, more flexible, realistic, and informative.

Chapter 9

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) (Crossman 1957, Welford 1968), but to date, no similar metric has been established for text entry. In this paper, I introduce a text entry method-independent throughput metric based on Shannon information theory (1948). To explore the practical usability of the metric, I conducted an experiment in which 16 participants typed with a laptop keyboard using different cognitive sets, i.e., speed-accuracy biases. The results show that as a performance metric, text entry throughput remains relatively stable under different speed-accuracy conditions. I also evaluated a smartphone keyboard with 12 participants, finding that throughput varied least compared to other text entry metrics. This work allows researchers to characterize text entry performance with a single unified measure of input efficiency.[1]

9.1 Introduction

Since the earliest days of interactive computing, text entry has remained one of the most important and frequent of all computing activities performed by humans. Alongside pointing, text entry is fundamental to interactive computing systems—almost all platforms provide it [139], and immense energy has been poured into creating advanced methods (e.g., [267]),

[1]This chapter is adpated from: Mingrui "Ray" Zhang, Shumin Zhai, Jacob O. Wobbrock. (2019). Text entry throughput: Towards unifying speed and accuracy in a single performance metric. Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI 19). New York: ACM Press.

calculating performance bounds (e.g., [203]), improving evaluation methods (e.g., [201]), and providing evaluation tools (e.g., [10,167,247]). When assessing the performance of text entry methods, however, a fundamental challenge remains: As with all human performance, speed and accuracy trade off against each other, complicating the assessment of text entry methods in the presence of such tradeoffs. How can I draw firm performance conclusions in the presence of such tradeoffs? Table 9.1 illustrates the problem:

Table 9.1: Two hypothetical text entry methods posing a speed-accuracy tradeoff. Which has better overall performance? Throughput gives a way of equitably comparing methods, even across studies.

Method	**Speed (WPM)**	**Error Rate (%)**
A	13.2	4.8
B	16.7	7.2
Difference (B-A)	*3.5 (B is better)*	*2.4 (B is worse)*

Modern text entry evaluations (see, e.g., [139,201,247]) allow for the separate measurement of speed and accuracy, usually reported, respectively, as words per minute (WPM) [136] and three error rates: uncorrected, corrected, and total errors [201]. Strings are presented for transcription and participants are instructed to "proceed quickly and accurately" [201, 242, 259]. Uncorrected errors are those that remain in the transcribed string, and are therefore at odds with speed. Corrected errors are those made but fixed during entry (e.g., backspaced)—as such, they take time, and are therefore subsumed in the speed measure. The question, then, is how to reconcile speed with uncorrected errors.

The instruction to "proceed quickly and accurately" is an interesting, and concerning, one [266]. Every human actor has his or her 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 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.

A text input system is, quite literally and conceptually, a communications channel in the information-theoretic sense. Shannon's information theory [191] 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.

I propose that text entry evaluations are describable in terms of Shannon's model. 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 throughput based on text entry speed and the uncorrected error rate. The experiment showed that for the same person with the same text entry method, the throughput measure exhibited less variation compared to other text entry metrics 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 contributions of this work are: (1) The formalization of text entry transcription tasks as Shannon information transmission tasks; (2) A new throughput metric unifying speed and accuracy in text entry, which enables comparisons of overall text entry efficiency; and (3) Empirical results validating the stability of this new throughput metric across different speed-accuracy biases, thereby characterizing a text entry method's communication efficiency. Although I do not seek to replace speed and accuracy as text input performance metrics, and I encourage them to be reported in all future studies, I believe that the new throughput metric can provide a valuable unified measure for characterizing the overall efficiency of text input methods.

9.2 Desired Properties of a Throughput Metric for Text Entry

Below, I convey the view of the desired properties of an ideal text entry throughput metric. These properties generally hold for other text entry metrics and for throughput in Fitts' law [137]:

1. Method independence. The metric should be text entry method-independent. As with metrics like words per minute [136] and uncorrected error rate [201], the throughput metric should not require method-specific knowledge to be calculated. In other words, throughput should apply to any text entry method.

2. Use of P, T, and time only. Following method-independence, the throughput metric should be computed only with knowledge of the presented string (P), transcribed string (T), and total entry time. Everything that happens during the entry process, such as corrections, should be unrelated to the metric.

3. Performance isolation. The calculation of the metric should only depend on the performance of the system (human and input method), like existing speed [136] and accuracy [201] metrics. It should not include external factors such as which phrase set is used in an evaluation. Although the test phrases do influence the metric, they should represent the intended application domain. Otherwise, an experimenter could "cheat" by selecting a phrase set favorable to the method.

4. Speed-accuracy invariance. Within reasonable limits, the metric should be relatively stable across participants' different subjective speed-accuracy biases, thereby characterizing the communications channel, not a particular participant bias.

5. Compliance with established text entry evaluation procedures. Evaluation procedures should not be artificially constrained to enable the use of the new throughput metric. Researchers should be able to calculate the metric without making participants enter text in a specific way, e.g., by forcing them to enter every character correctly.

6. Simplicity of measurement. Ideally, the metric should be easy to measure. By "easy," I mean both the data collection and calculation steps should be as straightforward as possible,

similar to calculating existing text entry error rates [201].

9.3 Mapping Shannon Information Theory to Text Entry

Before defining throughput for text entry, I must establish how Shannon's information-theoretic communications model [191] maps to the text entry transcription process. In Shannon's original model of a communications system (Figure 9.1), a message is produced by an information source, transmitted by a transmitter through a channel, received by a receiver, and recorded at a destination. The message can be perturbed by noise, meaning the message sent is not necessarily the message received.

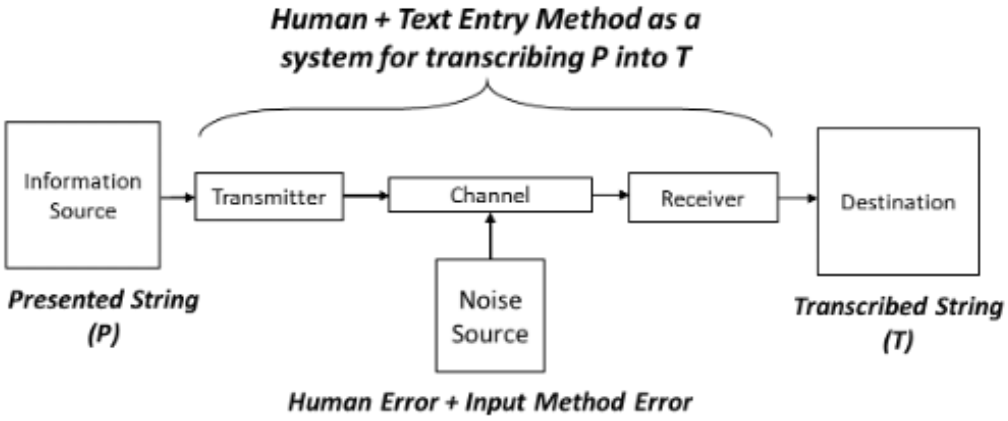

Figure 9.1: Shannon's information transmission model, with labels in bold mapping the model to the text entry transcription process. In a text entry transcription task, after the presented string (P) is first shown, the participant enters the transcribed string (T) using whatever text entry method is under investigation. Usually, a generic test-bed application is used, which presents phrases from a representative corpus and provides for the transcribed text to be entered directly below, as shown in next figure.

The presented string (P) can be viewed as the information source, and the transcribed string (T) as the destination in Shannon's model. The human-and-text-entry-method serves as the overall transmission channel in Figure 9.2. If T is different from P, the message is considered to be perturbed by noise, which is the "error" in the channel. Throughput is

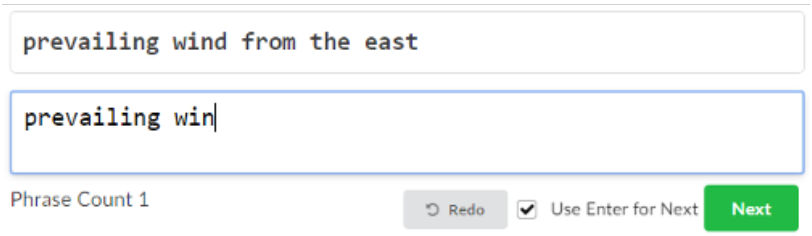

Figure 9.2: A text entry transcription task in the TextTest++ evaluation tool presented in this work.

thus the information transmission rate of the system.

9.3.1 A Discrete Channel With Noise

In the text entry transcription process, the human-and-text-entry-method transmits a message through a noisy discrete channel: characters are discrete, and errors appear because of noise. Shannon [191] (pp. 407-410) gave a concrete example of the transmission rate of such a channel. In his example, there are two possible symbols, 0 and 1, and they each are produced by the information source with probabilities p_0=p_1= 0.5. The information source transmits the symbols at a rate of 1000 symbols per second. During transmission, noise introduces errors. On average, say, 1 in 100 symbols is received incorrectly, e.g., a 0 is received as a 1, or a 1 is received as a 0. What is the information transmission rate of the channel?

A straightforward answer might seemingly be 990 bits/s, representing the transmission rate of correct symbols. However, because the receiver has no knowledge about where the errors occur, this is not the correct answer. (If the receiver knew the location of errors, it would know everything about the source, which would mean it has the same information as the source. But this is not the case.)

Transmission rate is the transmitted information per unit time. Suppose X refers to the source and Y refers to the receiver. Shannon defines the "mutual information" $I(X, Y)$ as

the transmitted information:

$$I(X,Y) = H(X) - H_Y(X) \tag{9.1}$$

$H(X)$ represents the information, or "entropy," of the source, and $H_Y(X)$ represents the conditional entropy, which is called "equivocation." It refers to the information of X that could not be gained from Y, the information lost during transmission. More detail can be found in Shannon's original work [191].

The following equations define entropy and equivocation, respectively:

$$H(X) = -\sum_i p(i) log_2 p(i) \tag{9.2}$$

$$H_Y(X) = -\sum_{i,j} p(i,j) log_2 p(i) \tag{9.3}$$

In Eqs. 9.1 and 9.3, i is each symbol in source X, and j is each symbol in receiver Y. The term $p(i)$ is the probability of symbol i being produced by the information source. The term $p(i,j)$ is the probability that the source produces symbol i and symbol j is concurrently received. In the following text, I use i to represent the character produced by the source, and j to represent the character received by the receiver. Moreover, I use $p_a(b)$ to represent the probability of event b given that the event a happened. Thus, the term $p_j(i)$ is the probability that the source produces symbol i given symbol j is received, and $p_i(j)$ is the probability that symbol j is received given symbol i is produced.

In Shannon's example, then, $H(X) = -(0.5 log_2 0.5 + 0.5 log_2 0.5) = 1.00$ bit per symbol. Furthermore, $p(0,0) = p(1,1) = 0.495, p(0,1) = p(1,0) = 0.005$. So $H_Y(X) = -2\times(0.495 log_2 0.99 + 0.005 log_2 0.01) = 0.081$ bits per symbol. Thus, $I(X,Y) = 0.919$ bits per symbol, and the transmission rate is therefore 919 bits/s.

The transmission rate in a text entry transcription process could be calculated as in the example above, if, for now, I were to assume that there were only substitution errors [247] in the text entry process (Figure 9.3). However, in text entry transcription tasks, characters might not only be substituted, but also omitted or inserted. For example, when attempting

to type “abc”, one might type “ac” or “abxc”. How should I deal with omission and insertion errors [247], which do not have a direct analogue in Shannon’s model? I therefore must extend the model.

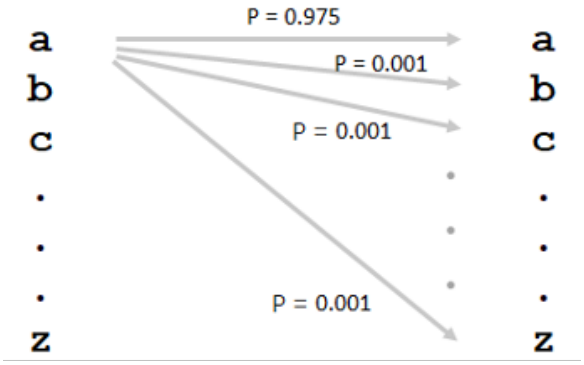

Figure 9.3: A character transmission probability graph with only substitution errors shown. The presented letter is on the left and the transcribed letter is on the right, with given probabilities.

9.3.2 The Null Character (∅)

The challenge of handling omission and insertion errors from an information-theoretic point of view is the lack of a character on either the sending or receiving side of the channel. For omissions, characters are sent by the source but never received. For insertions, characters are never sent by the source but somehow received.

To address this challenge, I extend Shannon’s substitution-only model with a null character. (I use “∅” as the notation for the null character in the remainder of this work .) Using the null character, omission errors occur when a sent character becomes ∅, and insertion errors are when ∅becomes a received character (Figure 9.4).

9.4 Calculation of Throughput

Having established how the Shannon model of information transmission maps to text entry transcription tasks—and to substitution, omission, and insertion errors—we are now able

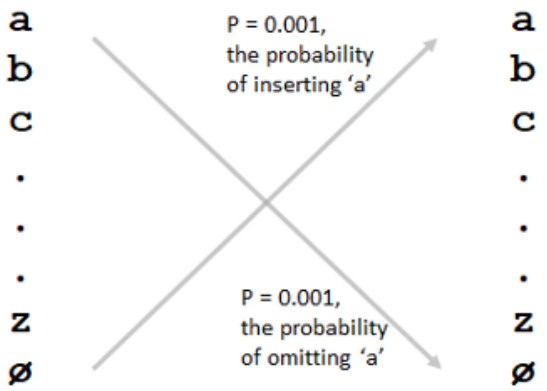

Figure 9.4: A character transmission probability graph with the null character "∅" for omission and insertion errors. Using this approach, I can calculate $p(\emptyset)$, $p(i, \emptyset)$, and $p(\emptyset, j)$ by counting the corresponding omission and insertion errors, just as for other symbols within the Shannon model. (Note that other related terms, like $p_i(\emptyset)$, can also be calculated using these three terms.)

to offer detailed steps for calculating text entry throughput:

Step 1. Calculate the source information. To calculate the source information $H(X)$, one must get the character distribution of the source according to Eq. 9.2. In text entry transcription tasks, that means the character distribution $p(i)$ of the presented strings, which could be known by counting the occurrences of each presented character. However, this approach requires large phrase sets to give representative estimates for each character. Furthermore, with the null character in the calculation, there will be a $p(\emptyset)$ in the source distribution. As null is just an imaginary character, including $p(\emptyset)$ would actually lead to different H(X) based on the observed number of omissions and insertions made. The null character should not change the source information, as it is used only for the calculation of the transmission probability.

To address this challenge of computing $H(X)$, I offer a practical solution: fixing the source information on a larger scale—the language level. For the English language, I use character-level information because throughput is a character-level information metric, one based on

symbols in Shannon's original formulation[2]. I use existing probabilities for 26 lowercase letters ('a'-'z') plus SPACE[3]. I normalize probabilities such that $\sum_{i=1}^{27} p(i) = 1.00$ (the exact probabilities for each letter are in Appendix D).

Step 2. Calculate transmission probabilities. To get the equivocation $H_Y(X)$, I need to get $p(i,j)$ and $p_j(i)$ for each character i in the information source (presented strings P), and each character j in the destination (transcribed strings T) according to Eq. 9.3. Furthermore, I need to accommodate the probability of $\emptyset$in this step as well. The new probability distribution of the 27 characters $p'(i)$ that accommodates the null character is $p'(i) = p(i)(1 - p(\emptyset))$. The term p($\emptyset$) reflects the number of insertion errors over all transmission instances:

$$p(\emptyset) = p'(\emptyset) = \frac{\sum_j N(\emptyset \rightarrow j)}{\sum_{i,j} N(i \rightarrow j)} \tag{9.4}$$

Here, $N(a)$ refers to the number, or count, of event a. N(a→b) refers to the count of character a becoming b. If I have the transmission probability of character i turning into character j, i.e., $p_i(j)$, then both terms can be calculated as follows. Note that Eq. 9.6 is an application of Bayes' theorem:

$$p(i,j) = p'(i)p_i(j) \tag{9.5}$$

$$p_j(i) = \frac{p(i,j)}{\sum_i p(i,j)} \tag{9.6}$$

However, it is not feasible to assume that every possible character i will be substituted by every other possible character j in a laboratory text entry evaluation, given various character frequencies. For example, one might have to type hundreds of phrases to observe one occasion of 'z' being substituted for 'a'. First, some characters like 'z' make few appearances

[2]I consider English characters as the symbol set in this study. It could also be the units in other languages as well.

[3]`http://www.macfreek.nl/memory/Letter_Distribution`

in most phrase sets, leading to few transmission instances of the character. Second, the specific attributes of the method under investigation (for example, the keyboard layout) might lead to few or no instances of substituting character j for character i. These improbabilities mean I cannot measure equivocation precisely unless I make the participant enter myriad phrases.

However, if I move a level above individual characters, the overall omission, insertion, and substitution error rate can be observed without too many phrases. Thus, for practical purposes, instead of seeking each $p(i, j)$, I compute the three overall rates, and treat them as equal for all characters as an approximation. The calculation of overall probabilities is shown below:

Overall insertion probability $p(I)$:

$$p(I) = \frac{\sum_{j \neq \emptyset} N(\emptyset \rightarrow j)}{\sum_{i,j} N(i \rightarrow j)} \tag{9.7}$$

Overall omission probability p(M):

$$p(M) = \frac{\sum_{i \neq \emptyset} N(i \rightarrow \emptyset)}{\sum_{i \neq \emptyset, j} N(i \rightarrow j)} \times (1 - p(I)) \tag{9.8}$$

Overall substitution probability p(S):

$$p(S) = \frac{\sum_{i \neq \emptyset, j \neq \emptyset, i \neq j} N(i \rightarrow j)}{\sum_{i \neq \emptyset, j} N(i \rightarrow j}) \times (1 - p(I)) \tag{9.9}$$

Overall probability of correct entries p(C)

$$p(C) = \frac{\sum_{i \neq \emptyset} N(i \rightarrow i)}{\sum_{i \neq \emptyset, j} N(i \rightarrow j)} \times (1 - p(I)) \tag{9.10}$$

An insertion error happens only when i=$\emptyset$and is the only error possible when i=$\emptyset$. Thus, there is a (1-p(I)) factor in Eqs. 9.8 9.9 9.10, as they only can happen when i$\neq\emptyset$.

To get $N(i \rightarrow j)$, i.e., how many times presented character i is transcribed as character j, I use work by MacKenzie & Soukoreff [138] to get the optimal alignments between a presented string P and a transcribed string T. When aligning P and T to calculate the minimum string

distance (MSD), multiple optimal alignments can exist. For example, there are two optimal alignments for P = "optimal" and T = "optiacl", each reflecting MSD = 2:

```
P1: optimal
T1: optiacl

P2: optima-l
T2: opti-acl
```

In the two alignment pairs above, substitution errors occur at character mismatches; omission errors occur where T has a '-'; and insertion errors occur where P has a '-'. By weighting these errors by the number of times they occur, I can calculate the substitution, omission, insertion and correct-entry probabilities. In the two pairs above, there are two substitutions, one insertion, one omission and eleven correct entries. As there are two alignments in total, the overall probability for substitutions is $[N('m' \rightarrow 'a') + N('a' \rightarrow 'c')]/2 = 1.0$; for insertions it is $[N(\emptyset \rightarrow 'c')]/2 = 0.5$; for omissions it is $[N('m' \rightarrow \emptyset)]/2 = 0.5$; for correct entries it is $11/2 = 5.5$.

Step 3. Calculate throughput. Once I have the overall error probabilities for the three error types and correct entries, I get the average probability for each character:

$$p_i(j) = \begin{cases} \frac{p(I)}{N(j)}, & i = \emptyset, j \neq \emptyset \\ \frac{p(M)}{N(j)} = p(M), & i \neq \emptyset, j = \emptyset \\ \frac{p(S)}{N(j)}, & (i \neq j) \neq \emptyset \\ \frac{p(C)}{N(j)} = p(C), & (i = j) \neq \emptyset \end{cases} \tag{9.11}$$

In Eq. 9.11, $N(j)$ is the number of different characters that can be received. For any non-null character i, its probability of omission for each j equals the overall omission probability, because j=$\emptyset$, so $N(j) = 1$. Its probability of correct-entry also equals the overall correct-entry probability, because $j = i$, $N(j) = 1$. In the experiment, there are 27 symbols ('a'-'z' and SPACE). Thus, for any non-null character i, its probability of substitution for each j

is $p(S)/26$, because there are 26 characters that are different from i. Finally, for $\emptyset$, the probability of insertion of each j is $p(I)/27$, because there are 27 non-null characters that can be inserted.

With Eqs. 9.5 9.6, all components for computing the mutual information $I(X,Y)$ can be had according to Eqs. 9.1, 9.2, 9.3, which is in units of bits per character. Multiplying I(X,Y) by entry speed, such as characters per second, gets text entry throughput, whose units is in bits per second (bits/s).

In review, to calculate text entry throughput: (1) use general English letter frequencies as the source information; (2) use $\emptyset$to handle omission and insertion errors, and align P and T to get the overall probabilities of each type of error; and (3) average the probabilities for each character in four categories: omissions, insertions, substitutions, and correct-entries. Then calculate throughput.

9.4.1 A Worked Example

To illustrate the calculation process, assume a person transcribes two phrases, P1 and P2, as T1 and T2, respectively:

```
P1: my watch fell in the water
T1: my wacch fell in water
P2: prevailing wind from the east
T2: previling wind on the east
```

For considerations of space, assume there is only one optimal alignment for each phrase pair:

```
P1: my watch fell in the water
T1: my wacch fell in ----water
P2: prevailing wind from the east
T2: prev-iling wind --on the east
```

In total, there are 7 omission errors, 0 insertion errors, 2 substitution errors, and 46 correct

entries. Thus $p(I)$ is 0.000, $p(M)$ is 0.127, $p(S)$ is 0.036, and $p(C)$ is 0.836. The value for $p(\emptyset)$ is 0 because there are no insertions.

The source information $H(X)$ is calculated according to English letter frequencies as 4.09 bits/character. After adding the null character $\emptyset$, I calculate $p'(i) = p(i)(1 - p(\emptyset))$; in this example, $p'(i) = p(i)$, as there are no insertion errors.

According to Eq. 9.11, for each $i \neq \emptyset$, the substitution probability of i becoming j is $p_i(j) = p(S)/N(j) = 0.036/26 = 0.0014$.

Omission and correct-entry probabilities are the same as their overall respective probabilities. Insertion probability is 0/27= 0.

Thus, I can calculate $p(i, j) = p'(i) \times p_i(j)$ for each pair of characters. And $p_j(i) = \frac{p(i,j)}{\sum_i p(i,j)}$. I thus calculate $H_Y(X)$ according to Eq. 9.3, which yields 0.852. The mutual information is $I(X, Y) = H(X) - H_Y(X) = 3.238$ bits/character. If the entry speed is four characters per second, throughput is $3.238 \times 4.000 = 12.952$ bits/s.

9.5 Study Method

I conducted an experiment to put the throughput calculation through its paces using two text entry methods: a laptop keyboard and a smartphone keyboard. I focused on three questions:

- Can I successfully manipulate different speed-accuracy biases for participants during text entry?
- How stable is the new throughput metric across different participant speed-accuracy biases?
- How does throughput stability compare to that of established speed and error rate metrics?

9.5.1 Participants

The study was a between-subjects study with a total of 27 participants. Of those, 15 participants (ages 22 – 27, 6 male, 10 female) used a laptop keyboard, and 12 other participants (ages 22 – 25, 6 male, 6 female) used a smartphone keyboard. Recruiting was conducted via email and word-of-mouth. One participant was left-handed; all others were right-handed. All participants indicated years of experience typing with laptop and smartphone keyboards. For the laptop keyboard, participants were compensated $15 USD plus a bonus for about 1.5 hours of their time. For the smartphone keyboard, participants were compensated $15 USD plus a bonus for about 1 hour of their time.

9.6 Apparatus

I compared two input methods: a laptop keyboard and a smartphone keyboard. The laptop keyboard was a Microsoft Surface Pro 4 typecover measuring 11.60" × 8.54" × 0.20". The smartphone keyboard ran on a Google Pixel measuring 2.74" × 5.66" × 0.33". Its keyboard was Gboard, a smartphone keyboard with advanced features including opt-out auto-correction, word completion, and a word prediction list. In the study, all advanced features were turned on for the Advanced condition, and turned off for the Plain condition. The size of smartphone keyboard was 2.74" × 2.60".

I ran the study with the new TextTest++ tool (see Figure 9.5). TextTest++ is a web-based text entry method-independent evaluation tool inspired by the popular Windows TextTest tool [247]. Timing for each phrase was from the first entered character to the last [136].

9.6.1 Procedure

There were five "cognitive sets" [68] that I manipulated in the laptop experiment: Extremely Accurate (*EA*), Accurate (A), Neutral (*N*), Fast (*F*), and Extremely Fast (*EF*). Each cognitive set represented a different speed-accuracy bias. For example, in *EA*, participants needed to type very carefully to avoid errors, while in *EF*, participants needed to type very fast, even if they might make many errors.

Prior research on the speed-accuracy tradeoff [92] shows that it is not enough to use verbal instructions to impose different cognitive sets. Instead, I designed a game-like text entry task. After participants transcribed a phrase, they received points based on their speed and accuracy relative to the stated objective for that phrase. More points meant a greater cash bonus at the study's end.

Figure 6a shows a text entry transcription trial with a presented string and the transcribed string below it. A 5-minute timer appeared on the left after 10 warm-up phrases were completed. When participants finished entering a phrase, if the total score increased according to the scoring criteria, a green indicator appeared to the right of the text box, accompanied by a "cha-ching" sound; if the score did not change, a black indicator appeared with a "flat" sound; if the score decreased, a red indicator appeared with a "losing" sound. The indicators are shown in Figure 9.5b.

Figure 9.5: (a) The TextTest++ interface. The timer is on the left; the condition selector and total score are on the right. After typing in the middle text area, participants could hit the ENTER key or "Next" button to go to the next phrase. (b) Indicators shown after finishing each phrase corresponding to the score increasing, no change, or decreasing, respectively.

Laptop Keyboard. In the laptop keyboard condition, each participant went through two identical blocks of trials. For each block, participants completed transcription trials under the five cognitive sets. For each cognitive set, participants had five minutes to transcribe

as many phrases as time allowed. I did not fully counterbalance the order of cognitive sets, as a cognitive set is a relative concept, and thus it was more intuitive for participants to conduct *EA* and *A* adjacently, and likewise *EF* and *F*. I also set *N*to be the last condition, as it was hard to type "neutrally" without having done the accurate and fast conditions first. There were in total 2 (order within accuracy-biased group, i.e., *EA*-*A*, *A*-*EA*) × 2 (order within speed-biased group, i.e., *EF*-*F*, *F*-*EF*) × 2 (order between accuracy- and speed-biased groups) = 8 orders in all.

To help participants understand the experiment design, they were taught the scoring mechanism (about which more detail will be given in the next section) before the test, accompanied by verbal instructions (Table 9.2). Before the study started, participants tried five phrases for each speed-accuracy condition, and tried a 5-minute test in the neutral condition ("*N*") to get familiar with the TextTest++ user interface and the reward mechanism.

To reduce learning, the phrases in each speed-accuracy condition were different. I mixed the phrase sets from MacKenzie & Soukoreff [140] and Vertanen & Kristensson [224]. I included all 500 phrases of the former source, and extracted 362 phrases without numbers and acronyms from the latter source. The average number of words per phrase was 5.78, and average number of characters per word was 4.97. After participants finished one speed-accuracy condition, the typed phrases were eliminated from subsequent conditions. After each block of trials covering all five speed-accuracy conditions, participants had a rest for five minutes, and then began the second block of trials.

Smartphone Keyboard. In the smartphone keyboard condition, there were two different blocks: one with the advanced features of auto-correction, word completion, and word prediction enabled (Advanced), and one with these advanced features disabled (Plain). For each block, there were three cognitive sets: Accurate, Neutral, and Fast. I still set Neutral as the last condition, resulting in 2 (order between accuracy-biased and speed-biased conditions, i.e., *A*-*F*, *F*-*A*) × 2 (order between two sessions, i.e., Advanced-Plain, Plain-Advanced) = 4 orders in all.

The basic procedure and phrase set were the same as in the laptop keyboard condition.

Table 9.2: Verbal instructions given to participants used in different speed-accuracy conditions.

	Verbal Instruction
EA	"Try to type very accurately and ensure that each key-press is correct."
A	"Try to type accurately. You can make corrections as long as the final string is correct."
N	"Try to type fast and accurately, as in your daily life. A few errors are acceptable."
F	"Try to type fast, and it's OK to make errors."
EF	"Try to type as physically fast as you can, still typing according to the text, but you can ignore errors."

Participants were told to hold the phone and type using a two-handed posture with two thumbs. I modified TextTest++ to display properly on a smartphone screen.

9.6.2 Bonus Mechanism

Figure 7, below, depicts the process for determining participants' bonuses:

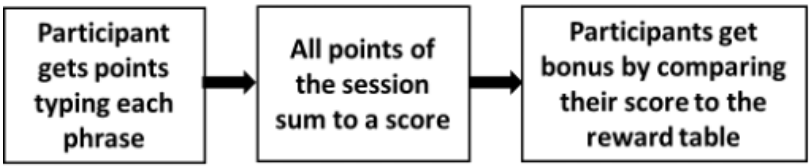

Figure 9.6: Procedure for calculating bonuses for each session.

The points awarded for each phrase in different speed-accuracy conditions are shown in Table 3. Participants received points for a transcribed phrase if they met the accuracy criterion in "Baseline Points." Failing that, they received no points for a phrase if they met the criterion in "No Points." Failing that, they actually lost points according to "Lose Points."

Table 9.3: Points awarded for each phrase in five cognitive sets based on accuracy criterion. The first item in each cell is the points, followed by the criterion to gain or lose those points. E refers to the number of uncorrected errors.

	Baseline Points	**No Points**	**Lose Points**
EA	+10; E = 0 and no corrections	E = 0 but corrections >0	-5; E >0
A	+9; E = 0	0 <E <5	-4; E≥5
N	+8; E <5	5 ≤E <10	-3; E ≥10
F	+7; E <10	10 ≤E <15	-2; E ≥15
EF	+6; E <15	E ≥15	—

In Table 3, E refers to the number of uncorrected errors, i.e., the minimum string distance (MSD) [201] between the presented string P and the transcribed string T. As each phrase consisted of a similar number of characters, I measured the error in characters rather than error percentage. Note that for the *EA* condition, along with leaving no errors, participants had to make no error corrections in the typing process to gain the baseline points.

As another example, consider the Accurate condition (A). If there are no uncorrected errors in the transcribed string, the participant will get 9 points. In the Neutral condition (N), a participant would get zero points if there are 5 errors in the transcribed string, and would lose 3 points if there are 10 or more uncorrected errors.

The allocated testing time for each condition was fixed; thus, in order to get as high a score as possible, participants had to type as many phrases as possible, while still meeting the accuracy criterion for points. This scheme ensured that participants would strive for their best performance in each condition, rather than type unnecessarily slowly to meet the accuracy criterion for each phrase.

The bonus table for the laptop keyboard is given in Table 9.6.2, which shows expected minimum speeds (WPM_e) and the total required score (S_r; see Eq. 9.12) for each condition.

(For the smartphone keyboard, I halved the laptop speeds in the corresponding *A/N/F* conditions.)

Table 9.4: Bonuses based on the expected speeds (WPM_e) and required scores (S_r) (Eq. 9.12) for the five speed-accuracy conditions. Each cell contains two values, the minimum expected typing speed (WPM_e) and the required score (S_r) to get the corresponding bonus. WPM_e is used to calculate S_r.

	WPM_e | S_r thresholds for given bonuses				
EA	15 | 125	35 | 425	65 | 875	105 | 1475	145 | 2075
A	20 | 170	40 | 440	70 | 845	110 | 1385	150 | 1925
N	25 | 200	45 | 440	75 | 800	115 | 1280	155 | 1760
F	30 | 215	50 | 425	80 | 740	120 | 1160	160 | 1580
EF	35 | 215	55 | 395	85 | 665	125 | 1025	165 | 1385
$$$	**$1**	**$2**	**$3**	**$4**	**$5**

For each speed-accuracy condition, a participant's total points are summed up, and compared to the calculation of the required score (Sr) for that condition, as follows:

$$S_r = \frac{WPM_e \times TM \times BP}{AWP} \times 0.75 - 100 \tag{9.12}$$

In Eq. 9.12, *TM* is total minutes, *BP* is baseline points from Table 9.6.2 (thus ranging from +6 – +10), and *AWP* is average words per phrase. Based on the phrase set and experiment procedure, I set *TM* to 10 and *AWP* to 5. The reason for the term "× 0.75 – 100" is based on the pilot study to make it easier to get bonuses.

In general, then, participants received an $N reward if their total points was above the required score (S_r) shown in Table 9.6.2. For example, the score in the Neutral (*N*) condition for a reward of $3 is 800; for $4, it is 1280. Thus, if one reached 895 points in condition *N*, one would get a $3 bonus for that condition.

Although the payoff scheme might seem a bit complicated, it was straightforward for participants: from *EA* to *EF*, they got fewer points for each phrase, but also a smaller error

penalty with more room to make errors. I ensured all participants understood *EA*-*EF* before the experiment, having them demonstrate their typing strategy in each condition. Each participant was supposed to get at least a $1 bonus for each speed-accuracy condition; thus, I made the corresponding typing speeds slow. In contrast, I made the speeds for a $5 bonus high so that this level would not be exceeded by even an expert typist. As a result, all participants had reason to try hard to gain points.

9.7 Results

I compared four metrics: speed (WPM), uncorrected error rate, adjusted speed (AdjWPM), and throughput in both keyboard conditions. Throughput exhibited the least variance across different speed-accuracy conditions, and displayed a mild bell-shaped pattern from Extremely Accurate (*EA*) to Extremely Fast (*EF*). Prior studies of throughput in Fitts' law also show throughput to be reasonably stable across different speed-accuracy biases, but never invariant (see, e.g., [68, 137, 265]).

9.7.1 Laptop Keyboard Condition

In all, 7342 phrases were collected in the laptop keyboard condition. There were 2101 phrases from warm-up and practice sessions, which were not included in the analysis. The average uncorrected error rate, speed, AdjWPM, and throughput of the 16 participants are shown in Figure 9.7.

As Figure 9.7 shows, the reward mechanism worked well on manipulating different cognitive sets. From *EA* to *EF*, participants left more errors and typed faster. Errors and speed varied a lot from *EA* to *EF*, but throughput was quite stable across conditions.

I used coefficient of variation (*CV*) to measure the variance of groups of data that have different units. *CV* equals the ratio of the standard deviation to the mean: the smaller the *CV*, the less varied the data. I calculated the *CV* of the four metrics as: 1.05 for error rate; 0.13 for speed, 0.04 for AdjWPM and 0.03 for throughput. Thus, AdjWPM varied less than speed (raw WPM) but still more than throughput. Interestingly, the bell-shaped curve indicates that throughput was highest in the Neutral condition, and dropped when it

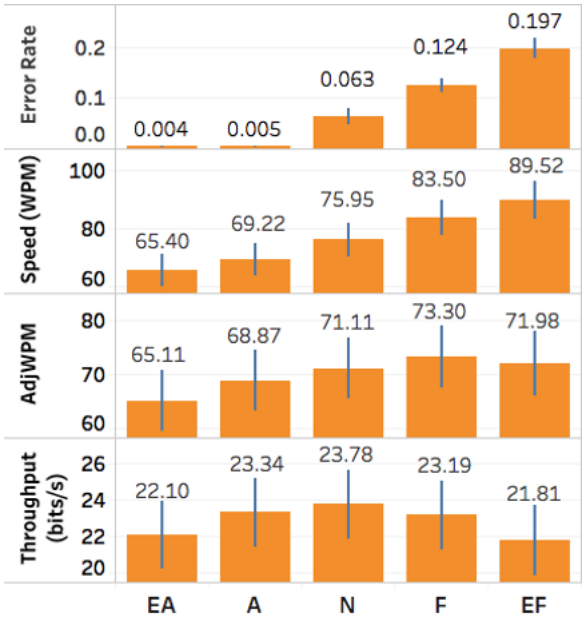

Figure 9.7: Results for the laptop keyboard in five cognitive sets, from Extremely Accurate to Extremely Fast. Error bars represent ±1 standard error. Note the very different y-axis ranges, which are set for visual comparison of differences within each metric across speed-accuracy conditions.

was heavily accuracy- or speed-biased. Not so for AdjWPM, which generally increased with WPM, indicating that it was not compensating much for increasing errors, as throughput was. The shape of throughput results across speed-accuracy conditions aligns with the findings of Liu et al. [131] on command selection, which showed a similar pattern for throughput in pointing.

To quantitatively evaluate the results, I performed a non-parametric analysis on the four text entry metrics. Friedman tests showed there was a significant main effect of speed-accuracy condition on error rate ($\chi^2_{(4,N=80)} = 56.65, p < .001$), speed ($\chi^2_{(4,N=80)} = 61.00, p < .001$), AdjWPM ($\chi^2_{(4,N=80)} = 23.15, p < .001$), and throughput ($\chi^2_{(4,N=80)} = 23.25, p < .001$). For error rate and speed, Wilcoxon signed-rank tests using Holm's sequential Bonferroni procedure [94] to correct for multiple comparisons showed that every condition was sig-

nificantly different from every other ($p < .005$). However, for AdjWPM, there were no significant differences between the Neutral and Fast or Neutral and Accurate conditions, but Fast and Accurate, and Extremely Fast and Extremely Accurate, were significantly different ($p < .05$). As shown in Figure 9.7, AdjWPM showed an ascending trend from *EA* to *F*, with faster conditions generating higher AdjWPM.

The significant effect of speed-accuracy condition on throughput also led us to investigate pairwise differences. Using Wilcoxon signed-rank tests corrected with Holm's sequential Bonferroni procedure [94], I found no significant differences between each pair of Accurate, Neutral, and Fast. Interestingly, there was also no significant difference between Extremely Accurate and Extremely Fast, indicating that throughput decreased similarly in both extreme conditions. The two Extreme conditions were significantly different from the Neutral condition ($p < .05$), which was reasonable because there were unnatural constraints in the Extreme conditions: in the *EA* condition, one had to press each key correctly to get points; in the *EF* condition, one had to type as physically fast as one could. Similarly, there was a significant difference between *EA* and A ($p < .05$), and *EF* and F ($p < .05$).

To provide further evidence of the stability of throughput across speed-accuracy conditions, I applied bootstrapping on the experiment data. Bootstrapping is a statistical test that randomly samples the data with replacement. It is useful for small data sets, and can estimate confidence intervals (*CI*). To test how similar throughput is between two conditions, I subtract one condition from another, resulting in a new data set of differences. I then resampled the data 10,000 times and derived an estimated 99.5% *CI* from 0.25% to 99.75%. The interval was corrected with the Holm-Bonferroni correction [94], as there were 10 pairwise comparisons among the speed-accuracy conditions in total. If zero was located inside the *CI*, there might not be a significant difference between any two conditions. The results from bootstrapping were similar to the significance testing results. No significant differences were found between the *A/N/F* conditions, or between the *EA/EF* conditions (*N-A* 99.5% *CI* [-0.18, +1.04]; *N-F* 99.5% *CI* [-0.31, +1.88]; A-F 99.5% *CI* [-1.00, +1.43]; *EA-EF* 99.5% *CI* [-1.24, +2.01]). Collectively, these results indicate that the throughput measure is quite stable across speed-accuracy conditions.

9.7.2 Smartphone Keyboard Condition

In all, 4811 phrases were collected on the smartphone. There were 1494 phrases from warm-up and practice sessions, which were not included in the analysis. The average uncorrected error rate, speed, AdjWPM, and throughput are shown in Figure 9.8.

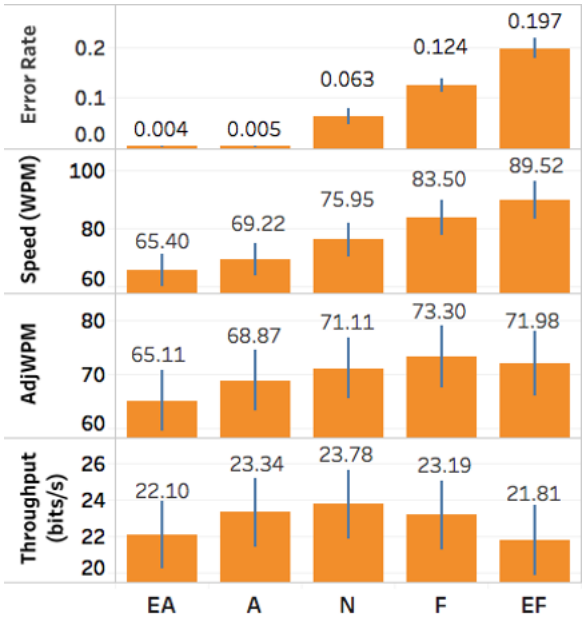

Figure 9.8: Results for the smartphone keyboard in three cognitive sets, from Accurate to Fast. The Advanced plots are for the keyboard with auto-correction, word completion, and word prediction. The Plain plots are for the same keyboard without these advanced features. Error bars represent ±1 standard error. Note the very different y-axis ranges, which are set for visual comparison of differences within each metric across speed-accuracy conditions.

Non-parametric Friedman tests showed that there were significant differences in throughput among different speed-accuracy conditions for both the Advanced ($\chi^2_{(2,N=36)} = 12.50, p < .005$) and Plain ($\chi^2_{(2,N=36)} = 17.17, p < .001$) keyboards. This led us to perform post hoc pairwise comparisons using Wilcoxon signed-rank tests corrected with Holm's sequential Bonferroni procedure [94]. I found that with the Advanced keyboard, there was a signifi-

cant difference between *N/A* ($p < .05$), but no significant difference between *A/F* and *N/F*. With the Plain keyboard, the difference was significant between *N/A* and *A/F* ($p < .05$), but not between *N/F*. For other metrics, pairwise comparisons between the *A/N/F* conditions were significantly different in error rate and speed for both smartphone keyboards. For AdjWPM, the only non-significantly different pairwise comparison was *N/F* for the Advanced keyboard.

Thus, although throughput showed an ascending trend with entry speed, the difference between each speed-accuracy condition was smaller than for the other text entry metrics.

Interestingly, Wilcoxon signed-rank tests also showed that there was no significant difference between the two keyboards in any cognitive set condition, which means that performance was not significantly different whether typing with or without auto-correction, word completion, and word prediction.

9.8 Discussion

The laptop keyboard condition provided empirical support for the practical value of the new throughput metric and its estimation for text entry. From the consistent results under neutral, speed-biased, and accuracy-biased conditions, I showed that throughput was relatively stable. Because the *EA/EF* conditions were intentionally extreme and unnatural, it was reasonable that throughput decreased in such conditions. Actual text entry evaluations will not generally require participants to perform in such unnatural ways, so the practical impact of the *EA* and *EF* conditions is minimal.

For both keyboards, the *F/EF* conditions had higher AdjWPM than in other conditions, indicating that the faster participants typed, the better they performed under this metric, even though their error rates increased significantly. This finding is evidence that AdjWPM is speed-biased and fails to adequately correct for increasing errors. In contrast, the new throughput metric showed a bell-shaped performance curve in the laptop keyboard condition, indicating the existence of a speed-accuracy tradeoff. As the difficulty of the task increased, participants no longer typed within their comfortable performance range, and thus their throughput decreased. The same conclusion was also reached in prior work [131, 198],

albeit for tasks other than text entry.

In the smartphone keyboard condition, although the Accurate condition had the lowest throughput, that was reasonable considering that making corrections and typing every character correctly on a touch screen was more difficult than doing so on a physical keyboard. It was also interesting that there was no significant increase in throughput using autocorrection, word completion, and word prediction. Looking at the average speed and error rate, I could even see drops in performance in some conditions with these advanced features. Some participants reported that they felt distracted looking at and selecting from a word list during typing, which might have increased their entry time. One participant reported that each time after she pressed a key, she spent time searching the word list to see if there was an expected word. Thus, the uncertainty brought about by the advanced features might have increased the response time of participants. Similar findings about how advanced features can harm typing performance have been reported elsewhere [116].

On the whole, the experiment evaluated the new throughput metric for text entry across different speed-accuracy conditions using a differential reward mechanism. All of the participants understood the bonus scheme and how to maximize their points in each condition. However, they found it hard to distinguish between Fast and Extremely Fast conditions, which indicated that it was generally difficult to manipulate more than three cognitive sets (Accurate/Neutral/Fast) unless there was a clear difference between the criteria in each condition. From the observations, most participants typed in an accuracy-biased manner—they tended to correct most errors. When typing in a speed-biased manner, they first felt uncomfortable letting errors remain.

As a metric, throughput also exhibited other desirable properties: the data-collection process was exactly the same as in current text entry evaluation processes [139], and the calculation was straightforward following the steps I outlined above.

9.8.1 Limitations

In the throughput algorithm, I average the overall probability of insertion, omission, substitution and correct-entries to calculate each $p(i \rightarrow j)$, as the throughput metric is method-independent. However, researchers dealing with their own specific text entry methods might want to get the exact probability of each error instance. For example, one could simulate the transmission probability based on the interface (e.g., keyboard layout). Using overall error rates as an approximation for each character-level error is the major limitation of this work—but as limitations go, it is a practical, not theoretical, one.

9.9 Future Work

I see at least four possible directions for future work. First, one could experiment with different input methods, which might include speech, gesture, and other new text entry interfaces. In this study, I only evaluated keyboards. However, the throughput metric will work with any text entry method. Second, one could experiment with other languages in which the character information and the input method are different from English. For example, Chinese is a logogram-based language very different than a phonogram-based language like English. Third, in the throughput calculation, one could use entropy on higher-level language constructs, such as the word-level. For example, Shannon [191] pointed out that English word-level entropy is 11.82 bits per word. This might be useful for evaluating the "semantic performance" of a method. And finally, there could be more accurate and robust ways of estimating throughput other than the error category estimation method I took in this study.

9.10 Conclusion

In this work, I presented a new text entry method-independent unified speed-accuracy metric throughput built on Shannon information theory. The experiment on laptop and smartphone keyboards showed that throughput was stable across different speed-accuracy biases compared to other metrics like error rate, words per minute, and adjusted words per minute. The hope is that throughput will be calculated and reported to support compar-

isons across devices, text entry methods, and participants. The advantageous properties of throughput and the theoretical derivation from information theory make it suitable as a performance-level metric. At the same time, speed and accuracy should also be reported, as they provide essential practical insight, even if they are more dependent than throughput on task-specific experimental conditions.

Chapter 10

A COMPARATIVE STUDY OF LEXICAL AND SEMANTIC EMOJI SUGGESTION SYSTEMS

Emoji suggestion systems based on typed text have been proposed to encourage emoji usage and enrich text messaging; however, such systems' actual effects on the chat experience are unknown. I built an Android keyboard with both lexical (word-based) and semantic (meaning-based) emoji suggestion capabilities and compared these in two different studies. To investigate the effect of emoji suggestion in online conversations, I conducted a laboratory text-messaging study with 24 participants and a 15-day longitudinal field deployment with 18 participants. I found that participants picked more semantic suggestions than lexical suggestions and perceived the semantic suggestions as more relevant to the message content. Our subjective data showed that although the suggestion mechanism did not affect the chatting experience significantly, different mechanisms could change the composing behavior of the users and facilitate their emoji-searching needs in different ways.[1]

10.1 Introduction

Most forms of text-based computer-mediated communication (CMC) lack nonverbal 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 [44, 101, 241]. Emojis are already widely used in text-based CMC, with nearly every instant messaging platform supporting their entry. Five billion emojis were sent per day on Facebook Messenger in 2017 [33], and half of all Instagram comments included an emoji as of mid-2015 [57].

[1]This chapter is adapted from: Mingrui "Ray" Zhang, Alex Mariakakis, Jacob Burke and Jacob O. Wobbrock. (2021). A comparative study of lexical and semantic emoji suggestion systems. Proceedings of iConference 2021. Beijing, China (March 17-31, 2021). Lecture Notes in Computer Science. Switzerland: Springer.

Table 10.1: 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"	🎈🎁🪂🎉💗	✌️😒😑😐😪

Many mobile keyboards offer emojis as a set of pictographic Unicode characters. 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 [100,229]. These products usually come in two variations—lexical and semantic suggestions—as shown in Table 10.1. 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 [100]; Figure 10.1), proposed emojis are based on the meaning of the message's content rather than on specific keywords.

Although emojis themselves are known to enrich conversations [44,101], 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 [19, 56, 63]. 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 three

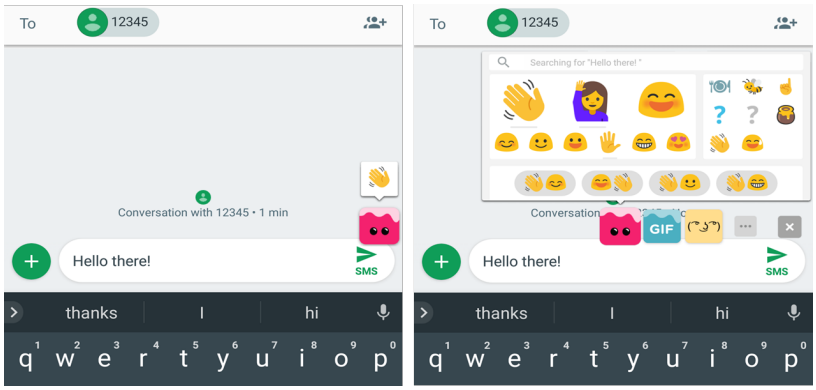

Figure 10.1: The semantic emoji suggestion application Dango [100]. When text is typed, Dango pops up a suggested emoji based on semantic message content. The user can tap on an icon to see more options.

emoji suggestion mechanisms: no suggestions, lexical suggestions, and semantic suggestions. The results showed that the chatting experience between strangers is not influenced by the emoji use, which was not explored by previous literature focusing on communication between friends and family members [44,101]. To evaluate the emoji usage in daily settings, I also conducted a 15-day field deployment. I found that emoji suggestion systems increased emoji usage overall, with users picking more emojis via semantic suggestions versus lexical suggestions or no suggestions. I also 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. Semantic suggestions were perceived as more relevant to the message content, while lexical suggestions were perceived as containing more unusual emojis. The semantic suggestions served as a clue to the tone of the message and even changed the user's input behavior in some cases. Based on the findings, I propose several design guidelines for emoji suggestion systems.

The contributions of this work are: (1) results from an in-lab study comparing emoji suggestion mechanisms within the mobile chat experience; (2) results from a longitudinal field deployment that tracked realistic usage of emoji suggestion systems; and (3) design guidelines of emoji suggestion systems based on the findings from the studies.

10.2 Emoji Keyboard Implementation

I built the Android keyboard using the open source project *AnySoftKeyboard*[2]. The keyboard interface is shown in Figure 10.2. The keyboard uses the default auto-correction mechanism, but the word-suggestion feature is replaced with emoji suggestions. Users can enter special characters or numbers by tapping the upper-left button; they can enter emojis by tapping the lower-left button. Note that tapping the emoji suggestions would add the corresponding emoji in the text rather than replace the text with the emoji.

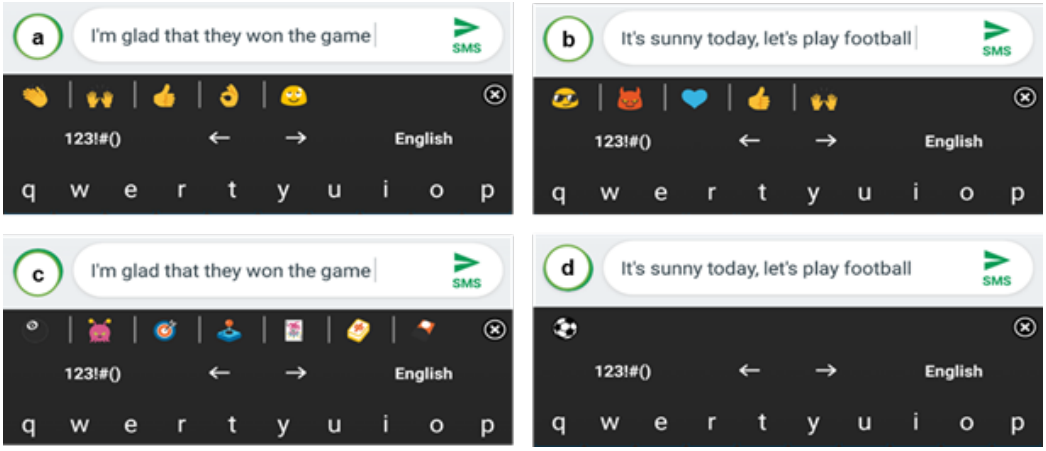

Figure 10.2: (a, b) Semantic suggestion in the keyboard implementation always provides five emojis based on the message's content. (c, d) The number of emojis provided by lexical suggestion varies according to the number of keywords present. (c) If a keyword is related to many emojis, the user can scroll to select them. (d) When there is only one emoji related to the keyword "football," only that suggestion is shown.

10.2.1 Emoji Suggestion Mechanism

The overall text entry interaction of the keyboard is shown in Figure 10.3. As a user types in the text box, the keyboard provides word suggestions in the candidate list. When the user finishes typing a word, the keyboard suggests emojis instead of words in the candidate

[2]https://github.com/AnySoftKeyboard/AnySoftKeyboard

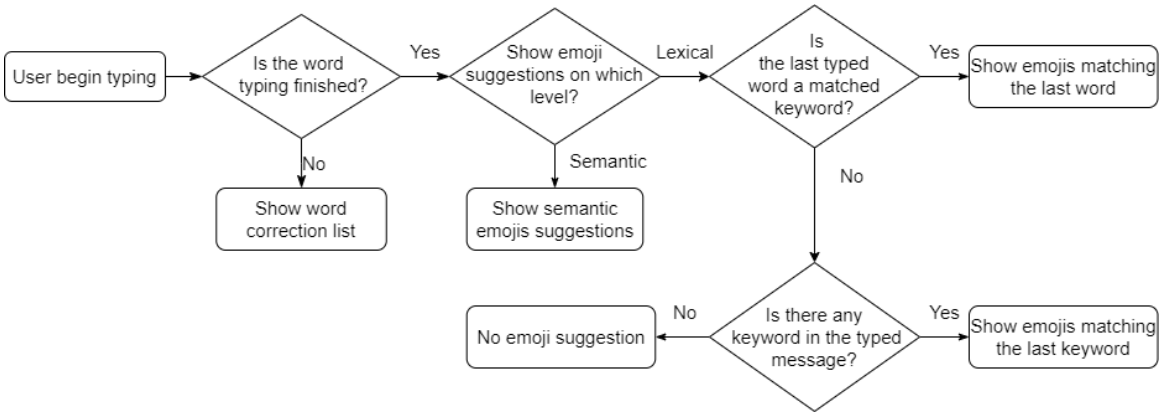

Figure 10.3: A diagram of the text entry and emoji suggestion process with the keyboard

list. If the user picks an emoji from the list, it is inserted at the end of the message.

The suggestion result varies based on the mechanism in use. With semantic suggestion, the keyboard always presents five emojis after the user finishes typing a word. Suggestions are generated using the DeepMoji model [63] running on a remote server. The keyboard sends an HTTP POST request to the server each time the user finishes typing a word, and the server returns the top-five related emojis. The amount of information transmitted is small and there were no latency concerns in the implementation or studies.

With lexical suggestion, the keyboard suggests emojis only if the keyword list contains the last-typed word. If no emoji matches the last-typed word, the keyboard presents the most recent suggestions. For example, if the user types "football field," the keyboard will continue to suggest the football emoji ⚽ because there is no lexical match with the word "field." If no word anywhere in the message has a match in the emoji keyword list, the keyboard provides no suggestions. Lexical suggestion is implemented using the open-source emoji library emojilib[3]. The library provides a .json file containing 1,502 emojis and their corresponding keywords. For example, the clapping emoji 👏 has words "hands," "praise," "applause," "congrats" and "yay."

[3]https://github.com/muan/emojilib

10.2.2 Design Rationale

I did not force the frequency of emoji suggestion updates to be the same for both lexical and semantic suggestion mechanisms, as these mechanisms are fundamentally different in nature. For lexical suggestion, the opportune moment for updating the emoji suggestions is straightforward—whenever a keyword has been typed. For semantic suggestion, however, it is unclear when the suggestions should be updated because it is not obvious when the user is finished typing. Thus, the keyboard updates the emoji suggestions after the user finishes typing each word, not just keywords.

I also did not force an equal number of emoji suggestions across the two keyboards. Using a machine learning model for semantic suggestion returns a fixed number of emojis (five in DeepMoji), but lexical suggestion can produce a variable number of emojis. Adding extra emojis when lexical suggestion produces too few emojis would confuse users with unrelated emojis, and conversely, trimming potentially relevant emojis from the semantic suggestions would make for a keyboard unrepresentative of its full potential.

10.2.3 Data Logging

The keyboard logs input statistics related to text and emoji entry: the number of typed characters, the number of deleted characters, the number of emojis manually added from the traditional emoji enumeration interface, and the number of emojis selected from the two suggestion lists. To respect participants' privacy, the keyboard did not log the content of any typed messages.

10.3 Laboratory Experiment

I first conducted a controlled lab study to investigate the usage of emoji with different suggestion levels between strangers. Previous studies [110, 241] indicated that emoji usage between people in close relationships can foster communication. However, people also spend a huge amount of time with acquaintances, strangers, or online communities, and the effect of emojis and emoji suggestion mechanisms can be unclear in those situations. Hence, I recruited participants who did not know each other before the study to simulate such

situations.

10.3.1 Participants

Twenty-six participants (15 females, 11 males) between 18 and 34 years old (M=28.9, SD=4.2) were recruited via emails, word-of-mouth, and convenience sampling in a large university setting. The participants were randomly divided into 13 pairs (one pair was removed for analysis in the later section). The pairs were constructed such that the participants did not know each other and did not meet face-to-face until the end of the study. Each participant was given $8 USD as compensation for the 30-minute study.

10.3.2 Apparatus

Participants were provided with Nexus 6P smartphones running Google Android 7.0. The keyboard was installed on each phone. *Wechat*[4] was used as the instant message application because Wechat provides a function to export the chat history. I used the chat history to verify the data logged by the mobile keyboard. Participants were instructed to avoid using Wechat's built-in button for emoji entry since it bypassed the keyboard's logging functionality.

10.3.3 Procedure

Participants were told that they would take part in an online chat experiment using the mobile keyboard. They chatted with another participant for three 10-minute sessions, each of which was assigned to one of three emoji suggestion conditions: no-suggestion, lexical, or semantic. The order of the conditions was fully counterbalanced across participants. The participants were told that they could steer the conversation towards any topic of their choosing but were told that a "recent activity" could be used to start. I did not constrain their topics as each pair may have been more comfortable discussing their own topics. The participants were also told that the only difference between the sessions would be the keyboard's emoji suggestion results, but they were not told anything about the suggestion

[4]https://www.wechat.com/en/

mechanisms—neither what they were nor how they worked.

Before the conversation began, the participants were told to fill out a questionnaire that asked about their online chat and emoji use behaviors, including their online communication frequency and their emoji usage frequency during online communication. After each session, the participants filled out another questionnaire asking about their chat experience (Table 10.2). This questionnaire probed their engagement (Q1, Q2, and Q6) and perceived expressiveness and clarity (Q3, Q4, and Q5) during the chat experience. Both questionnaires were derived from prior work on CMC [217]. When participants used lexical or semantic suggestions in a session, they also completed the usability questionnaire shown in Table 10.2, which was adapted from the SUS survey [29]. At the end of the 30-minute session, participants were interviewed with two open-ended questions: (1) "How do you like the suggestion keyboards? Do you find they affect you (in negative or positive ways) in online chatting?" and (2) "Do you find any problems with the keyboard suggestion mechanism, or do you have any suggestions?"

10.3.4 Design & Analysis

The study was a single-factor three-level within-subjects design with the suggestion mechanism as the independent variable: no-suggestion, lexical, and semantic. I utilized multiple statistical analyses according to the nature of the dependent variables: character count measures were analyzed using the aligned rank transform procedure [244]; emoji count measures fit a Poisson distribution, and were therefore analyzed with mixed model Poisson regression; Likert-scale responses were treated as ordinal measures and were therefore analyzed with mixed model ordinal logistic regression. Further specifics are given with each analysis in the results.

10.4 Results of the Laboratory Study

In this section, I describe the results of the study comparing the three levels of the *suggestion* factor: no suggestions, lexical suggestions, and semantic suggestions.

During the study, one pair of participants did not conduct what I considered a realistic

Table 10.2: The survey questions about the chat experience and the usability survey for the suggestion keyboards. Answers were provided via Likert scales ranging from 1 (strongly disagree) to 7 (strongly agree).

Chat Experience Questions
Q1. The chatting experience was interesting.
Q2. My attention was focused on the conversation.
Q3. I could express my emotion clearly using the keyboard.
Q4. I felt constrained in the types of expressions I could make.
Q5. I was able to get an impression of my partner.
Q6. The chatting experience excites my curiosity.
Usability Questions
Q1. I used the emoji suggestion a lot in my typing, and it was useful.
Q2. I would like to use this system frequently.
Q3. I thought the system was easy to use.
Q4. The system did well on proposing relevant emojis.
Q5. I like the emoji suggestion system better than the no-suggestion system.

conversation. In one of their sessions, they sent only nonsensical numbers and capital letters to each other. This participant pair was therefore excluded from the analyses, and another pair was recruited in their place. Thus, the dataset included 12 valid participant pairs with two pairs per Suggestion order due to full counterbalancing (3! conditions = 6 orders). I collected 12×3=36 data logs of valid sessions, together with 72 surveys regarding the chat experience and 48 usability surveys for emoji suggestion. I conducted formal analysis with open coding, in which research team members identified any themes or codes they discovered from the 48 responses to the open-ended questions on their online chat behaviors.

10.4.1 Participant Phone Use

Among the 24 participants, 22 stated that they always communicate with their phone, while the other two stated that they only used their phone sometimes. Nine participants stated that they always use emojis in online conversations, 14 sometimes, and one seldom. As

for how the participants normally enter emojis, 14 participants manually selected emojis from a list, one participant used lexical suggestions from the keyboard, and nine used both methods.

10.4.2 Count Measures

The descriptive results of the logged data are shown in Table 10.3. *Total Characters* is the number of characters excluding emojis sent in the conversation; *Total Emojis* is the number of emojis used in the conversation, however they might have been inputted; and *Selected Emojis* is the number of emojis picked from the suggestion list.

Table 10.3: Means (and standard deviations) of *Total Characters*, *Total Emojis*, and *Selected Emojis* in three conditions.

	Total Characters	Total Emojis	Selected Emojis
No suggestions	545.33 (211.58)	2.17 (2.85)	N/A
Lexical	542.04 (224.42)	3.29(3.51)	0.88 (1.33)
Semantic	579.79 (239.38)	3.29 (2.93)	2.17 (2.37)

A non-parametric aligned rank transform [244] with a mixed model analysis of variance was performed on *Total Characters*. Suggestion had no significant effect on *Total Characters* ($F(2, 46) = 0.78$, n.s.), indicating that the suggestion mechanism did not affect the overall volume of characters participants exchanged.

Total Emojis and *Selected Emojis* were conditionally fit to a Poisson distribution, as is common for count data [20], and mixed model Poisson regression was conducted on both measures. Suggestion had only a marginal effect on *Total Emojis* ($\chi^2_{(2,N=48)} = 5.25, p = .072$). However, Suggestion did have a significant effect on *Selected Emojis* ($\chi^2_{(1,N=48)} = 7.76, p < .05$), with semantic suggestion resulting in more selected emojis than lexical suggestion. This result indicates that although the total number of emojis participants used across conditions was similar, participants selected more semantic-generated emojis than lexical-generated ones.

10.4.3 Subjective Results

Participants responded to the questionnaires along a 7-point Likert Scale (1 = strongly disagree, 7=strongly agree), so the data were analyzed using mixed model ordinal logistic regression. Surprisingly, there were no significant results across the different Suggestion levels for any question regarding either the chat experience or usability (Table 10.2).

Looking at participants' interview answers, I found that participants did notice the difference between the suggestion mechanisms and provided more positive feedback on semantic suggestions than the other conditions. Five participants mentioned that semantic suggestions were convenient and timesaving. The convenience came from the relevance of the semantic suggestions. P13 pointed out, "*The first one [semantic] is better than the second one [lexical], showing more emotion-related emojis. The second one is related to the word itself and it makes no sense to use the emoji in the conversation.*" P25 preferred the semantic-level prediction because it was "*reflective of the tone of the message.*" I also found that the appearance of semantic level suggestions enriched the composer's chatting experience: Although P19 did not use many emojis during the study, she stated that "*their [emojis'] appearance in suggestion bars makes me feel good.*"

On the other hand, two participants preferred word-level prediction, and they did so because it sometimes provided more unusual emojis than semantic-level prediction. P18 said, "*the (keyword) prediction is fun because it predicts more unusual emojis, and that's unexpected.*" Five participants wanted more options from the semantic-level prediction. P1 suggested, "*Increase the amount of emoji that are an option. If you don't have much options to put for prediction, use the most frequent used emoji as an option for the user.*" Participants also mentioned that they did not usually insert emojis within their messages, so it would be less distracting if the suggestions were only relevant for the end of their sentences.

10.4.4 Discussion of the Laboratory Study

Based on the analysis of emoji counts in the study, I found that although different suggestion levels resulted in similar amounts of inputted emojis, participants tended to pick more from

semantic suggestions than from lexical suggestions. The finding was that the suggestion type did not affect the chat experience significantly. One explanation is that different suggestion mechanisms only affect how the user inputs emojis, rather than which emojis they input. As long as they can input the expected emojis, the chat experience is not affected.

Another interesting fact was that the emoji usage did not affect the chat experience between strangers, which was not covered in previous literature. Three participants mentioned that they did not feel comfortable sending emojis to strangers. P2 wrote, "*To be honest, I am indeed more engaged with the emoji prediction system but I do not think I got a 'full' sense because I use emoji less when chatting with strangers than with friends.*" This indicated that although emojis can foster the communication between closed relationships, people are less motivated to send emojis to strangers, which can be explained by that strangers shared less common ground, thus their conversations were "more superficial and general in culture" [40].

10.5 Field Deployment

I then conducted a 15-day field deployment to explore the longitudinal effects of the different emoji suggestion systems. Unlike the in-lab study which was conducted between strangers, this study focused on the usability of the emoji suggestion systems and on their effects on emoji usage during everyday conversations.

10.5.1 Participants

Eighteen participants (8 females, 10 males) between 18 and 43 years old (M=24.0, SD=6.4) were recruited via emails, flyers, and word-of-mouth. Inclusion criteria required that participants were able to use English as their primary language and owned a smartphone with Android version above 6.0 that they used on a daily basis. Those who were in the laboratory study were not allowed to participate in the field deployment due to prior exposure. The 15-day study contained three five-day periods. Participants were compensated $20 USD in the first two periods and $40 for the third, adding to $80 total.

10.5.2 Procedure

The study was conducted as a partial within-subjects design with the suggestion mechanism as the independent variable. All of the participants used the *no-suggestion* keyboard in the first five-day period as a baseline (however, they could still input emoji from the emoji selection panel). During the second period, half of the participants used the *lexical suggestion* keyboard while the other half used the *semantic suggestion* keyboard. Everyone returned to the *no-suggestion* keyboard during the last period to determine whether they returned to their baseline behavior. In psychology terms, the study compared an ABA condition sequence to an ACA condition sequence.

When participants were enrolled, they were asked to fill out the same questionnaire about online chatting and emoji usage as in the laboratory study. Participants were told that they would be using an emoji suggestion system during the field study, but that they were free to use or ignore the suggestions as they pleased. Participants were instructed to use the keyboard whenever they were typing in English and to keep their phone network connected so they could retrieve emoji suggestion results. The same usage information was logged as before (*Total Characters, Total Emojis*, and *Selected Emojis*). After participants signed the consent form, the keyboard was installed on their phone. The keyboard was configured to participants' personal preferences, including its aesthetic theme and vibration behavior.

Participants met with a researcher after each five-day period to have their keyboards reconfigured to another condition and fill out a short questionnaire about the experience (Table 10.4). After the second period, when emoji suggestions were provided, participants also completed the same usability survey as in the first study (see Table 10.2).

10.6 Results of Field Deployment

I collected 54 data logs (18 participants × 3 periods), 18 survey results about the usability of emoji suggestions, and 54 open responses analyzed using inductive analysis [213]. As before, *suggestion* was the independent variable of three levels: no-suggestion, lexical, and semantic.

Table 10.4: The survey questions after each period. The emoji suggestions were offered only during period 2, which is why the questions are different for that period.

Survey After Period 1
Do you find yourself using emojis more or less often than before the study? Why?
Survey After Period 2
1. How do you like or dislike the suggestion keyboard? Do you find it affecting you (in negative or positive ways) in online communication? 2. Do you find yourself using emojis more often than before the study? Why? 3. Do you have any comments about the keyboard emoji suggestions?
Survey After Period 3
1. What do you think of the current keyboard for this period? 2. Do you find yourself using emojis more or less often than before the study? Why? 3. After the whole period, do you have any comments about the keyboard emoji suggestions?

10.6.1 Participant Phone Use

Among the 18 participants, 14 stated that they always communicate with their phone, three sometimes, and one seldom. Four participants stated that they always use emojis in online conversations, 11 sometimes, and three seldom. As for the participants' typical emoji entry method, 10 participants manually selected emojis from a list, one participant used lexical suggestions from the keyboard, and seven used both methods.

10.6.2 Count Measures

The descriptive statistics for *Total Characters, Total Emojis* and *Selected Emojis* per day are shown in Figure 10.4. Unsurprisingly, participants used more emojis with lexical and semantic suggestions than with no suggestions. On average, participants who used lexical suggestions in the second period increased their emoji usage by 31.5% over their baseline, while participants who used semantic suggestions increased their usage by 125.1%. I note that the average usage of daily emoji seems low (fewer than 5 emojis per day). After looking

into the data, I found that some participants used over 10 emojis per day, while the other participants used less than one emoji per day.

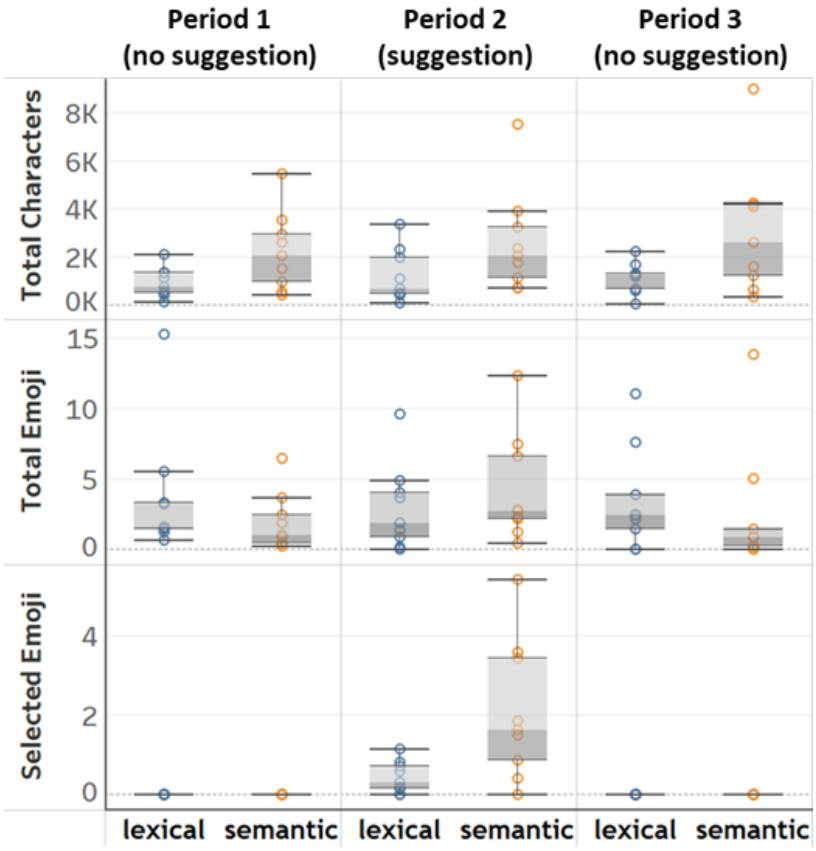

Figure 10.4: Box plot for Total Characters, Total Emoji, and Selected Emoji per day from the field deployment dataset. Within each period, the left box indicates the lexical keyboard group, while the right box indicates the semantic keyboard group.

A Wilcoxon signed-rank test was performed on *Total Characters* and *Total Emojis* between the first and second periods for each group separately. *Total Characters* was not significantly different between the two periods for either Suggestion condition. *Total Emojis* was significantly different between the two periods for semantic suggestions ($p<.05$), but not for lexical suggestions. Despite the fact that emoji usage increased in both conditions, only semantic suggestions encouraged participants to input more emojis.

A Mann-Whitney U test was performed on *Total Characters*, *Total Emojis*, and Selected Emojis by Suggestion for the second period in which the suggestion keyboards were used. The test revealed no significant differences between semantic and lexical suggestions for *Total Characters* and *Total Emojis*; however, semantic suggestions resulted in significantly more Selected Emojis than lexical suggestions ($Z=-2.43$, $p<.05$), indicating that those who used semantic suggestions entered a larger proportion of emojis from the suggestion list than from manually picking. This result aligned with findings from the in-lab study.

Furthermore, I analyzed the difference in *Total Emojis* between the different periods by Suggestion using Mann-Whitney U tests. Results showed that emoji usage increased significantly more with semantic suggestions than with lexical suggestions from the first to second period ($p<.001$). The change between the first and third periods was not significantly different, indicating that the change in emoji usage was due to the emoji suggestion and not just time.

10.6.3 Questionnaire Results

The Likert scale responses from the usability survey during the second period were analyzed using mixed model ordinal logistic regression. No statistically significant differences were found between the semantic and lexical suggestions for any of the questions.

10.6.4 Discussion of the Field Deployment

The quantitative analysis results are similar to the in-lab study: the total emoji inputs were similar between different suggestion levels in period 2, and users chose more semantic suggestions than lexical suggestions. Again, based on the survey results, suggestion mechanisms did not influence the online conversation experience significantly.

Semantic suggestion group participants liked the **convenience** of the prediction, mentioning that the auto-generated emojis saved their time and "resulted in a faster and better product in regard to being able to seamlessly add emojis into everyday text" (P5). Another frequently mentioned advantage was **relevance**. As a consequence, participants mentioned that the semantic emoji suggestions helped them to **understand the tones** of their mes-

sage:

> *I must say that the predictions were accurate most of the times … It could guess when my sentences have a positive connotation and a negative one.* (P11)

More interestingly, two participants mentioned that the suggestions **altered their original language style**:

> *I feel that there have been a few instances in which I would use a particular emoji when using a keyboard that was not enabled with emoji suggestion, and when this keyboard suggested a different emoji, I felt that it suited my preferences better.* (P15)

> *I would start phrasing the sentences differently to kind of trick the keyboard into predicting the specific emoji I want without having to go to the menu and select it manually.* (P5)

Lexical suggestion group participants expressed neutral opinions of the suggestion system. Two participants liked the relevance of the suggestions. For lexical level, the relevance is more providing related emojis on the literal meanings of a word:

> *I'm pleased that when [lexical suggestion] provided suggestions based on the context of a word, such as smiley faces when typing "happy".* (P13)

Participants also enjoyed the **various options** that lexical suggestions provided. More importantly, P1 mentioned a unique case that lexical-level suggestions might be good at— **ironic emoji usage**— where he used random emojis to express sarcasm:

> *I don't use emojis a lot, but when I do, they're usually in an ironic sort of way. The emoji suggestion keyboard allowed me to do this at times that I didn't think there was a relevant emoji.*

Comparing the responses after the second and third periods revealed suggestions for ways that the two suggestion mechanisms could be improved. For semantic suggestions, participants suggested **increasing the variety** of emoji options. P12 also mentioned about "sarcasm usage", as he "*often uses emojis to supplement or change the emotion of the mes-*

sage." P9 wished for a keyboard that could be aware of the app he was using and provide situational emojis. He noted that his mind-set "i*s very different when texting friends than when writing an email for work.*"

For lexical suggestions, participants wanted **more relevant suggestions**. P20 offered a detailed example: "*Sometimes the predicted emoji missed the meaning of what I was typing. For example, when responding to a friend who was apologizing to me, I typed, 'No worries.' I say this in a positive way, however, the emojis suggested were sad or anxious expressions, probably based on the last word typed, which was 'worries'. Therefore, the suggestion missed the intended meaning of the phrase, so maybe it would be impactful to work on the algorithm to detect multiple words/phrases to better understand the meaning within a message.*" The above observation is the very reason for why semantic suggestion systems have been proposed in the past [20, 63].

10.7 Discussion

The goal was to examine the impact of emoji suggestion on online conversations. In particular, I sought to answer two primary questions: (1) How do emoji suggestion systems affect the chat experience? (2) Do lexical and semantic suggestion systems affect daily emoji usage differently? I first conducted an in-lab study, finding that the suggestion systems in use did not affect the overall chat experience for conversation between strangers. A possible explanation is that the suggestion levels only affect the ease of inputting an emoji, e.g. *how they input emojis*, but not affect *whether to input an emoji* and *what emojis they input.* I also found that emoji usage did not significantly affected the chatting experience between strangers.

On the other hand, the field deployment revealed that the suggestion systems influenced the amount of emoji usages. For the semantic level group, even without knowing the details of the suggestion mechanism, participants were pleasantly surprised that the predicted emojis were related to the sentiment of their messages. During the field deployment, participants picked more emojis in their daily conversations from semantic suggestions than from lexical suggestions. I also found that the semantic suggestions provided the user with clues of the

message tone, which caused the users to input a different emoji and adjust their language styles. At the same time, lexical suggestions provided more diverse emoji options, which pleased the participants and enabled iconic usages.

10.7.1 Design Guidelines for Emoji Suggestion Systems

Based on feedback from the user studies, I propose several design guidelines for future emoji suggestion systems:

Suggestion Diversity. Emoji suggestion systems should suggest various types of emojis, ranging from emojis that portray objects to emojis that portray emotions. Although semantic suggestions were preferred in the study, many participants wanted the system to provide more suggestions than just face emojis. Some participants also appreciated that the lexical suggestion system would sometimes suggest rare emojis. Suggestions from multiple systems could be combined to provide more diverse emojis. Lexical suggestion could provide emojis as the user is typing a sentence, and once the user has finished the sentence, semantic suggestion could provide emojis that reflect the message's overall meaning. Combining the two suggestion schemes could be useful because not all messages contain strong semantic information, and people also use emojis to provide additional information for certain words [44], such as changing the tone.

Personalization and contextualization. Through the two studies, I found that the participants had various preference on emojis: those who liked semantic suggestions were in favor of facial emojis, while those who liked lexical suggestions were in favor of object and unusual emojis. Beyond providing the most common suggestions, emoji suggestion systems should be aware of the user's personal favorites and usage behaviors. Usage behaviors could be based on categories (e.g., faces, hearts) or the emotions that the user prefers to express. In addition, it would be useful if the suggestion keyboard could recognize the recipient or the usage scenario. For example, a user might want heart emojis when chatting with a family member on a messaging app, but object emojis when composing an email.

Avoiding Intrusion. Participants of the in-lab study mentioned that they only input

emojis after finishing typing the sentence, hence emoji suggestion keyboards should only predict emojis when necessary. Some participants only wanted suggestions at the end of messages, as they found the always-on style of semantic suggestions to be distracting.

10.7.2 Limitations

One limitation of the study is that the suggestion frequency of the two emoji systems was not the same. The semantic suggestion system updated with each new word typed, while the lexical suggestion system updated only after each pre-defined keyword. Thus, participants were exposed to more suggestions in the semantic condition than in the lexical condition. Collecting a similar measure could have been done in the other studies by counting the number of selected emojis and dividing by the total number of emoji suggestions; however, such a metric would neglect many other factors that affect selection rate (*e.g.*, time duration, ordering of emojis).

Another limitation is in the keyboard implementation, namely that the existing semantic-level suggestion model I used contains only 64 possible emojis, thus limiting the diversity of possible suggestions. The DeepMoji model could be extended to more emojis, but I chose to stay with the original set to align with the findings from Felbo et al.'s prior work [63] since there is no available conversation datasets with emojis for fine tuning the model.

10.8 Conclusion

In this work, I compared two emoji suggestion systems: lexical and semantic. Specifically, I explored whether the suggestion type affected the online chat experience and how people perceive the two suggestion types. The laboratory study showed that different emoji suggestion mechanisms did not affect the chatting experience with strangers. The longitudinal field deployment showed that semantic suggestions led to an increase in emoji usage and were preferred because of their relevance to emotions. As other research in this area has found [44, 101, 110], I can conclude that emojis themselves, rather than the type of suggestion system, affects the chat experience most profoundly. However, I found that semantic suggestions could be perceived as a tone clue of the message, and also affect the language

style of the user. I believe that by incorporating semantic information in emoji suggestion, researchers can provide better experiences in text-based computer-mediated communications.

Chapter 11

DISCUSSION

In this chapter, I will discuss the different levels of text entry research, situate the dissertation in the related work, and discuss what aspects text entry research—including accessibility and intelligent interactions—should focus on in the future.

11.1 Shifting the Focus of Text Entry Research Back to People

The development of text entry technology has three levels of focus: interaction-focused, intelligence-focused, and people-focused. Interaction-focused text entry research aims to improve the interaction efficiency on the operating level: using fewer motor actions to type a set of phrases as fast and accurately as possible. In this level, the center topics are (1) the interface of the keyboard, (2) coding systems of a language, and (3) modeling human motors to predict the performance of a potential text entry system. For (1), different layouts were invented to improve the motor efficiency of the interface. For example, the Dvorak [1] layout was to improve the typing efficiency by rearranging the keys so that typing common English text would require less finger motion than the QWERTY layout. Other alternative layouts such as the IJQwerty keyboard [27] were to improve the accuracy of gesture typing. For (2), when there are not enough keys to cover all letters of a language, coding systems decide the number of key presses required to produce a character. For English, most of the research focused on the chorded keyboard (e.g. pressing multiple keys to produce a character or phrase), such as Perkinput [14] and Twiddler [134]. The coding system design has an essential role when designing input methods for languages that have a large character set, such as Chinese. For (3), researchers combined evaluation metrics with human motor models to derive the theoretical performance of a text entry system, which can in return

[1] `https://en.wikipedia.org/wiki/Dvorak_keyboard_layout`

guide the design of new interfaces. For example, the famous Fitts' Law [67] which models the relationship of human finger motion with the interface layout was extensively applied in the layout design [26] and keyboard performance modeling [25, 204].

While interaction-focused research optimizes the interaction efficiency on the fundamental (human motion) level, language input is a high-level activity that operates beyond purely operating buttons and interfaces. The symbols being operated have meanings and use patterns that could be utilized to further improve communication efficiency. The second level, intelligence-focused text entry research, thus tries to model the language usage and assist its input via automatic predictions. In this level, the center topics are (1) language modeling, and (2) enabling novel text entry interactions. For (1), after Goodman introduced language modeling for soft keyboards [82], modern keyboards have adopted language models to map the noisy touch points to candidate words, based on the probability of the spatial distribution and the word usage frequency. Sophisticated modeling techniques such as Finite State Transducer [166] and neural networks [4] have boosted the accuracy of keyboards with noisy input (such as touch screen keyboards). For (2), the language modeling has enabled auto-correction and word-prediction features, and novel interfaces such as gesture typing [121], T9 keyboard [174], Dasher [234] are also heavily built upon the intelligent modeling algorithms. Because of those novel intelligent interactions, keyboards that require less physical space and less motor effort are made possible, such as keyboards on smartwatches [81, 223].

While the introduction of intelligent modeling techniques has significantly improved the input efficiency, it still focuses on the "text level input", where text here represents the symbol of the language, rather than the medium of the ideas. The ideal level of text entry systems should be people-focused: text entry systems need to improve their "interpretation" abilities so that they could understand the user's intention using contextual data. Here the context could be the surrounding text, the user's action in other modalities, the user's physical surroundings, the user's personal communication histories, and more. For example, WalkType [78] uses motion sensors to help the user type accurately when they are typing while walking; Smart Compose [36] and Smart Reply [108] use the metadata such as title,

sender, and surrounding text to provide email writing suggestions; ReType [195] utilizes the gaze data to help to locate the correction target for keyboard correction. Many projects in my dissertation aim to reach the people-focused level, such as by incorporating a larger language context (PhraseFlow and TypeAnywhere).

The trend of text entry research has gradually shifted from the interaction-focused paradigm into the people-focused paradigm, as the neural network models are capable of making sense of data from multiple sources: not only the text but also audio, images, and motion sensor data. The model can take a larger personal context to help the users express their minds, instead of only transcribing the text. However, making text entries people-focused does not conflict with interaction-focused and intelligence-focused levels; in fact, it will introduce new challenges and opportunities for interaction design, and intelligent models are the foundation of those interactions. Think of the system as a personal assistant—what will you say to an assistant to schedule a meeting? To correct an error? To send an email, or to search for a movie? How to design the interaction for those tasks, and how can we train a model to implement the features? People-focused text entry asks the researchers to think about the human experience of how people express their minds and design interactions and models based on such experience.

11.2 Advancing the Field of Information Input Research Beyond Desktop Computing Environment

The desktop computing environment has "forced" us to think in a way about what information input interaction should be: a keyboard for entering text character by character, a cursor pointing to the current modified position, and a mouse moving the cursor around. The way of thinking has been applied to many other computing platforms such as smartphones, TV, and voice-based interfaces, even if they do not equip with a keyboard and mouse at all. This pattern also influenced the way to input other information, such as the emoji keyboard, although that information did not have an alphabet. However, by adopting "the desktop way" for information entry, we could potentially limit the benefit of new interfaces, leading to an unnecessarily complex communication experience.

This dissertation work advances the evaluation and interaction paradigm of information input by providing alternative ways beyond the traditional desktop computing environment. On the evaluation side, The *Transcription Sequences* model introduces a flexible way of evaluating advanced features in the smartphone computing environment, such as auto-correction and word prediction. The *Text Entry Throughput* evaluates how much information is transmitted during the typing process, instead of focusing on the literal comparisons. On the interaction side, *Type, Then Correct* utilizes gestures and machine learning to bring new ways for correction without cursors. *PhraseFlow* introduces "delayed correction", by taking larger language context into consideration, which is different from the desktop paradigm where the character is fixed once they are typed. *TypeAnywhere* tries to resemble the desktop typing experience, but it only resembles the "typing" part: there is no cursor for navigation, correction is done in the *Type Then Correct* style, and the decoding is done in the *PhraseFlow* style.

Finally, the accessible graphic information systems advocate a conversational way for input interactions. For information that does not have an obvious alphabet, "hunt and peck" over a list of options might not be the optimal way to interact with them. As we tend to refer to a piece of information with its content when talking with others, using natural language descriptions as the input & output requires significantly less cognitive load.

11.3 Rethinking What is Needed for Accessible Communication

The media formats for digital information are and continue to be increasing. People do not only rely on text to talk with each other: stickers and emojis, video chats and streaming services, or even interactive articles and visualizations. The rich variety of information has expanded our ability to express ourselves, but at the same time, it also brings access problems for people with disabilities. Thus having the access to the text does not mean having equal opportunity for online communication. Instead of adding an extra layer and creating a specific solution for each format, I again argue for designing humanlike experiences to make the information accessible, which might be less deterministic but more conversational.

Contrary to much existing assistive technology that heavily relies on the system to au-

tomatically recognize and explain inaccessible information, I find human-based (such as peer-based or crowdsourcing-based) methods can be more efficient. One example is Ga11y, which uses crowdsourcing to provide annotations for animated GIFs. Unlike automated methods where all the rules and data are pre-defined (including machine learning models), human-based methods are more flexible, more personalized, and more adaptive to different information formats (especially for user-created content such as GIFs). However, there is a lack of a general platform that users can easily ask others for help. Many researchers have built crowd-sourcing platforms such as VizWiz [28] for photo annotations, Revamp [231] for online shopping, or online ASL forum for learning science [34], yet many tasks can be aggregated into a single platform (for example, the annotation platform Ga11y offers can be easily extended to other visual elements beyond GIFs). Another effort that needs to be done is to provide easy interfaces for the sender/creator to add accessible elements (for example, automatically adding and aligning text descriptions for videos [169]) so that the piece of information is "born with accessibility".

11.4 Balancing User Control and Automation in Communication Systems

According to Beaudouin-Lafon [21], 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. The end goal of humanlike communication systems is to become a partner, allowing the user to conduct natural conversations as if talking with another person. To achieve the goal, AI plays an important role in making sense of the input, interpreting the user's intention, and performing natural responses. However, AI usually behaves like a black box, introducing uncertainty into the interaction; on the other hand, communication tasks such as text entry require precision and predictability. Hence the challenge is how to design intelligent text entry technologies that maintain users' control.

However, adding "more automation" inherently sacrifices user control. In *PhraseFlow*, the user must type multiple words to enjoy the benefit of phrase-level correction; in *Type, Then*

Correct, the user performs a throwing gesture first without knowing the correction result. Assistive writing agents such as smart compose goes further - they even help generate the whole paragraph without allowing the user to intervene in the process.

While user control is limited in the direct interaction process, there are ways to add transparency and controllability, either through extra interactions (such as allowing the user to take further steps or configurations) or through feedback. A high level of automation can be favored as long as it builds trust with the user while maintaining certain controllability. In text entry systems, a typical example of offering both trust and controllability is the autocorrect interface of the smartphone keyboard. Autocorrection triggers whenever the user presses the space key after typing a word. However, such convenience also introduces awkward results when the correction fails. Designers had made various improvements to increase the controllability of the user, such as highlighting the best candidate before the space press to increase the predictability and utilizing backspace to revert correction as an easy undo mechanism. At the same time, better language models are constantly developed to increase precision, and personalized dictionaries are added to build trust with the user. There are multiple principles and guidelines on how to design the interaction with general AI systems [6,97,163]. And researchers who want to build a successful intelligent communication system should have a cost-benefit analysis of the trade-off between user control and automation.

Chapter 12

CONCLUSION

The dissertation has demonstrated how to design, build and evaluate more intelligent, intuitive and inclusive communication interactions using the power of artificial intelligence, in three directions: (1) for text information, it introduces intelligent text production interactions that understand the user's intention and accelerate the input process; (2) for pictorial information, it presents assistive systems designed with voice modality and crowdsourcing to help blind or low vision (BLV) users use and understand emojis and animated GIFs; (3) for evaluation, it proposes models, metrics, tools and empirical studies to measure the performance and impact of input technology equipped with advanced functionalities. Together, the work demonstrates the thesis statement: *Artificial intelligence can enable and improve advanced text production and accessible interactions with pictures; in addition, new metrics for text entry enable the evaluation of advanced capabilities.*

12.1 Summary of Contributions

The specific contributions of this dissertation include:

12.1.1 Artifact Contributions

Below are a list of interactive techniques and systems I contribute:

- 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 [273] (chapter 3).
- 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 [271] (chapter 4).

- *TypeAnywhere*, a ten-finger QWERTY style text entry interaction on any surface. 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 [275] (chapter 5).
- *Voicemoji*, a speech-based emoji entry interaction. *Voicemoji* contains several emoji entry commands and provides semantic emoji suggestions to support new emoji exploration [270] (chapter 6).
- *Ga11y*, an automatic GIF annotation system for blind or low vision users, combining the power of machine and human intelligence [276] (chapter 7).

Below are a list of open-sourced tools and systems I contribute:

- 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 (chapter 4).[1]
- The decoder model and the interface of *TypeAnywhere* is open-sourced.[2]
- The implementation of the *Voicemoji* system is open-sourced.[3]
- The system of *Ga11y* including the mobile client, the server and the annotation website is open-sourced.[4]
- The *TextTest++* web platform for conducting text entry evaluation tasks. *TextTest++*

[1] `https://github.com/DrustZ/CorrectionRNN`

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

[3] `https://github.com/DrustZ/VoiceEmoji`

[4] `https://github.com/DrustZ/Ga11y`

provides interface and evaluation metrics based on the *T-seq* model and the *text entry throughput*. The implementation is open-sourced.[5]

- The tool of calculating text entry throughput is open-sourced.[6]

12.1.2 Empirical Contributions

- The series of usability studies on the intelligent text entry interactions including *PhraseFlow*, *Type, Then Correct* and *TypeAnywhere*.
- Interviews to understand the current experience interacting with emojis and animated GIFs of blind and low vision (BLV) users on mobile devices [270,276], and the usability evaluation of *Voicemoji* and *Ga11y* systems (chapter 6, 7).
- In-lab and deployment studies on comparing the effect of lexical and semantic emoji suggestion mechanisms on online communication [269] (chapter 10).

12.1.3 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 [272] (chapter 8).
- The *text entry throughput*, a metric that unifies the speed and accuracy. *Text entry throughput* is derived from the information theory [191], and it is less sensitive to the speed-accuracy trade-off [274] (chapter 9).

12.2 Future Work

I plan to continue my goal of building and evaluating technology to facilitate human-machine communication. Specifically, I see opportunities and challenges for information communication systems with two trends: (1) the emergence of non-traditional computing platforms

[5] `https://github.com/DrustZ/TextTestPP`

[6] `https://github.com/DrustZ/Throughput`

such as AR/VR and wearable devices, (2) the wide deployment of AI-infused information systems on commercial products. I thus propose three future directions:

Building multi-modal input systems incorporating large scale contextual information. What does a future information input interaction look like? The ubiquitous computing era is here, and devices today evolve rapidly into various form factors. We therefore need to design input interfaces to make them easy to use in mobile and wearable settings and support cross-device input tasks. *TypeAnywhere* acts as my first step to the exploration. Other than the universal availability, there are also opportunities to bring different modalities into the communication process, instead of purely typing. Utilizing modalities such as gaze or facial expressions could provide extra contextual information (beyond the language context) to identify the user's intention (for example, ReType [195] utilizes gaze as a clue for locating error positions).

Moreover, with powerful hardware processors and advanced machine learning algorithms, our devices are now able to detect our activities from multiple data sources. In the future, I plan to build personalized context-aware input systems by utilizing off-the-shelf sensor data. For example, a prediction system that provides replies with different linguistic styles based on the app the user is using (e.g., email vs. message apps), or different contents based on a user's current activity (e.g., suggesting locations when the user is using a map when driving).

The second direction is to **design accessible tools for emerging forms of information.** Today's world is full of new forms of information beyond traditional text and images, including videos, interactive content on mobile apps, and AR/VR content. However, most of these emerging forms are inaccessible to people with disabilities such as motor, vision, or hearing impairments. I plan to address this challenge in two ways: 1) to support the understanding of the information: similar to my Ga11y work, I will combine machine learning and crowdsourcing to provide explanations of non-textual information sources. 2) to support the interaction with the information: intuitive and practical interactions can be approached through participatory design, which gains insights from target users' experiences.

The last direction is to **evaluate the broader impact of AI-infused communication systems.** How to design human-in-the-loop interactions for intelligent information systems, and how to evaluate their impact on the human behavior and society? Advances in AI have empowered commercial systems to intelligently present, recommend and collaborate with users to interact with information. Those systems, including predictive text systems such as Gmail Smart Reply and conversational agents such as Alexa, are widely deployed and used by billions of users. However, while the emergence of Artificial Intelligence-Mediated Communication (AI-MC) presents clear practical benefits on communication efficiency, their broader impacts, such as how they affect a user's mental models and linguistic routines, and how they are perceived in various social contexts, still need to be investigated. I plan to design metrics that take psychological and communication theories to evaluate intelligent language production techniques. Furthermore, most intelligent algorithms are deployed as black boxes, lacking transparency to the user. I am interested in designing human-in-the-loop interactions to allow end-users to understand and adjust intelligent systems, including deriving active-learning algorithms for personalization and building user-feedback mechanisms to tune underlining models. By understanding, evaluating, and designing intelligent information interaction systems, I aim to realize the vision of Human-Computer Symbiosis for AI-MC.

BIBLIOGRAPHY

[1] A. Agresti. *Analysis of Ordinal Categorical Data*. Wiley Series in Probability and Statistics. Wiley, Hoboken, NJ, USA, 2010.

[2] Kim Albarella. The secret language of emoji, 2018. `https://staysafeonline.org/blog/secret-language-emoji/`.

[3] Ohoud Alharbi, Ahmed Sabbir Arif, Wolfgang Stuerzlinger, Mark D. Dunlop, and Andreas Komninos. Wisetype: A tablet keyboard with color-coded visualization and various editing options for error correction. In *Proceedings of Graphics Interface 2019*, GI 2019, Kingston, Ontario, 2019. Canadian Information Processing Society.

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

[5] Rui A. Alves and Teresa Limpo. Progress in written language bursts, pauses, transcription, and written composition across schooling. *Scientific Studies of Reading*, 19(5):374–391, 2015. `https://app.dimensions.ai/details/publication/pub.1046423967`.

[6] 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. Guidelines for human-ai interaction. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, CHI '19, page 1–13, New York, NY, USA, 2019. Association for Computing Machinery.

[7] 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. 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, page 173–182. JMLR.org, 2016.

[8] Michelle Annett and Walter F. Bischof. Your left hand can do it too! investigating intermanual, symmetric gesture transfer on touchscreens. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '13, page 1119–1128, New York, NY, USA, 2013. Association for Computing Machinery.

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

[10] Ahmed Sabbir Arif and Ali Mazalek. Webtem: A web application to record text entry metrics. In *Proceedings of the 2016 ACM International Conference on Interactive Surfaces and Spaces*, ISS '16, page 415–420, New York, NY, USA, 2016. Association for Computing Machinery.

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

[12] Ahmed Sabbir Arif and Wolfgang Stuerzlinger. 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, page 383–392, New York, NY, USA, 2013. Association for Computing Machinery.

[13] Shiri Azenkot and Nicole B. Lee. Exploring the use of speech input by blind people on mobile devices. In *Proceedings of the 15th International ACM SIGACCESS Conference on Computers and Accessibility*, ASSETS '13, New York, NY, USA, 2013. Association for Computing Machinery.

[14] Shiri Azenkot, Jacob O. Wobbrock, Sanjana Prasain, and Richard E. Ladner. Input finger detection for nonvisual touch screen text entry in perkinput. In *Proceedings of Graphics Interface 2012*, GI '12, page 121–129, CAN, 2012. Canadian Information Processing Society.

[15] Shiri Azenkot and Shumin Zhai. 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, page 251–260, New York, NY, USA, 2012. Association for Computing Machinery.

[16] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv:1409.0473*, 2016.

[17] Saeideh Bakhshi, David A. Shamma, Lyndon Kennedy, Yale Song, Paloma de Juan, and Joseph 'Jofish' Kaye. Fast, cheap, and good: Why animated gifs engage us. In

Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems, page 575–586, New York, NY, USA, 2016. Association for Computing Machinery.

[18] Nikola Banovic, Ticha Sethapakdi, Yasasvi Hari, Anind K. Dey, and Jennifer Mankoff. 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, New York, NY, USA, 2019. Association for Computing Machinery.

[19] Francesco Barbieri, Miguel Ballesteros, Francesco Ronzano, and Horacio Saggion. Multimodal emoji prediction. *arXiv:1803.02392*, 2018.

[20] Francesco Barbieri, Jose Camacho-Collados, Francesco Ronzano, Luis Espinosa-Anke, Miguel Ballesteros, Valerio Basile, Viviana Patti, and Horacio Saggion. Semeval 2018 task 2: Multilingual emoji prediction. In *Proceedings of The 12th International Workshop on Semantic Evaluation*, pages 24–33, New Orleans, Louisiana, June 2018. Association for Computational Linguistics.

[21] Michel Beaudouin-Lafon. Designing interaction, not interfaces. In *Proceedings of the Working Conference on Advanced Visual Interfaces*, AVI '04, page 15–22, New York, NY, USA, 2004. Association for Computing Machinery.

[22] Erin Beneteau, Olivia K. Richards, Mingrui Zhang, Julie A. Kientz, Jason Yip, and Alexis Hiniker. Communication breakdowns between families and alexa. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, CHI '19, page 1–13, New York, NY, USA, 2019. Association for Computing Machinery.

[23] Hrvoje Benko, Andrew D. Wilson, and Patrick Baudisch. Precise selection techniques for multi-touch screens. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '06, page 1263–1272, New York, NY, USA, 2006. Association for Computing Machinery.

[24] Xiaojun Bi, Shiri Azenkot, Kurt Partridge, and Shumin Zhai. Octopus: Evaluating touchscreen keyboard correction and recognition algorithms via. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '13, page 543–552, New York, NY, USA, 2013. Association for Computing Machinery.

[25] Xiaojun Bi, Yang Li, and Shumin Zhai. Ffitts law: Modeling finger touch with fitts' law. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '13, page 1363–1372, New York, NY, USA, 2013. Association for Computing Machinery.

[26] Xiaojun Bi, Barton A. Smith, and Shumin Zhai. Multilingual touchscreen keyboard design and optimization. *Human–Computer Interaction*, 27(4):352–382, 2012.

[27] Xiaojun Bi and Shumin Zhai. *IJQwerty: What Difference Does One Key Change Make? Gesture Typing Keyboard Optimization Bounded by One Key Position Change from Qwerty*, page 49–58. Association for Computing Machinery, New York, NY, USA, 2016.

[28] Jeffrey P. Bigham, Chandrika Jayant, Hanjie Ji, Greg Little, Andrew Miller, Robert C. Miller, Robin Miller, Aubrey Tatarowicz, Brandyn White, Samual White, and Tom Yeh. Vizwiz: Nearly real-time answers to visual questions. In *Proceedings of the 23nd Annual ACM Symposium on User Interface Software and Technology*, UIST '10, page 333–342, New York, NY, USA, 2010. Association for Computing Machinery.

[29] John Brooke. Sus: a quick and dirty usability scale. *Usability evaluation in industry*, 1:189, 1996.

[30] John Brooke. Sus: A retrospective. *J. Usability Studies*, 8(2):29–40, February 2013.

[31] 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. Language models are few-shot learners. *arXiv:2005.14165*, 2020.

[32] Christopher Bryant and Hwee Tou Ng. How far are we from fully automatic high quality grammatical error correction? In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 697–707, Beijing, China, July 2015. Association for Computational Linguistics.

[33] Jeremy Burge. 5 billion emojis sent daily on messenger, 2017. `https://blog.emojipedia.org/5-billion-emojis-sent-daily-on-messenger/`.

[34] Anna C. Cavender, Daniel S. Otero, Jeffrey P. Bigham, and Richard E. Ladner. Aslstem forum: Enabling sign language to grow through online collaboration. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '10, page 2075–2078, New York, NY, USA, 2010. Association for Computing Machinery.

[35] Ciprian Chelba, Mohammad Norouzi, and Samy Bengio. N-gram language modeling using recurrent neural network estimation. *arXiv:1703.10724v2*, 1, 2017.

[36] 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. Gmail smart compose: Real-time assisted writing. *arXiv:1906.00080*, 2019.

[37] Xiang 'Anthony' Chen, Tovi Grossman, and George Fitzmaurice. 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, page 615–620, New York, NY, USA, 2014. Association for Computing Machinery.

[38] Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using

rnn encoder-decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar, October 2014. Association for Computational Linguistics.

[39] Yuri Choi, Kyung Hoon Hyun, and Ji-Hyun Lee. Image-based tactile emojis: Improved interpretation of message intention and subtle nuance for visually impaired individuals. *Human–Computer Interaction*, 35(1):40–69, 2020.

[40] Leigh Clark, Nadia Pantidi, Orla Cooney, Philip Doyle, Diego Garaialde, Justin Edwards, Brendan Spillane, Emer Gilmartin, Christine Murad, Cosmin Munteanu, Vincent Wade, and Benjamin R. Cowan. What makes a good conversation? challenges in designing truly conversational agents. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, CHI '19, page 1–12, New York, NY, USA, 2019. Association for Computing Machinery.

[41] Neil Cohn, Jan Engelen, and Joost Schilperoord. The grammar of emoji? constraints on communicative pictorial sequencing, 2019.

[42] Juliet Corbin and Anselm Strauss. Basics of qualitative research: Techniques and procedures for developing grounded theory, 2014.

[43] Sam Cox. World emoji day: making emojis more accessible, July 2019. `https://wearesocial.com/uk/blog/2019/07/world-emoji-day-making-emojis-more-accessible`.

[44] Henriette Cramer, Paloma de Juan, and Joel Tetreault. 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, page 504–509, New York, NY, USA, 2016. Association for Computing Machinery.

[45] E. R. F. W. Crossman. The speed and accuracy of simple hand movements. *The Nature and Acquisition of Industrial Skills.*, 1957.

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

[47] Wenzhe Cui, Suwen Zhu, Mingrui Ray Zhang, H. Andrew Schwartz, Jacob O. Wobbrock, and Xiaojun Bi. 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, page 487–499, New York, NY, USA, 2020. Association for Computing Machinery.

[48] Cristian Danescu-Niculescu-Mizil, Justin Cheng, Jon Kleinberg, and Lillian Lee. You had me at hello: How phrasing affects memorability. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 892–901, Jeju Island, Korea, July 2012. Association for Computational Linguistics.

[49] Florian Daniel, Pavel Kucherbaev, Cinzia Cappiello, Boualem Benatallah, and Mohammad Allahbakhsh. Quality control in crowdsourcing: A survey of quality attributes, assessment techniques, and assurance actions. *ACM Comput. Surv.*, 51(1), January 2018.

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

[51] Mark Davis and Peter Edberg. Unicode emoji, 2020. `https://unicode.org/reports/tr51/`.

[52] Nicola Dell, Vidya Vaidyanathan, Indrani Medhi, Edward Cutrell, and William Thies. "yours is better!": Participant response bias in hci. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '12, page 1321–1330, New York, NY, USA, 2012. Association for Computing Machinery.

[53] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv:1810.04805*, 2019.

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

[55] Steven Dow, Anand Kulkarni, Scott Klemmer, and Björn Hartmann. Shepherding the crowd yields better work. In *Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work*, CSCW '12, page 1013–1022, New York, NY, USA, 2012. Association for Computing Machinery.

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

[57] Instagram Engineering. Emojineering part 1: Machine learning for emoji trends, October 2016.

[58] Jason Eppink. A brief history of the gif (so far). *Journal of Visual Culture*, 13(3):298–306, 2014.

[59] Abigail Evans and Jacob Wobbrock. Taming wild behavior: The input observer for obtaining text entry and mouse pointing measures from everyday computer use. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '12, page 1947–1956, New York, NY, USA, 2012. Association for Computing Machinery.

[60] Jacqui Fashimpaur, Kenrick Kin, and Matt Longest. Pinchtype: Text entry for virtual and augmented reality using comfortable thumb to fingertip pinches. In *Extended*

Abstracts of the 2020 CHI Conference on Human Factors in Computing Systems, CHI EA '20, page 1–7, New York, NY, USA, 2020. Association for Computing Machinery.

[61] Anna Maria Feit and Antti Oulasvirta. Pianotext: Redesigning the piano keyboard for text entry. In *Proceedings of the 2014 Conference on Designing Interactive Systems*, DIS '14, page 1045–1054, New York, NY, USA, 2014. Association for Computing Machinery.

[62] Anna Maria Feit, Daryl Weir, and Antti Oulasvirta. How we type: Movement strategies and performance in everyday typing. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, CHI '16, page 4262–4273, New York, NY, USA, 2016. Association for Computing Machinery.

[63] Bjarke Felbo, Alan Mislove, Anders Søgaard, Iyad Rahwan, and Sune Lehmann. Using millions of emoji occurrences to learn any-domain representations for detecting sentiment, emotion and sarcasm. *arXiv preprint arXiv:1708.00524*, 1, 2017.

[64] Leah Findlater, Jon E. Froehlich, Kays Fattal, Jacob O. Wobbrock, and Tanya Dastyar. Age-related differences in performance with touchscreens compared to traditional mouse input. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '13, page 343–346, New York, NY, USA, 2013. Association for Computing Machinery.

[65] Leah Findlater and Jacob Wobbrock. Personalized input: Improving ten-finger touchscreen typing through automatic adaptation. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '12, page 815–824, New York, NY, USA, 2012. Association for Computing Machinery.

[66] Leah Findlater, Jacob O. Wobbrock, and Daniel Wigdor. 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, page 2453–2462, New York, NY, USA, 2011. Association for Computing Machinery.

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

[68] Paul M. Fitts and Bernard Radford. Information capacity of discrete motor responses under different cognitive sets. *Journal of experimental psychology*, 71 4:475–82, 1966.

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

[70] Andrew Fowler, Kurt Partridge, Ciprian Chelba, Xiaojun Bi, Tom Ouyang, and Shumin Zhai. 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, page 649–658, New York, NY, USA, 2015. Association for Computing Machinery.

[71] Brigitte N. Frederick. Fixed-, random-, and mixed-effects anova models: A user-friendly guide for increasing the generalizability of anova results. In *Advances in Social Science Methodology*, pages 111–122. Stamford, CT: JAI Press, 1999.

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

[73] Vittorio Fuccella and Benoît Martin. *TouchTap: A Gestural Technique to Edit Text on Multi-Touch Capable Mobile Devices*, pages 6–15. Association for Computing Machinery, New York, NY, USA, 2017.

[74] Cole Gleason, Amy Pavel, Himalini Gururaj, Kris Kitani, and Jeffrey Bigham. Making gifs accessible. In *The 22nd International ACM SIGACCESS Conference on Computers and Accessibility*, ASSETS '20, New York, NY, USA, 2020. Association for Computing Machinery.

[75] Cole Gleason, Amy Pavel, Xingyu Liu, Patrick Carrington, Lydia B. Chilton, and Jeffrey P. Bigham. Making memes accessible. In *The 21st International ACM SIGACCESS Conference on Computers and Accessibility*, ASSETS '19, page 367–376, New York, NY, USA, 2019. Association for Computing Machinery.

[76] Cole Gleason, Amy Pavel, Emma McCamey, Christina Low, Patrick Carrington, Kris M. Kitani, and Jeffrey P. Bigham. Twitter a11y: A browser extension to make twitter images accessible. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, CHI '20, page 1–12, New York, NY, USA, 2020. Association for Computing Machinery.

[77] Alec Go, Richa Bhayani, and Lei Huang. Twitter sentiment classification using distant supervision, 2009.

[78] Mayank Goel, Leah Findlater, and Jacob Wobbrock. 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, page 2687–2696, New York, NY, USA, 2012. Association for Computing Machinery.

[79] Mikael Goldstein, Robert Book, Gunilla Alsiö, and Silvia Tessa. Non-keyboard qwerty touch typing: A portable input interface for the mobile user. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '99, page 32–39, New York, NY, USA, 1999. Association for Computing Machinery.

[80] Jun Gong and Peter Tarasewich. A new error metric for text entry method evaluation. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '06, page 471–474, New York, NY, USA, 2006. Association for Computing Machinery.

[81] Jun Gong, Zheer Xu, Qifan Guo, Teddy Seyed, Xiang 'Anthony' Chen, Xiaojun Bi, and Xing-Dong Yang. 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, page 1–14, New York, NY, USA, 2018. Association for Computing Machinery.

[82] Joshua Goodman, Gina Venolia, Keith Steury, and Chauncey Parker. Language modeling for soft keyboards. In *Proceedings of the 7th International Conference on Intelligent User Interfaces*, IUI '02, page 194–195, New York, NY, USA, 2002. Association for Computing Machinery.

[83] Google. Gboard android play store page, 2022. `https://play.google.com/store/apps/details?id=com.google.android.inputmethod.latin`.

[84] Mitchell Gordon, Tom Ouyang, and Shumin Zhai. 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, page 3817–3821, New York, NY, USA, 2016. Association for Computing Machinery.

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

[86] Yizheng Gu, Chun Yu, Zhipeng Li, Zhaoheng Li, Xiaoying Wei, and Yuanchun Shi. Qwertyring: Text entry on physical surfaces using a ring. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, 2:1, March 2021.

[87] Darren Guinness, Edward Cutrell, and Meredith Ringel Morris. Caption crawler: Enabling reusable alternative text descriptions using reverse image search. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, page 1–11, New York, NY, USA, 2018. Association for Computing Machinery.

[88] Sandra G. Hart and Lowell E. Staveland. Development of nasa-tlx (task load index): Results of empirical and theoretical research. In Peter A. Hancock and Najmedin Meshkati, editors, *Human Mental Workload*, volume 52 of *Advances in Psychology*, pages 139 – 183. North-Holland, New York, 1988.

[89] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.

[90] Kenneth Heafield. Kenlm: Faster and smaller language model queries. In *Proceedings of the Sixth Workshop on Statistical Machine Translation*, pages 187–197, Edinburgh, Scotland, July 2011. Association for Computational Linguistics.

[91] Donald Hedeker and Robert D. Gibbons. A random-effects ordinal regression model for multilevel analysis. *Biometrics*, 50(4):933–944, 1994.

[92] Richard P. Heitz. The speed-accuracy tradeoff: history, physiology, methodology, and behavior. *Frontiers in Neuroscience*, 8, 2014.

[93] Amanda Hess and Quoctrung Bui. What love and sadness look like in 5 countries, according to their top gifs. *The New York Times*, 1(1),

December 2017. https://www.nytimes.com/interactive/2017/12/29/upshot/gifs-emotions-by-country.html.

[94] Sture Holm. A simple sequentially rejective multiple test procedure. *Scandinavian Journal of Statistics*, 6(2):65–70, 1979.

[95] Christian Holz and Patrick Baudisch. Understanding touch. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '11, page 2501–2510, New York, NY, USA, 2011. Association for Computing Machinery.

[96] Jonggi Hong, Seongkook Heo, Poika Isokoski, and Geehyuk Lee. 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, page 1233–1236, New York, NY, USA, 2015. Association for Computing Machinery.

[97] Eric Horvitz. Principles of mixed-initiative user interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '99, page 159–166, New York, NY, USA, 1999. Association for Computing Machinery.

[98] Albert H. Huang, David C. Yen, and Xiaoni Zhang. Exploring the potential effects of emoticons. *Inf. Manage.*, 45(7):466–473, November 2008.

[99] Apple Inc. ios dictation, Jul 2020. https://support.apple.com/en-us/HT208343.

[100] WHIRLSCAPE Inc. Dango your emoji assistant, 2016. https://getdango.com.

[101] Minal Jain, Sarita Seshagiri, and Simran Chopra. 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, page 767–774, New York, NY, USA, 2016. Association for Computing Machinery.

[102] Jialun "Aaron" Jiang, Jed R. Brubaker, and Casey Fiesler. Understanding diverse interpretations of animated gifs. In *Proceedings of the 2017 CHI Conference Extended Abstracts on Human Factors in Computing Systems*, CHI EA '17, page 1726–1732, New York, NY, USA, 2017. Association for Computing Machinery.

[103] Jialun "Aaron" Jiang, Casey Fiesler, and Jed R. Brubaker. "the perfect one": Understanding communication practices and challenges with animated gifs. *Proc. ACM Hum.-Comput. Interact.*, 2(CSCW), November 2018.

[104] B. E. John and A. Newell. Cumulating the science of hci: From s-r compatibility to transcription typing. *SIGCHI Bull.*, 20(SI):109–114, March 1989.

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

[106] Annika Kaltenhauser, Naundefineda Terzimehić, and Andreas Butz. Memeography: Understanding users through internet memes. In *Extended Abstracts of the 2021 CHI Conference on Human Factors in Computing Systems*, New York, NY, USA, 2021. Association for Computing Machinery.

[107] Shao kang Lo. The nonverbal communication functions of emoticons in computer-mediated communication. *Cyberpsychology & behavior : the impact of the Internet, multimedia and virtual reality on behavior and society*, 11 5:595–7, 2008.

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

[109] Slava Katz. 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):400–401, 1987.

[110] Ryan Kelly and Leon Watts. Characterising the inventive appropriation of emoji as relationally meaningful in mediated close personal relationships, 9 2015. Experiences of Technology Appropriation: Unanticipated Users, Usage, Circumstances, and Design ; Conference date: 20-09-2015 Through 20-09-2015.

[111] Joon-Gyum Kim, Taesik Gong, Evey Huang, Juho Kim, Sung-Ju Lee, Bogoan Kim, JaeYeon Park, Woojeong Kim, Kyungsik Han, and JeongGil Ko. Bringing context into emoji recommendations (poster). In *Proceedings of the 17th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '19, page 514–515, New York, NY, USA, 2019. Association for Computing Machinery.

[112] Yoon Kim, Yacine Jernite, David Sontag, and Alexander M. Rush. Character-aware neural language models. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, page 2741–2749. AAAI Press, 2016.

[113] Yoon Sang Kim, Byung Seok Soh, and Sang-Goog Lee. A new wearable input device: Scurry. *IEEE Transactions on Industrial Electronics*, 52(6):1490–1499, 2005.

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

[115] Masaki Kobayashi, H. Morita, Masaki Matsubara, N. Shimizu, and Atsuyuki Morishima. An empirical study on short- and long-term effects of self-correction in crowdsourced microtasks. In *HCOMP*, pages 79–87, Palo Alto, CA, USA, 2018. AAAI Press.

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

[117] Thomas Költringer, Poika Isokoski, and Thomas Grechenig. Twostick: Writing with a game controller. In *Proceedings of Graphics Interface 2007*, GI '07, page 103–110, New York, NY, USA, 2007. Association for Computing Machinery.

[118] Andreas Komninos, Mark Dunlop, Kyriakos Katsaris, and John Garofalakis. A glimpse of mobile text entry errors and corrective behaviour in the wild. In *Pro-*

ceedings of the 20th International Conference on Human-Computer Interaction with Mobile Devices and Services Adjunct, MobileHCI '18, page 221–228, New York, NY, USA, 2018. Association for Computing Machinery.

[119] Andreas Komninos, Emma Nicol, and Mark D. Dunlop. Designed with older adults to supportbetter error correction in smartphone text entry: The maxiekeyboard. In *Proceedings of the 17th International Conference on Human-Computer Interaction with Mobile Devices and Services Adjunct*, MobileHCI '15, page 797–802, New York, NY, USA, 2015. Association for Computing Machinery.

[120] Neal Krawetz. Looks like it - the hacker factor blog, 2011. `https://www.hackerfactor.com/blog/index.php?/archives/432-Looks-Like-It.html`.

[121] Per-Ola Kristensson and Shumin Zhai. 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, page 43–52, New York, NY, USA, 2004. Association for Computing Machinery.

[122] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations. *CoRR*, abs/1909.11942:1, 2019.

[123] Minkyung Lee and Woontack Woo. Arkb: 3d vision-based augmented reality keyboard. In *Online Proceeding of the 13th International Conference on Artificial Reality and Telexistence*, page 1, Keio University, Tokyo, Japan, 2003. IEEE.

[124] Luis A. Leiva, Alireza Sahami, Alejandro Catala, Niels Henze, and Albrecht Schmidt. Text entry on tiny qwerty soft keyboards. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, CHI '15, page 669–678, New York, NY, USA, 2015. Association for Computing Machinery.

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

[126] James R. Lewis. 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):425–428, 1999.

[127] Gideon Lewis-Kraus. The fasinatng ... fascinating history of autocorrect. *Wired*, 2014. `https://www.wired.com/2014/07/history-of-autocorrect/`.

[128] Yuncheng Li, Yale Song, Liangliang Cao, Joel Tetreault, Larry Goldberg, Alejandro Jaimes, and Jiebo Luo. Tgif: A new dataset and benchmark on animated gif description. *arXiv*, 2016.

[129] Eckhard Limpert, Werner A. Stahel, and Markus Abbt. Log-normal distributions across the sciences: Keys and clues: On the charms of statistics, and how mechanical models resembling gambling machines offer a link to a handy way to characterize

log-normal distributions, which can provide deeper insight into variability and probability—normal or log-normal: That is the question. *BioScience*, 51(5):341–352, 05 2001.

[130] RC Littell, PR Henry, and CB Ammerman. Statistical analysis of repeated measures data using sas procedures. *Journal of animal science*, 76(4):1216—1231, April 1998.

[131] Wanyu Liu, Olivier Rioul, Michel Beaudouin-Lafon, and Yves Guiard. Information-theoretic analysis of human performance for command selection. In Regina Bernhaupt, Girish Dalvi, Anirudha Joshi, Devanuj K. Balkrishan, Jacki O'Neill, and Marco Winckler, editors, *Human-Computer Interaction – INTERACT 2017*, pages 515–524, Cham, 2017. Springer International Publishing.

[132] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692:1, 2019.

[133] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv:1711.05101*, 2019.

[134] Kent Lyons, Thad Starner, Daniel Plaisted, James Fusia, Amanda Lyons, Aaron Drew, and E. W. Looney. Twiddler typing: One-handed chording text entry for mobile phones. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '04, page 671–678, New York, NY, USA, 2004. Association for Computing Machinery.

[135] Kelly Mack, Edward Cutrell, Bongshin Lee, and Meredith Ringel Morris. Designing tools for high-quality alt text authoring. In *The 23rd International ACM SIGACCESS Conference on Computers and Accessibility*, ASSETS '21, New York, NY, USA, 2021. Association for Computing Machinery.

[136] I. Scott MacKenzie. A note on calculating text entry speed, 2015. `https://www.yorku.ca/mack/RN-TextEntrySpeed.html`.

[137] I. Scott MacKenzie and Poika Isokoski. Fitts' throughput and the speed-accuracy tradeoff. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '08, page 1633–1636, New York, NY, USA, 2008. Association for Computing Machinery.

[138] I. Scott MacKenzie and R. William Soukoreff. A character-level error analysis technique for evaluating text entry methods. In *Proceedings of the Second Nordic Conference on Human-Computer Interaction*, NordiCHI '02, page 243–246, New York, NY, USA, 2002. Association for Computing Machinery.

[139] I. Scott MacKenzie and R. William Soukoreff. Text entry for mobile computing: Models and methods,theory and practice. *Human–Computer Interaction*, 17(2-3):147–198, 2002.

[140] I. Scott MacKenzie and R. William Soukoreff. Phrase sets for evaluating text entry techniques. In *CHI '03 Extended Abstracts on Human Factors in Computing Systems*,

CHI EA '03, page 754–755, New York, NY, USA, 2003. Association for Computing Machinery.

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

[142] I. Scott MacKenzie and Shawn X. Zhang. The design and evaluation of a high-performance soft keyboard. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '99, page 25–31, New York, NY, USA, 1999. Association for Computing Machinery.

[143] Catherine Macleod, Nancy Ide, and Ralph Grishman. The american national corpus: A standardized resource for american english. In *Proceedings of the Second International Conference on Language Resources and Evaluation (LREC'00)*, page 1, Athens, Greece, May 2000. European Language Resources Association (ELRA).

[144] Haley MacLeod, Cynthia L. Bennett, Meredith Ringel Morris, and Edward Cutrell. Understanding blind people's experiences with computer-generated captions of social media images. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, CHI '17, page 5988–5999, New York, NY, USA, 2017. Association for Computing Machinery.

[145] Edgar Matias, I. Scott MacKenzie, and William Buxton. One-handed touch typing on a qwerty keyboard. *Hum.-Comput. Interact.*, 11(1):1–27, March 1996.

[146] Charles E. McCulloch and John M. Neuhaus. *Generalized Linear Mixed Models*, chapter 1, page 1. American Cancer Society, New York, 2005.

[147] Nora McDonald, Sarita Schoenebeck, and Andrea Forte. Reliability and inter-rater reliability in qualitative research: Norms and guidelines for cscw and hci practice. *Proc. ACM Hum.-Comput. Interact.*, 3(CSCW), November 2019.

[148] MDN contributors. Mdn web docs - aria, 2020. `https://developer.mozilla.org/en-US/docs/Web/Accessibility/ARIA`.

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

[150] Microsoft. Swiftkey homepage, 2020. `https://www.microsoft.com/en-us/swiftkey`.

[151] Kate M. Miltner and Tim Highfield. Never gonna gif you up: Analyzing the cultural significance of the animated gif. *Social Media + Society*, 3(3):2056305117725223, 2017.

[152] Aarthi Easwara Moorthy and Kim-Phuong L. Vu. Privacy concerns for use of voice activated personal assistant in the public space. *International Journal of Human–Computer Interaction*, 31(4):307–335, 2015.

[153] Valerie S. Morash, Yue-Ting Siu, Joshua A. Miele, Lucia Hasty, and Steven Landau.

Guiding novice web workers in making image descriptions using templates. *ACM Trans. Access. Comput.*, 7(4), November 2015.

[154] Meredith Ringel Morris, Jazette Johnson, Cynthia L. Bennett, and Edward Cutrell. Rich representations of visual content for screen reader users. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, page 1–11, New York, NY, USA, 2018. Association for Computing Machinery.

[155] Meredith Ringel Morris, Annuska Zolyomi, Catherine Yao, Sina Bahram, Jeffrey P. Bigham, and Shaun K. Kane. "with most of it being pictures now, i rarely use it": Understanding twitter's evolving accessibility to blind users. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, CHI '16, page 5506–5516, New York, NY, USA, 2016. Association for Computing Machinery.

[156] Martez E. Mott, Radu-Daniel Vatavu, Shaun K. Kane, and Jacob O. Wobbrock. Smart touch: Improving touch accuracy for people with motor impairments with template matching. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, CHI '16, page 1934–1946, New York, NY, USA, 2016. Association for Computing Machinery.

[157] Eugene W. Myers. An o(nd) difference algorithm and its variations. *Algorithmica*, 1:251–266, 1986.

[158] Pandu Nayak. Understanding searches better than ever before, October 2019. `https://blog.google/products/search/search-language-understanding-bert/`.

[159] Saul B. Needleman and Christian D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48(3):443–453, 1970.

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

[161] Hwee Tou Ng, Siew Mei Wu, Ted Briscoe, Christian Hadiwinoto, Raymond Hendy Susanto, and Christopher Bryant. The conll-2014 shared task on grammatical error correction. In *Proceedings of the Eighteenth Conference on Computational Natural Language Learning: Shared Task*, pages 1–14, Baltimore, Maryland, June 2014. Association for Computational Linguistics.

[162] Tao Ni, Doug Bowman, and Chris North. Airstroke: Bringing unistroke text entry to freehand gesture interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '11, page 2473–2476, New York, NY, USA, 2011. Association for Computing Machinery.

[163] Donald A. Norman. How might people interact with agents. *Commun. ACM*, 37(7):68–71, jul 1994.

[164] Nielsen Norman. 10 usability heuristics for user interface design, 2020. `https://www.nngroup.com/articles/ten-usability-heuristics/`.

[165] Stephen Oney, Chris Harrison, Amy Ogan, and Jason Wiese. 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, page 2799–2802, New York, NY, USA, 2013. Association for Computing Machinery.

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

[167] Tim Paek and Bo-June (Paul) Hsu. Sampling representative phrase sets for text entry experiments: A procedure and public resource. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '11, page 2477–2480, New York, NY, USA, 2011. Association for Computing Machinery.

[168] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[169] Amy Pavel, Gabriel Reyes, and Jeffrey P. Bigham. *Rescribe: Authoring and Automatically Editing Audio Descriptions*, page 747–759. Association for Computing Machinery, New York, NY, USA, 2020.

[170] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar, October 2014. Association for Computational Linguistics.

[171] Paraskevas Petsanas, Christos Sintoris, and Nikolaos Avouris. Dealing with input mode confusion during dual-language keyboard use. In Carmelo Ardito, Rosa Lanzilotti, Alessio Malizia, Helen Petrie, Antonio Piccinno, Giuseppe Desolda, and Kori Inkpen, editors, *Human-Computer Interaction – INTERACT 2021*, pages 20–28, Cham, 2021. Springer International Publishing.

[172] Henning Pohl, Christian Domin, and Michael Rohs. Beyond just text: Semantic emoji similarity modeling to support expressive communication 4 6 4 2 6 0 3 . *ACM Trans. Comput.-Hum. Interact.*, 24(1), March 2017.

[173] K R Prajwal, C V Jawahar, and Ponnurangam Kumaraguru. Towards increased accessibility of meme images with the help of rich face emotion captions. In *Proceedings of the 27th ACM International Conference on Multimedia*, MM '19, page 202–210, New York, NY, USA, 2019. Association for Computing Machinery.

[174] Ryan Qin, Suwen Zhu, Yu-Hao Lin, Yu-Jung Ko, and Xiaojun Bi. Optimal-t9: An optimized t9-like keyboard for small touchscreen devices. In *Proceedings of the 2018*

ACM International Conference on Interactive Surfaces and Spaces, ISS '18, page 137–146, New York, NY, USA, 2018. Association for Computing Machinery.

[175] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI blog*, 2019.

[176] Radim Řehůřek and Petr Sojka. Software framework for topic modelling with large corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta, May 2010. ELRA. `http://is.muni.cz/publication/884893/en`.

[177] Shyam Reyal, Shumin Zhai, and Per Ola Kristensson. Performance and user experience of touchscreen and gesture keyboards in a lab setting and in the wild. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, CHI '15, page 679–688, New York, NY, USA, 2015. Association for Computing Machinery.

[178] Mark Richardson, Matt Durasoff, and Robert Wang. Decoding surface touch typing from hand-tracking. In *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology*, UIST '20, page 686–696, New York, NY, USA, 2020. Association for Computing Machinery.

[179] David Rodrigues, Diniz Lopes, Marília Prada, Dominic Thompson, and Margarida V. Garrido. A frown emoji can be worth a thousand words: Perceptions of emoji use in text messages exchanged between romantic partners. *Telematics and Informatics*, 34(8):1532 – 1543, 2017.

[180] Helena Roeber, John Bacus, and Carlo Tomasi. Typing in thin air: The canesta projection keyboard - a new method of interaction with electronic devices. In *CHI '03 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '03, page 712–713, New York, NY, USA, 2003. Association for Computing Machinery.

[181] Sherry Ruan, Jacob O. Wobbrock, Kenny Liou, Andrew Ng, and James A. Landay. Comparing speech and keyboard text entry for short messages in two languages on touchscreen phones. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, 1(4), January 2018.

[182] Susana Rubio, E. M. Díaz, J. Martín, and J. Puente. Evaluation of subjective mental workload: A comparison of swat, nasa-tlx, and workload profile methods. *Applied Psychology*, 53:61–86, 2004.

[183] Alexander M. Rush, Sumit Chopra, and Jason Weston. A neural attention model for abstractive sentence summarization. *arXiv:1509.00685*, 2015.

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

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

[186] Timothy Salthouse. Perceptual, cognitive, and motoric aspects of transcription typing. *Psychological bulletin*, 99:303–19, 06 1986.

[187] Dominik Schmidt, Florian Block, and Hans Gellersen. A comparison of direct and indirect multi-touch input for large surfaces. In Tom Gross, Jan Gulliksen, Paula Kotzé, Lars Oestreicher, Philippe Palanque, Raquel Oliveira Prates, and Marco Winckler, editors, *Human-Computer Interaction – INTERACT 2009*, pages 582–594, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[188] Raymond Scupin. The kj method: A technique for analyzing data derived from japanese ethnology. *Human Organization*, 56(2):233–237, 1997.

[189] A. Sears and B. Shneiderman. High precision touchscreens: Design strategies and comparisons with a mouse. *Int. J. Man Mach. Stud.*, 34:593–613, 1991.

[190] L. H. Shaffer. Timing in the motor programming of typing. *Quarterly Journal of Experimental Psychology*, 30:333 – 345, 1978.

[191] C. E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423, 1948.

[192] Ashley Sheridan. Emoji and accessibility, 2019. `http://www.ashleysheridan.co.uk/blog/Emoji+and+Accessibility`.

[193] Weinan Shi, Chun Yu, Xin Yi, Zhen Li, and Yuanchun Shi. Toast: Ten-finger eyes-free typing on touchable surfaces. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, 2(1), March 2018.

[194] Rachel N. Simons, Danna Gurari, and Kenneth R. Fleischmann. "i hope this is helpful": Understanding crowdworkers' challenges and motivations for an image description task. *Proc. ACM Hum.-Comput. Interact.*, 4(CSCW2), October 2020.

[195] Shyamli Sindhwani, Christof Lutteroth, and Gerald Weber. Retype: Quick text editing with keyboard and gaze. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, CHI '19, page 1–13, New York, NY, USA, 2019. Association for Computing Machinery.

[196] Brian A. Smith, Xiaojun Bi, and Shumin Zhai. Optimizing touchscreen keyboards for gesture typing. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, CHI '15, page 3365–3374, New York, NY, USA, 2015. Association for Computing Machinery.

[197] Xavier Snelgrove. Teaching robots to feel: Emoji & deep learning, 2016. `https://getdango.com/emoji-and-deep-learning/`.

[198] R. W. Soukoreff and I. S. MacKenzie. An informatic rationale for the speed-accuracy trade-off. In *2009 IEEE International Conference on Systems, Man and Cybernetics*, number 1 in 1, pages 2890–2896, 2009.

[199] R. William Soukoreff. *Quantifying Text Entry Performance*. York University, 2010.

[200] R. William Soukoreff and I. Scott MacKenzie. 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, page 319–320, New York, NY, USA, 2001. Association for Computing Machinery.

[201] R. William Soukoreff and I. Scott MacKenzie. 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, page 113–120, New York, NY, USA, 2003. Association for Computing Machinery.

[202] R. William Soukoreff and I. Scott MacKenzie. Recent developments in text-entry error rate measurement. In *CHI '04 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '04, page 1425–1428, New York, NY, USA, 2004. Association for Computing Machinery.

[203] R. William Soukoreff and Ian Scott MacKenzie. Theoretical upper and lower bounds on typing speed using a stylus and a soft keyboard. *Behav. Inf. Technol.*, 14:370–379, 1995.

[204] R. William Soukoreff and Ian Scott MacKenzie. Towards a standard for pointing device evaluation, perspectives on 27 years of fitts' law research in hci. *International Journal of Human-Computer Studies*, 61:751–789, 2004.

[205] Caleb Southern, James Clawson, Brian Frey, Gregory Abowd, and Mario Romero. 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, page 317–326, New York, NY, USA, 2012. Association for Computing Machinery.

[206] Todd Spangler. Giphy video launches: Gif-sharing platform adds audiovisual clips from universal pictures, other partners, December 2019. `https://variety.com/2019/digital/news/giphy-video-launch-universal-pictures-1203430165/`.

[207] Srinath Sridhar, Anna Maria Feit, Christian Theobalt, and Antti Oulasvirta. 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, page 3643–3652, New York, NY, USA, 2015. Association for Computing Machinery.

[208] Abigale Stangl, Meredith Ringel Morris, and Danna Gurari. "person, shoes, tree. is the person naked?" what people with vision impairments want in image descriptions. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, page 1–13, New York, NY, USA, 2020. Association for Computing Machinery.

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

[210] SwiftKey. Swiftkey emoji report, 2015. `https://www.scribd.com/doc/262594751/SwiftKey-Emoji-Report`.

[211] Ryo Takahashi, Masaaki Fukumoto, Changyo Han, Takuya Sasatani, Yoshiaki Narusue, and Yoshihiro Kawahara. Telemetring: A batteryless and wireless ring-shaped keyboard using passive inductive telemetry. In *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology*, UIST '20, page 1161–1168, New York, NY, USA, 2020. Association for Computing Machinery.

[212] TechCrunch. Tenor hits 12b gif searches every month, 2018. `https://social.techcrunch.com/2018/02/20/tenor-hits-12b-searches-in-its-gif-keyboard-every-month/`.

[213] David R. Thomas. A general inductive approach for analyzing qualitative evaluation data. *American Journal of Evaluation*, 27(2):237–246, 2006.

[214] Garreth W. Tigwell and David R. Flatla. Oh that's what you meant! reducing emoji misunderstanding. In *Proceedings of the 18th International Conference on Human-Computer Interaction with Mobile Devices and Services Adjunct*, MobileHCI '16, page 859–866, New York, NY, USA, 2016. Association for Computing Machinery.

[215] Garreth W. Tigwell, Benjamin M. Gorman, and Rachel Menzies. *Emoji Accessibility for Visually Impaired People*, page 1–14. Association for Computing Machinery, New York, NY, USA, 2020.

[216] Tornadoweb. Tornado web server, 2020. `https://www.tornadoweb.org/en/stable/`.

[217] Linda Klebe Trevino and Jane Webster. Flow in computer-mediated communication: Electronic mail and voice mail evaluation and impacts. *Communication Research*, 19(5):539–573, October 1992.

[218] Twitter. How to make images accessible for people, 2021. `https://help.twitter.com/en/using-twitter/picture-descriptions`.

[219] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *arXiv:1706.03762*, 2017.

[220] Dan Venolia and Forrest Neiberg. T-cube: A fast, self-disclosing pen-based alphabet. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '94, page 265–270, New York, NY, USA, 1994. Association for Computing Machinery.

[221] Veroniiiica. How do people with vision impairments use emoji?, Jul 2018. `https://www.perkinselearning.org/technology/blog/how-do-people-vision-impairments-use-emoji`.

[222] Keith Vertanen, Crystal Fletcher, Dylan Gaines, Jacob Gould, and Per Ola Kristensson. 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*, page 1–12, New York, NY, USA, 2018. Association for Computing Machinery.

[223] Keith Vertanen, Dylan Gaines, Crystal Fletcher, Alex M. Stanage, Robbie Watling, and Per Ola Kristensson. 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, page 1–14, New York, NY, USA, 2019. Association for Computing Machinery.

[224] Keith Vertanen and Per Ola Kristensson. 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, page 295–298, New York, NY, USA, 2011. Association for Computing Machinery.

[225] Keith Vertanen and Per Ola Kristensson. Complementing text entry evaluations with a composition task. *ACM Trans. Comput.-Hum. Interact.*, 21(2), February 2014.

[226] Keith Vertanen, Haythem Memmi, Justin Emge, Shyam Reyal, and Per Ola Kristensson. 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, page 659–668, New York, NY, USA, 2015. Association for Computing Machinery.

[227] Daniel Vogel and Patrick Baudisch. Shift: A technique for operating pen-based interfaces using touch. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '07, page 657–666, New York, NY, USA, 2007. Association for Computing Machinery.

[228] W3Techs. Usage statistics of gif for websites, august 2021, 2021. `https://w3techs.com/technologies/details/im-gif`.

[229] Will Walmsley. Exploring emoji: The quest for the perfect emoticon, February 2015. `http://minuum.com/exploring-emoji-the-quest-for-the-perfect-emoticon/`.

[230] Joseph B. Walther and Kyle P. D'Addario. The impacts of emoticons on message interpretation in computer-mediated communication. *Social Science Computer Review*, 19:324–347, 2001.

[231] Ruolin Wang, Zixuan Chen, Mingrui Ray Zhang, Zhaoheng Li, Zhixiu Liu, Zihan Dang, Chun Yu, and Xiang 'Anthony' Chen. Revamp: Enhancing accessible information seeking experience of online shopping for blind or low vision users. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, New York, NY, USA, 2021. Association for Computing Machinery.

[232] Ruolin Wang, Chun Yu, Xing-Dong Yang, Weijie He, and Yuanchun Shi. Eartouch: Facilitating smartphone use for visually impaired people in mobile and public scenarios. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, CHI '19, page 1–13, New York, NY, USA, 2019. Association for Computing Machinery.

[233] Yuan Wang, Yukun Li, Xinning Gui, Yubo Kou, and Fenglian Liu. Culturally-embedded visual literacy: A study of impression management via emoticon, emoji,

sticker, and meme on social media in china. *Proc. ACM Hum.-Comput. Interact.*, 3(CSCW), November 2019.

[234] David J. Ward, Alan F. Blackwell, and David J. C. MacKay. 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, page 129–137, New York, NY, USA, 2000. Association for Computing Machinery.

[235] Léonie Watson. Accessible emoji, December 2016. `https://tink.uk/accessible-emoji/`.

[236] Daryl Weir, Henning Pohl, Simon Rogers, Keith Vertanen, and Per Ola Kristensson. Uncertain text entry on mobile devices. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '14, page 2307–2316, New York, NY, USA, 2014. Association for Computing Machinery.

[237] Mark Weiser. The computer for the 21st century. *SIGMOBILE Mob. Comput. Commun. Rev.*, 3(3):3–11, July 1999.

[238] A. Welford. *Fundamentals of skill*. Methuen, London, England, 1968.

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

[240] Andrew D. Wilson and Maneesh Agrawala. Text entry using a dual joystick game controller. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '06, page 475–478, New York, NY, USA, 2006. Association for Computing Machinery.

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

[242] Jacob Wobbrock and Brad Myers. Trackball text entry for people with motor impairments. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '06, page 479–488, New York, NY, USA, 2006. Association for Computing Machinery.

[243] Jacob O. Wobbrock. Situationally aware mobile devices for overcoming situational impairments. In *Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, EICS '19, New York, NY, USA, 2019. Association for Computing Machinery.

[244] Jacob O. Wobbrock, Leah Findlater, Darren Gergle, and James J. Higgins. The aligned rank transform for nonparametric factorial analyses using only anova procedures. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '11, page 143–146, New York, NY, USA, 2011. Association for Computing Machinery.

[245] Jacob O. Wobbrock, Krzysztof Z. Gajos, Shaun K. Kane, and Gregg C. Vanderheiden. Ability-based design. *Commun. ACM*, 61(6):62–71, May 2018.

[246] Jacob O. Wobbrock, Shaun K. Kane, Krzysztof Z. Gajos, Susumu Harada, and Jon Froehlich. Ability-based design: Concept, principles and examples. *ACM Trans. Access. Comput.*, 3(3), April 2011.

[247] Jacob O. Wobbrock and Brad A. Myers. Analyzing the input stream for character-level errors in unconstrained text entry evaluations. *ACM Trans. Comput.-Hum. Interact.*, 13(4):458–489, December 2006.

[248] Jacob O. Wobbrock, Brad A. Myers, and John A. Kembel. 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, page 61–70, New York, NY, USA, 2003. Association for Computing Machinery.

[249] 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. Huggingface's transformers: State-of-the-art natural language processing. *ArXiv*, abs/1910.03771:1, 2019.

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

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

[252] Ziang Xie, Anand Avati, Naveen Arivazhagan, Dan Jurafsky, and Andrew Y. Ng. Neural language correction with character-based attention, 2016.

[253] Zheer Xu, Weihao Chen, Dongyang Zhao, Jiehui Luo, Te-Yen Wu, Jun Gong, Sicheng Yin, Jialun Zhai, and Xing-Dong Yang. 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, page 1–13, New York, NY, USA, 2020. Association for Computing Machinery.

[254] 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. 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, page 883–899, New York, NY, USA, 2019. Association for Computing Machinery.

[255] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. Xlnet: Generalized autoregressive pretraining for language understanding. *arXiv:1906.08237*, 2020.

[256] Richard Yao. The surging popularity of gifs in digital culture, April 2018.

[257] Xin Yi, Chen Wang, Xiaojun Bi, and Yuanchun Shi. 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, page 1–13, New York, NY, USA, 2020. Association for Computing Machinery.

[258] Xin Yi, Chun Yu, Weijie Xu, Xiaojun Bi, and Yuanchun Shi. Compass: Rotational keyboard on non-touch smartwatches. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, CHI '17, page 705–715, New York, NY, USA, 2017. Association for Computing Machinery.

[259] Xin Yi, Chun Yu, Mingrui Zhang, Sida Gao, Ke Sun, and Yuanchun Shi. Atk: Enabling ten-finger freehand typing in air based on 3d hand tracking data. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology*, UIST '15, page 539–548, New York, NY, USA, 2015. Association for Computing Machinery.

[260] Ying Yin, Tom Yu Ouyang, Kurt Partridge, and Shumin Zhai. 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, page 2775–2784, New York, NY, USA, 2013. Association for Computing Machinery.

[261] Adams Wei Yu, David Dohan, Minh-Thang Luong, Rui Zhao, Kai Chen, Mohammad Norouzi, and Quoc V. Le. Qanet: Combining local convolution with global self-attention for reading comprehension. *arXiv:1804.09541*, 2018.

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

[263] Difeng Yu, K. Fan, H. Zhang, Diego Monteiro, Wenge Xu, and H. Liang. Pizzatext: Text entry for virtual reality systems using dual thumbsticks. *IEEE Transactions on Visualization and Computer Graphics*, 24:2927–2935, 2018.

[264] Torsten Zesch. 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*, pages 529–538, Avignon, France, April 2012. Association for Computational Linguistics.

[265] Shumin Zhai, Jing Kong, and Xiangshi Ren. Speed-accuracy tradeoff in fitts' law tasks: On the equivalency of actual and nominal pointing precision. *Int. J. Hum.-Comput. Stud.*, 61(6):823–856, dec 2004.

[266] Shumin Zhai, Jing Kong, and Xiangshi Ren. Speed–accuracy tradeoff in fitts' law tasks—on the equivalency of actual and nominal pointing precision. *International*

Journal of Human-Computer Studies, 61(6):823–856, 2004. Fitts' law 50 years later: applications and contributions from human-computer interaction.

[267] Shumin Zhai and Per Ola Kristensson. The word-gesture keyboard: Reimagining keyboard interaction. *Commun. ACM*, 55(9):91–101, September 2012.

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

[269] Mingrui "Ray" Zhang, Alex Mariakakis, Jacob Burke, and Jacob O. Wobbrock. A comparative study of lexical and semantic emoji suggestion systems. In Katharina Toeppe, Hui Yan, and Samuel Kai Wah Chu, editors, *Diversity, Divergence, Dialogue*, pages 229–247, Cham, 2021. Springer International Publishing.

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

[271] Mingrui Ray Zhang, He Wen, and Jacob O. Wobbrock. 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, page 843–855, New York, NY, USA, 2019. Association for Computing Machinery.

[272] Mingrui Ray Zhang and Jacob O. Wobbrock. 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, page 831–842, New York, NY, USA, 2019. Association for Computing Machinery.

[273] Mingrui Ray Zhang and Shumin Zhai. Phraseflow: Designs and empirical studies of phrase-level input. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, CHI '21, New York, NY, USA, 2021. Association for Computing Machinery.

[274] Mingrui Ray Zhang, Shumin Zhai, and Jacob O. Wobbrock. 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*, CHI '19, page 1–13, New York, NY, USA, 2019. Association for Computing Machinery.

[275] Mingrui Ray Zhang, Shumin Zhai, and Jacob O. Wobbrock. Typeanywhere: A qwerty-based text entry solution for ubiquitous computing. In *CHI Conference on Human Factors in Computing Systems*, CHI '22, New York, NY, USA, 2022. Association for Computing Machinery.

[276] Mingrui Ray Zhang, Mingyuan Zhong, and Jacob O. Wobbrock. Ga11y: An automated gif annotation system for visually impaired users. In *CHI Conference on Human Factors in Computing Systems*, CHI '22, New York, NY, USA, 2022. Association for Computing Machinery.

[277] Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'15, page 649–657, Cambridge, MA, USA, 2015. MIT Press.

[278] Xiaoyi Zhang, Anne Spencer Ross, Anat Caspi, James Fogarty, and Jacob O. Wobbrock. Interaction proxies for runtime repair and enhancement of mobile application accessibility. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, page 6024–6037, New York, NY, USA, 2017. Association for Computing Machinery.

[279] Rui Zhou, Jasmine Hentschel, and Neha Kumar. Goodbye text, hello emoji: Mobile communication on wechat in china. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, page 748–759, New York, NY, USA, 2017. Association for Computing Machinery.

[280] Suwen Zhu, Tianyao Luo, Xiaojun Bi, and Shumin Zhai. Typing on an invisible keyboard. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, CHI '18, page 1–13, New York, NY, USA, 2018. Association for Computing Machinery.

Appendix A

APPENDIX FOR TYPE, THEN CORRECT

Table A.1: Example phrases of different error types used in the experiment. Error text is highlighted in bold; insertion errors are represented by parentheses. Correct text is provided at the end of each phrase.

	Near-error	**Far-error**
Typos	The season would end the **net** week. //**next**	Untreated septicemic plague is universally fatal, but early treatment with antibiotics reduces the **morality** rate to between 4 and 15 percent. //**mortality**
Word change	I suggest you get facts **after** judging anyone. //**before**	This book is **very** touching. It tells Dorie's story of all the unbelievably horrible things while growing up. //**quite**
Insertion	Where do you want to meet to walk **()** there? //**over**	If you're just waiting for Vol 2, why **()** you buy this in the first place? //**did**

Appendix B

APPENDIX FOR VOICEMOJI

Table B.1: Phrase Set For the Emoji Entry Session

	Emoji Description	**English Phrases**
1	Hamburger	Are you going to join us for lunch 🍔
2	Folded hands	Keep me posted please! 🙏
3	Crying face	I am so sorry 😢
4	Sun	Florida is great ☀️
5	Ear	What do you hear 👂
6	Wrapped gift	A 🎁 isn't necessary
7	Musical note	He would love anything about rock 🎵
8	School	And leave me at 🏫 alone
9	Airplane	I'm on a ✈️
10	Hundred points	I think that answer is 💯
11	Smiling face with open mouth	😃 How are you
12	Raised fist	✊ I am trying again
13	Disappointed face	😟 We are all fragile
14	Eyes	👀 See you soon
15	OK hand sign	👌 I'll get you one.
16	Face with tears of joy	😂
17	Red heart	❤️
18	Smiling face with heart-shaped eyes	😍

19	Loudly crying face	
20	Thumb up	
21	Face with hand over mouth	
22	Face with rolling eyes	
23	Probing cane	
24	Cat face with tears of joy	
25	Woman gesturing no	

Table B.2: Phrase Set For the Emoji Suggestion Session

	Sentiment	**English Phrases**
1	Negative	Can't sleep. my tooth is aching.
2	Negative	So tired, I didn't sleep well at all last night.
3	Negative	Math review, I'm going to fail the exam.
4	Negative	Still hungry after eating
5	Positive	My dentist appointment today was quite enjoyable.
6	Positive	I'm moving to a new place
7	Positive	Just listened to a new song and It was good
8	Positive	Happy birthday to you

Appendix C

APPENDIX FOR GA11Y

C.1 Sample Annotations Collected from Mturk Workers

Annotations for Fig C.1 from different task designs:

Freeform Thor is facing the camera and is winking and smiling as the camera pans in. Thor is wearing his signature red cape and his hair is long and flowing to his shoulders.. Usage Scenarios: It can be used to express a sense of a shared feeling or a secret being shared between two people.

Semi-structured Chris Hemsworth (Thor) is smiling widely with his teeth as the camera zooms in on his face, he winks.. Usage Scenarios: This can be used for a situation where someone might want to flirt and express the wink to the person that they're sending it to,

Structured Character: Thor, a Greek God. Doing: Thor is winking his eye. Expression: Joy or an inside joke. Activity: Thor is expressing happiness and smiling. Thor is played by Chris Hemsworth and this character is from the film series the Avengers by Marvel.. The gif is a happy moment of Thor smiling and winking. Usage Scenarios: People might use this gif to express that they understand an inside joke

C.2 Sample Machine-Generated Annotations

The machine generated annotations are described as follows for Figure C.1. There are many misrecognized objects and scenarios.

(f) "Automatically generated caption is a close-up of a person's face. Best guess is lip. The labels are Glasses and Vision care and Eyelash";

(g) "Automatically generated caption is logo, company name. Best guess is metal wheels.

Figure C.1: GIFs for the annotation evaluation study with BLV users. **(a)** A cartoon eye opening; **(b)** a clip from the movie *thor*; **(c)** a sticker of light bulb; **(d)** a clip from an old movie; **(e)** a cartoon clip with a speedy driving meme

The labels are Font and Cylinder and Rectangle. The text in the image is LOOKING FOR IDEAS";

(h) "Automatically generated caption is a yellow and green balloon. Best guess is spongebob quarantine meme. The labels are Eye and Smile and Cartoon";

(i) "Automatically generated caption is a person standing on a dirt hill. Best guess is soil. The labels are Sky and Asphalt and Grass. The text in the image is GIF";

(j) "Automatically generated caption is a cartoon of a dog. Best guess is cartoon. The labels are Hair and Head and Cartoon. The text in the image is GOOD NIGHT".

Appendix D

APPENDIX FOR THROUGHPUT

Table D.1: English letter frequencies used in this paper, adapted from `http://www.macfreek.nl/memory/Letter_Distribution`. We included SPACE and normalized frequencies to sum to 1.000.

Letter i	**p(i)**	**Letter i**	**p(i)**
a	0. 06545420428810268	b	0. 012614349400134882
c	0. 022382079660795914	d	0. 032895839710101495
e	0. 10287480840814522	f	0. 019870906945619955
g	0. 01628201251975626	h	0.0498866519336527
i	0.05679944220647908	j	0.0009771967640664421
k	0.005621008826086285	l	0.03324279082953061
m	0.020306796250368523	n	0.057236004874678816
o	0.061720746945911634	p	0.015073764715016882
q	0.0008384527300266635	r	0.049980287430261394
s	0.05327793252372975	t	0.07532249847431097
u	0.022804128240333354	v	0.007977317166161044
w	0.017073508770571122	x	0.0014120607927983009
y	0.014305632773116854	z	0.0005138874382474097
Space	0.18325568938199557	TOTAL	1.00000