GESTURE-EFFICIENT GLOBAL LANGUAGE INPUT METHODS

By

Daniel Mailman

Dalei Wu
Associate Professor of Computer Science
(Chair)

Cuilan Gao
Associate Professor of Mathematics
(Committee Member)

Yingfeng Wang
Associate Professor of Computer Science
(Committee Member)

Yu Liang
Professor of Computer Science
(Committee Member)

GESTURE-EFFICIENT GLOBAL LANGUAGE INPUT METHODS

By

Daniel Mailman

A Dissertation Submitted to the Faculty of the University of
Tennessee at Chattanooga in Partial
Fulfillment of the Requirements of the Degree
of Doctor of Philosophy

The University of Tennessee at Chattanooga
Chattanooga, Tennessee

May 2026

ABSTRACT

A main task of computer hardware and software is **lexical input**: sending elements of human language to applications such as word processors and spreadsheets. For many input sets, however, standard QWERTY keyboards do not provide efficient access, especially when the target material includes non-QWERTY characters, words, or phrases.

This dissertation presents ***μLex***, a general method for simplifying such input by assigning supplemental semantics to ordinary QWERTY keys. A central design choice is that these supplemental semantics are accessed through key-down duration rather than through key combinations, layout switching, or dead-key composition.

In simple cases, such as Spanish letters and punctuation, this method provides direct access to target characters with little need for instructional display. In more complex cases, displays indicate available choices and guide users through sequences leading to multiply mapped characters, words, and phrases. The dissertation develops this approach inductively, from simple 1-to-1 character mappings to more complex lexica, including pan-European characters, Vietnamese, the International Phonetic Alphabet (IPA), and Chinese.

The dissertation argues that gesture-efficient input methods for such lexica can be generated algorithmically from character- and phrase-frequency data. Efficiency is evaluated in gestures per

lexeme (GPL), where the denominator lexeme is the basic unit of lexical input. The result is a design and evaluation framework for non-QWERTY lexical input intended to reduce gesture cost while enhancing learnability and compatibility with existing devices and applications.

DEDICATION

To my dad – David Mailman – for being my first teacher. Thank you for teaching me that curiosity is a form of love: for books, for discovery, for the world. Your steady faith in the long game made it possible to keep going when progress was measured in time between distractions.

To my partner – Allison Finer – whose joy in the creative process and undaunted courage in the face of all that life threw her inspired me to keep going.

To my mentors – Syd Lamb, Lilly Chen – and friends – Marianne, Mike, Nancy, Jeff, Trey, Ruth, who challenged me, pushed when I stalled, and let up when I needed air; your generosity is stitched into every chapter.

This dissertation is, in part, my attempt to honor what you planted—discipline, tenderness, and the stubborn hope that hard work can become a gift.

This work is dedicated to you.

ACKNOWLEDGMENTS

I gratefully acknowledge the indispensable support and direction provided by my advisor, Dr. Dalei Wu.

To my dissertation committee members —Cuilan Gao, Yingfeng Wang, and Yu Liang — thank you for your time, guidance, and contributions to this dissertation and to the broader project it represents.

To the Dean of the Graduate School – Ethan Carver – for his encouragement and guidance.

To my colleagues and friends throughout the SimCenter, the Computer Science Department, and UTC at large: thank you for the encouragement.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

GPL, gestures per lexeme

IPA, International Phonetic Alphabet

GLOSSARY OF TERMS

lexeme approximately a word or a common set phrase in a language; used in this dissertation as the basic unit of lexical input

LIST OF SYMBOLS

$\leq$, Less than or equal to

CHAPTER 1

INTRODUCTION

This research presents new software to increase user input efficiency for non-QWERTY lexical input sets (***lexica***), such as accented Latin letters, non-Latin letters, and Chinese words and phrases, all of which are represented in the Unicode (Allen et al., 2012a) character set.

This software developed in this research, ***μLex***, facilitates non-QWERTY input. ***μLex*** is designed to optimize the number of Unicode characters sent to computer apps (e.g., word processors) for the number of user gestures on keyboards, mice, and specialized input peripherals.

A unique feature of ***μLex*** is its flexibility. It can be used for input sets as simple as "Spanish Letters" and as complex as "Chinese Phrases".

Chapter 2 discusses the types of lexica, the peripherals used for lexical input, defines gesture as it applies to peripherals, and posits gestural efficiency as a measure for input systems.

Chapter 3 presents an in-depth case-study and description of a ***μLex*** application for the input of Chinese characters, words, and phrases.

Chapter 3 uses the Chinese input case study and application description to discuss ***μLex*** design concepts including associating semantics with alternative actuations, associating visual feedback with actuations, and arranging sequenced choice sets for optimizing gestural efficiency.

Chapter 4 generalizes application design in terms of the design concepts.

Chapter 5 discusses an app typology for lexical input apps and associating it with characteristics of lexica.

CHAPTER 2

BACKGROUND

This research views global languages as having ***lexemes*** (words or phrases) made up of ***lexons*** (individual unicode characters). For example, the English lexeme /father-in-law/ has lexons [f,a,t,h,e,r,-,i,n,l,w], and the Chinese lexeme /**岳父**/ (/yuèfù/, /father-in-law/), has lexons [**岳**, **父**].

When using a qwerty keyboard to input English lexemes, the gestural efficiency is approximately one ***lexon*** (letter) for one ***gesture*** or 100%.

Specialty ***input methods*** (***IME***s) enable qwerty keyboards to input non-English lexons and lexemes. Figure 2.1 shows an example of a Chinese IME.

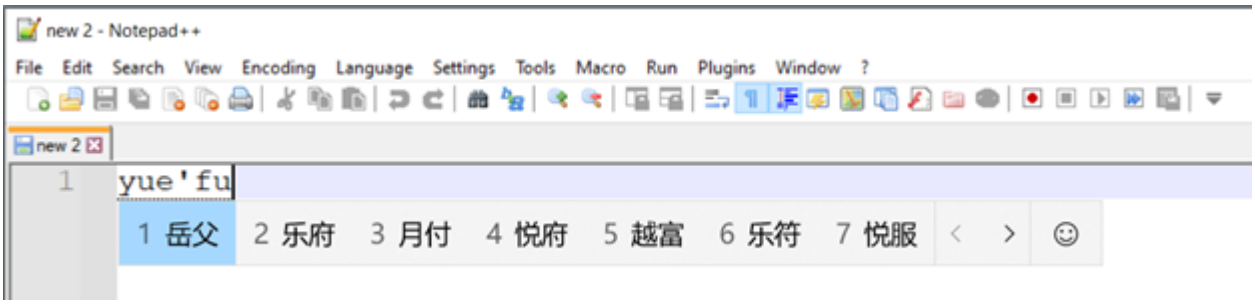

Figure 2.1: Example Chinese Input Method

In general, non-English IMEs have gestural efficiencies below 100%. For example, The Chinese IME inputs the two lexons with 6-8 qwerty gestures, for a ***gestural efficiency*** (***GE***) of

25% - 33%.

Global language IMEs are software that have dynamic displays indicating **gesture/choice associations** (***GCA***s). [TODO]example

Factors that contribute to increased IME usability are:

- Gestural Efficiency – Minimize the number of gestures required to effect lexical input.
- Touch typing – Minimize time and mental effort to learn gestures+choices GCA without reference to the display.
- Visual Scan – Minimize time and visual effort to locate displayed gesture+choice

The goal of this research is to develop techniques to produce IMEs which increase efficiency for global language by reducing the number of gestures needed to input lexemes and lexons.

The current research pulls from several diverse fields, including computer science, linguistics, and human factors psychology. This background section introduces concepts and terms to facilitate presentation of the research.

2.1 Non-QWERTY Lexica

In linguistics, a collection of words and multi-word set phrases is a ***lexicon*** (pl: ***lexica***) (Masini 2019). Individual items in a ***lexicon*** are ***lexemes***.

This research defines ***lexica*** as **sets** of statistically common lexon sequences for populations of (computer) users. So, the English language lexicon contains lexemes /a/, /the/, /in front of/, and /father-in-law/. Note that, for this research, lexica are for written, not spoken, language. Table 2.1 shows examples of lexica and their user communities.

Table 2.1 Examples of lexica and associated user communities

Lexicon	**User Community**
English Phrases	English Speakers
Chinese Phrases	Chinese Speakers
American English Legal Terms	American Lawyers
Emoticons	Texters

Lexemes are ***sequences*** of the 149,186 characters available in the Unicode standard (*Unicode FAQ*, 2023). In addition to the ***Basic Latin*** characters on the qwerty keyboard, Unicode contains many types of Unicode characters including emojis, numbers, diacritics, technical symbols, musical symbols, International Phonetic Alphabet (IPA) and Braille. It also contains language specific symbols in ***scripts***, such as Arabic, Greek, Hebrew, Japanese (Hiragana, Katakana), and Chinese.

2.2 Lexons are from the Unicode Standard

Unicode is effectively a superset of all inputtable characters (The Unicode Consortium 2025). Many linguistic input sets are summarized in Table 2.2, adapted from The Unicode Consortium (2025). The typologies (Alphabets, Abjads, Abugidas, Logosyllabaries, Simple Syllabaries, and Featural Syllabaries) are useful in understanding the relationships between a language's lexemes and its lexons.

Table 2.2 Writing-system types and example scripts

Type	Example scripts
Alphabets	Armenian, Avestan, Bassa Vah, Carian, Caucasian Albanian, Coptic, Cyrillic, Deseret, Elbasan, Georgian, Glagolitic, Gothic, Greek, Kayah Li, Latin, Lisu, Lycian, Lydian, Mahajani, Mandaic, Meroitic Cursive, Meroitic Hieroglyphs, Mongolian, Mro, N′ Ko, Ogham, Old Italic, Old Permic, Old Persian, Old Turkic, Osmanya, Pahawh Hmong, Pau Cin Hau, Runic, Shavian, Thaana, Tifinagh, Ugaritic, Wancho, Warang Citi
Abjads	Arabic, Hebrew, Imperial Aramaic, Inscriptional Pahlavi, Inscriptional Parthian, Manichaean, Nabataean, Old North Arabian, Old South Arabian, Old Uyghur, Palmyrene, Phoenician, Psalter Pahlavi, Samaritan, Syriac
Abugidas	Balinese, Batak, Bengali, Brahmi, Buginese, Buhid, Chakma, Cham, Devanagari, Grantha, Gujarati, Gurmukhi, Hanunóo, Javanese, Kaithi, Kannada, Kawi, Khmer, Khojki, Khudawadi, Lao, Lepcha, Limbu, Malayalam, Meetei Mayek, Modi, Myanmar, New Tai Lue, Newa, Oriya, Phags-pa, Rejang, Saurashtra, Sharada, Siddham, Sinhala, Sora Sompeng, Sundanese, Syloti Nagri, Tagalog, Tagbanwa, Tai Le, Tai Tham, Tai Viet, Takri, Tamil, Telugu, Thai, Tibetan, Tirhuta
Logosyllabaries	Egyptian Hieroglyphs, Han, Linear A, Sumero-Akkadian
Simple Syllabaries	Bamum, Bopomofo, Canadian Aboriginal Syllabics, Cherokee, Cypriot, Ethiopic, Hiragana, Katakana, Linear B, Mende Kikakui, Miao, Vai, Yi
Featural Syllabaries	Hangul

2.3 Lexical Input Strategies

There are two main strategies for software to give users choices leading to lexical input:

- **Character selection.** Users choose individual characters (lexons) one at a time to compose lexemes. Keyboarding English with a QWERTY layout is a common example.
- **Sequenced selection.** Users make choices from a display that associates gestures on peripherals with lexemes rather than individual characters. Mobile phone texting apps that offer word-level suggestions instead of letter-by-letter input are familiar examples. More complex sequenced-selection methods present a series of displays in which each selection changes the next set of available choices, so that a sequence of selections leads to a lexical item being input.

Character-selection systems may require more gestures (for example, keystrokes) to input a lexeme, but they have the advantage that any lexeme can be assembled from its comprising characters. Sequenced selection systems, on the other hand, can reduce the number of gestures needed for many lexemes; but, given the productive nature of language, they cannot practically encompass entire lexica.

Given that sequenced selection systems will not include all lexemes, they must be supplemented with character-selection systems. In practice, ***μLex*** apps therefore operate either as pure character-selection systems or as combined character + sequence-selection systems.

2.4 Lexical Input Peripheral Devices

μLex is device-agnostic; it can be used with common input peripherals such as keyboards, mice, and touchscreens as well as with less-common input peripherals, the number and variety of which is increasing over time (R. F. Rosenberg 1998). Enumerating and describing the full range of specialized, novel, and experimental input devices is outside the scope of this research. Instead, a single class of peripherals—***chordsets*** (chordal handsets)—is used as a representative hardware platform (Noyes 1983; Seibel 1962; R. Rosenberg and Slater 1999; Rude 2019; Tarniceriu, Dillenbourg, and Rimoldi 2013; Wu, C.-Y. Chen, and C.-H. Chen 2007). Later sections will describe how chordsets are used with ***μLex***.

2.5 Keyboards

For the development of the ***μLex*** software, the keyboard's salient feature is the ***marks*** on its keys which, taken together, comprise an ***input set***. Most keyboard marks indicate characters to be input. However, the keyboard is often used for other purposes than direct character input; the semantics of the marks may be redefined by the currently active software.

For instance, the standard Microsoft Windows Chinese ***input method*** (***IME***) (Zheng et al.

2011) presents candidate characters that can be selected via numeric key or mouse click. In this IME, the semantics of the numeric key marks are positions on the ***graphical user interface*** (***GUI***) rather than the literal digits themselves.

2.6 Mice

The salient feature for mice (and several other peripherals, such as touchscreens) is that they move a cursor to select positions on a software-defined GUI. Once a mouse is positioned on a GUI element, there are several possible completions to the gesture: different mouse buttons can be actuated, and each can be actuated in several ways (single click, double click, press-and-hold, and so on). Software is responsible for associating these mouse actuations with actions, which may or may not produce lexical input. For example, many GUIs use mouse-based selection on menus to allow users to choose software functionalities rather than enter characters, as illustrated in Figure 2.2.

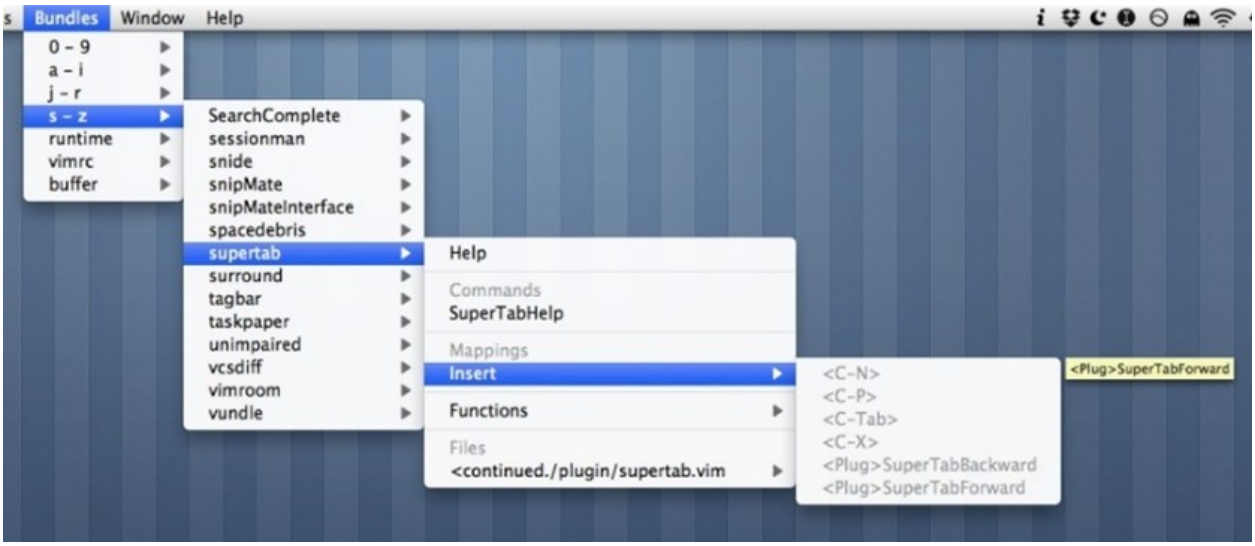

Figure 2.2: Example GUI menu selection using a mouse-controlled cursor

2.7 Chordal Input Peripherals

Chordsets are input peripherals that enable user commands by actuating several keys simultaneously, analogous to playing a chord on a piano. The number of distinct selectable items for a given chordset is the number of non-empty combinations of its individual actuators. For example, there are several one-hand chordsets (see Figure 2.3). With one actuator for each finger (five actuators), there are $2^5 - 1 = 31$ possible non-empty chords, a large enough set to make simple English text input feasible (Seibel 1962). Chordsets are sometimes used in contexts that are too small or constrained for a standard keyboard, such as underwater survey sleds or bicycle handlebars.

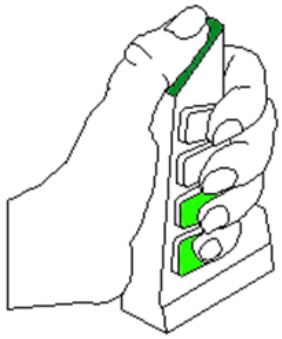

Figure 2.3: Hand positioned on a one-hand chordal handset

The salient feature of chordal input devices is that gestures are associated with discrete values; for example, a 5-actuator chordset may associate the 31 possible combinations of actuators with integers in the range 1-32. Each of these discrete integers can be associated by software with a distinct software functionality. Examples follow in future sections.

2.8 Device Agnosticism

It is not uncommon for software to allow GUI features designed for one input device to be optionally selected by another. For example, a GUI intended for keyboard selection via number keys may also allow the same choices to be selected via mouse clicks. Another common example is that many software applications provide ***keyboard accelerators*** to associate key marks with menu items, so that the same functionality can be invoked either by mouse or by keyboard (Hendy 2009; Pausch, Leatherby, and Hall 1991).

Figure 2.4 illustrates a GUI menu in which menu items can be selected either by mouse

position or by keyboard accelerators.

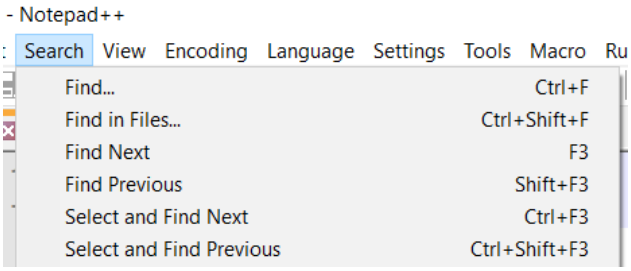

Figure 2.4: Example GUI menu showing keyboard accelerators associated with menu items

µLex is designed to be ***device agnostic***: ***µLex*** GUIs present choices that can be selected by any peripheral with the ability to send key-mark, positional, or discrete numeric information via the operating system.

2.9 Input Gestures

It is useful to view the input process in terms of user gestures. As a trivial example, to input the English lexeme '**cat**' with a keyboard, keyboards present a ***choice set*** consisting of the key marks, users make sequences of three ***selections*** from the choice set, and software inputs the lexeme. In general, Lexemes are input by ***sequences*** of ***selections*** among ***choice sets*** and ***selection*** is completed when ***actuation*** (press and release) of ***actuators*** (keys or mouse buttons) is complete.

For the purposes of this research:

- Keyboard gestures are defined as (re-)positioning the hand and/or finger(s) over the keyboard together with ***actuations*** (pressing and releasing) of keys.
- Mouse gestures are defined as (re-)positioning the cursor over GUI regions together with actuations of mouse-buttons.
- Chordal gestures are defined as actuating combinations of device actuators.

2.9.1 Alternative Actuations

Individual marked keys and GUI positional regions can be associated with several alternative actuations to increase available software functionalities. Table 2.3 illustrates example alternative keyboard actuations for the `B` key.

Table 2.3 Alternative keyboard actuations and associated software functionalities

Alternative Actuation	**Software Functionality**
`B`	Insert lowercase `b`
`<Shift>B`	Insert uppercase `B`
`<Ctrl>B`	Change selection to/from boldface

Once a cursor has been positioned on a GUI element, there are several mouse actuations that software may associate with different functionalities. Table 2.4 enumerates example alternative mouse-actuated software functionalities.

Table 2.4 Alternative mouse actuations and associated software functionalities

Alternative Actuation	**Software Functionality**
Single left click	Move cursor for insertion
Double left click	Select enclosing word
Triple left click	Select enclosing paragraph
Right click	Open context menu

2.10 Input Gestures Are Not Input Effort

Gestures and alternative actuations vary in user ***effort***:

- Typing `1` typically requires more effort than typing `a`.
- A triple click requires more effort than a single click.
- Holding `Shift` adds effort when capitalizing.
- Three-finger chords require more effort than single-finger chords.

Tables 2.3 and 2.4 give simple examples of such alternative actuations for keyboards and mice.

A related line of work focuses explicitly on user effort for lexical input. Several studies use simulated annealing or genetic algorithms to discover more ***effort-efficient*** keyboard layouts (Francis and Oxtoby 2006; Light and Anderson 1993; Onsorodi and Korhan 2020).

Effort considerations are deliberately ignored in the current research, which instead attempts to provide a more general measure of ***gesture*** that can be applied device-agnostically. Specifically, for this research, each alternative actuation—regardless of how many keys, clicks, or how much time or physical effort it requires—is quantified as a single gesture.

2.11 Gesture/Functionality Associations

All GUIs present users with associations between device gestures and software functionalities. For keyboards, these ***gesture/function associations*** (***GFAs***) make use of the choices presented by the **marks** on the individual keys. The marks are used for character input and, when combined with modifier keys such as `<Ctrl>`, `<Alt>`, `<Esc>`, and `<Shift>`, they invoke other functions such as keyboard accelerators (see Section 2.9.1 and Figure 2.4).

For mice, touchscreens, and other cursor-moving devices, GFAs are positional on the GUI itself (for example, menus as in Figure 2.4). Many GUIs do **not** use mice directly for lexical input. Instead, **supplemental** software such as virtual keyboards provides mouse-click/QWERTY-input GFAs, as illustrated in Figure 2.5.

Figure 2.5: Virtual keyboard providing QWERTY character gesture/functionality associations

For chordsets and other non-marked, non-cursor-positioning devices, GFAs are provided by supplemental displays, often available only as images or paper documents. Figure 2.6 shows an example display for the BAT chordal keyboard, pairing a printed chord/letter map with the physical device.

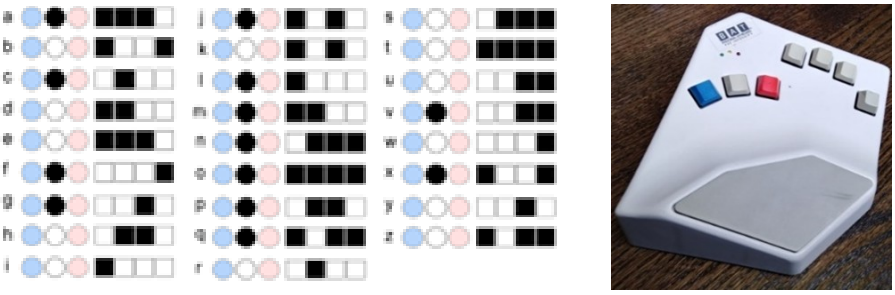

Figure 2.6: BAT chordal keyboard GFAs and hardware

2.12 Gestural Efficiency

By analogy with the definition of ***efficiency*** in physics, the basis for analysis of the software developed in this research is ***Gestural Efficiency*** (***GE***):

$$Gestural\ Efficiency\ (GE) = \frac{Characters}{Gestures}$$

Here, ***Gestures*** is the number of distinct user movements and ***Characters*** is the number of Unicode characters input. For example, the GE for keyboarding, virtual keyboard clicking, or chording the English lexeme *cat* is

$$GE_{[\text{cat}]} = \frac{3 \text{ characters}}{3 \text{ gestures}} = 1.$$

The GE for basic QWERTY keyboard input of English lexemes is thus effectively one ***character per gesture*** (***CPG***):

$$GE_{[\text{Eng,Kbd}]} = \frac{1 \text{ character}}{1 \text{ gesture}} = 1.$$

Even for single characters, GE is not always 1. For example, one way of inputting ***ñ*** in Microsoft Windows is to keyboard the six-gesture sequence shown in Figure 2.7.

```
<Fn>+<NumLk>
<Alt>+<Num 1>
<Alt>+<Num 6>
<Alt>+<Num 4>
<Fn>+<NumLk>
```

Figure 2.7: Six-gesture sequence to input **ñ** on Microsoft Windows

The GE for ***ñ*** using this input method is

$$GE_{[\tilde{n},\text{Kbd}]} = \frac{1 \text{ character}}{6 \text{ gestures}} \approx 0.17.$$

For some Far Eastern languages, such as Chinese, users commonly keyboard 5–12 gestures to select individual characters (漢字/汉字, "hànzì"), such as those shown in Table 2.5.

Table 2.5 Sample simplified Chinese characters

升 (shēng, "liter")	**吊** (diào, "lift up")	**闯** (chuǎng, "to rush")
迥 (jiǒng, "distant")	**的** (de, "possessive")	**一** (yī, "one")
是 (shì, "is")	**不** (bù, "not")	**了** (le, "perfective")

The GE for these **汉字** is approximately

$$\frac{1 \text{ character}}{12 \text{ gestures}} \leq GE_{[\text{汉字,Kbd}]} \leq \frac{1 \text{ character}}{5 \text{ gestures}}$$

or, numerically,

$$0.08 \leq GE_{[\text{汉字,Kbd}]} \leq 0.20.$$

CHAPTER 3

CASE STUDY AND REPRESENTATIVE APPLICATION

To facilitate easily-learned, gesture-efficient, on-demand non-QWERTY input, ***μLex*** input applications should enable:

- Easy switching between QWERTY and non-QWERTY.
- "Touch typing" non-QWERTY with high gestural efficiency.
- Accessibility via keyboards, mice, and other input peripherals.
- QWERTY access to non-QWERTY characters.

In the most general statement, and because the keyboard is among the most ubiquitous input peripherals, the non-QWERTY input problem reduces to presenting QWERTY-accessible choices that result in the highest number of non-QWERTY characters for the fewest gestures. The general strategy for optimizing the solution is to present choice sets of multi-character non-QWERTY lexemes—rather than single characters—for selection.

To initiate discussion, consider the example of laptop users in the field of linguistics writing papers in English about aspects of Chinese. For this example, most user time and effort are spent on word processor GUIs occupying monitors (Figure 3.1, left), and most of the document text is English, input using laptop keyboards. Periodically, and for relatively short periods of time, users input Chinese instead of English. For this use case, the non-QWERTY input set is Mandarin characters and lexemes.

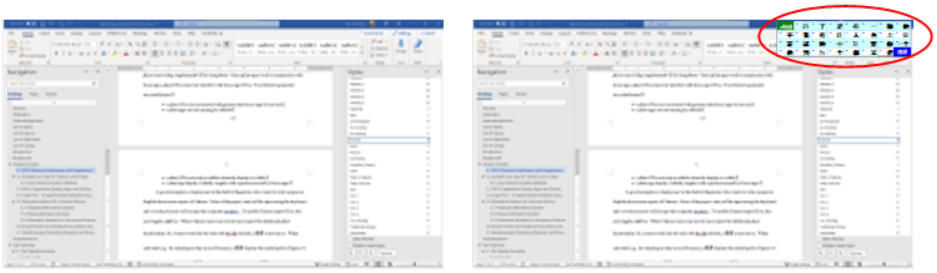

Figure 3.1: *μ*短语 supplements other apps to provide Chinese lexical input

Three resources are used to implement the ***μLex*** strategy:

- A set of Chinese characters, optionally with frequencies.
- A Chinese lexeme frequency list.
- A method for mapping QWERTY characters to Chinese characters and lexemes.

3.1 Chinese character frequency lists

Given the productive nature of language, no Chinese lexicon includes every possible lexeme. To enter lexemes not in the source frequency list, *μ*短语 enables input of individual 汉字 (hànzì, "Chinese characters"). The character frequency lists used in this research are from Da 2004. For completeness, an additional list without frequencies that includes a target set of Unicode characters with a pīnyīn pronunciation is generated programmatically.

To indicate available non-lexeme 汉字, *μ*短语 suffixes the green corner display text with a '+'.

For this research, the Unicode standard is used as the resource for the complete set of Mandarin characters (The Unicode Consortium 2025).

3.2 Chinese lexeme frequency lists from internet corpora

Lexeme frequency lists can be derived from corpora using readily available tools (Baroni and Kilgarriff 2006). The two Mandarin frequency lists used for this research are:

- Lancaster Corpus of Mandarin Chinese (**LCMC**, ~5,000 lexemes) (McEnery and Xiao 2004).
- Chinese Internet Corpus (**CIC**, ~50,000 lexemes) (Sharoff 2006).

3.3 Pīnyīn maps QWERTY to Chinese pronunciation

The most widely used system for representing Chinese pronunciation is ***pīnyīn***; it writes Chinese using the Latin alphabet and is used throughout the world (Library of Congress 1997). Because the Latin alphabet used in pīnyīn consists of the same 26 letters used for English (Central Intelligence Agency, Center for the Study of Intelligence 2007), all non-tone pīnyīn letters are available on the QWERTY keyboard (Table 3.1).

Pīnyīn spelling is syllable-based. In Standard Mandarin, each syllable can be analyzed as an *initial* (possibly null) followed by a *final*. The initial is the onset consonant (or zero onset), and the final comprises the vocalic nucleus and any offglide and/or nasal coda. For reference, the standard set of initials is:

- **Initials:** b, p, m, f, d, t, n, l, g, k, h, j, q, x, zh, ch, sh, r, z, c, s (plus null initial)

The commonly used finals include:

- **Finals:** a, o, e, i, u, ü; ai, ei, ao, ou; an, en, ang, eng, ong; ia, ie, iao, iu, ian, in, iang, ing, iong; ua, uo, uai, ui, uan, un, uang; üe, üan, ün; er

Because standard pīnyīn does not use the letter **v**, many QWERTY-based pīnyīn input methods repurpose **v** to represent **ü** (and related finals such as **üe**, **üan**, **ün**). For example, *nǚ* is often

typed as *nv*, and *lüè* as *lve*.

Table 3.1 Alphabetical key mapping for pīnyīn input (IPA + IPA description + approximate English cue)

Key	IPA	IPA description	Approx. English cue (very rough)
a	[a]	open front unrounded vowel	*a* in f*a*ther
ai	[ai̯]	[a] + [i̯] near-close front offglide	"eye" (shorter)
an	[an]	[a] + [n] alveolar nasal coda	"ahn" + n
ang	[aŋ]	[a] + [ŋ] velar nasal coda	"ahng"
ao	[au̯]	[a] + [u̯] near-close back rounded offglide	"ow" (shorter)
b	[p]	voiceless bilabial plosive (unaspirated)	*p* in *s*pin
c	[tsʰ]	voiceless alveolar affricate (aspirated)	"ts" in ca*ts* (stronger puff)
ch	[tʂʰ]	voiceless retroflex affricate (aspirated)	"ch" but tongue curled back
d	[t]	voiceless alveolar plosive (unaspirated)	*t* in *s*top
e	[ɤ]	close-mid back unrounded vowel	like "uh" but farther back
ei	[ei̯]	[e] + [i̯] near-close front offglide	"ay"
en	[ən]	[ə] + [n] alveolar nasal coda	"un" in *u*nder (approx.)
eng	[əŋ]	[ə] + [ŋ] velar nasal coda	"ung" (approx.)
er	[ɑɻ]	[ɑ] + rhotic/retroflex coloring	"are" with r-coloring
f	[f]	voiceless labiodental fricative	f in fun
g	[k]	voiceless velar plosive (unaspirated)	*k* in *s*kate
h	[x]	voiceless velar fricative	Scottish "ch" in loch (roughly)
i	[i]	close front unrounded vowel	ee in s*ee*
j	[tɕ]	voiceless alveolo-palatal affricate (unaspirated)	"j" said with tongue high/front
k	[kʰ]	voiceless velar plosive (aspirated)	k in kite (puff)
l	[l]	alveolar lateral approximant	l in low
m	[m]	bilabial nasal	m in me
n	[n]	alveolar nasal	n in no
o	[ɔ] [wo]	open-mid back rounded vowel (often with [w] glide)	"aw" / "wo" (varies)
ong	[ʊŋ] [oŋ]	rounded vowel + [ŋ] velar nasal coda	"oo" + ng (approx.)
ou	[ou̯]	[o] + [u̯] near-close back rounded offglide	"oh" glide
p	[pʰ]	voiceless bilabial plosive (aspirated)	p in pin (puff)
q	[tɕʰ]	voiceless alveolo-palatal affricate (aspirated)	aspirated j
r	[ɻ] [ʐ]	retroflex approximant / retroflex fricative	"r/zh" blend (no exact)
s	[s]	voiceless alveolar fricative	s in see
sh	[ʂ]	voiceless retroflex fricative	"sh" with tongue curled back
t	[tʰ]	voiceless alveolar plosive (aspirated)	t in top (puff)
u	[u]	close back rounded vowel	oo in f*oo*d
ui	[uei̯]	[u] + [ei̯] (second part: [ei̯] with [i̯] offglide)	"way" (approx.)
un	[uən]	[u] + [ən] (second part: [n] alveolar nasal coda)	"wen" (approx.)
uo	[wo]	[w] + [o] close-mid back rounded vowel	"wo"
v	[y]	close front rounded vowel (ü in IME input)	French *u* / German ü
ve	[yɛ]	[y] + [e] close-mid front unrounded vowel	"ü-eh" (approx.)
vn	[yn]	[y] + [n] alveolar nasal coda	"ü-n" (approx.)
x	[ɕ]	voiceless alveolo-palatal fricative	"sh" but hissier/fronted
z	[ts]	voiceless alveolar affricate (unaspirated)	"ds" in a*ds* (approx.)
zh	[tʂ]	voiceless retroflex affricate (unaspirated)	"j" with tongue curled back

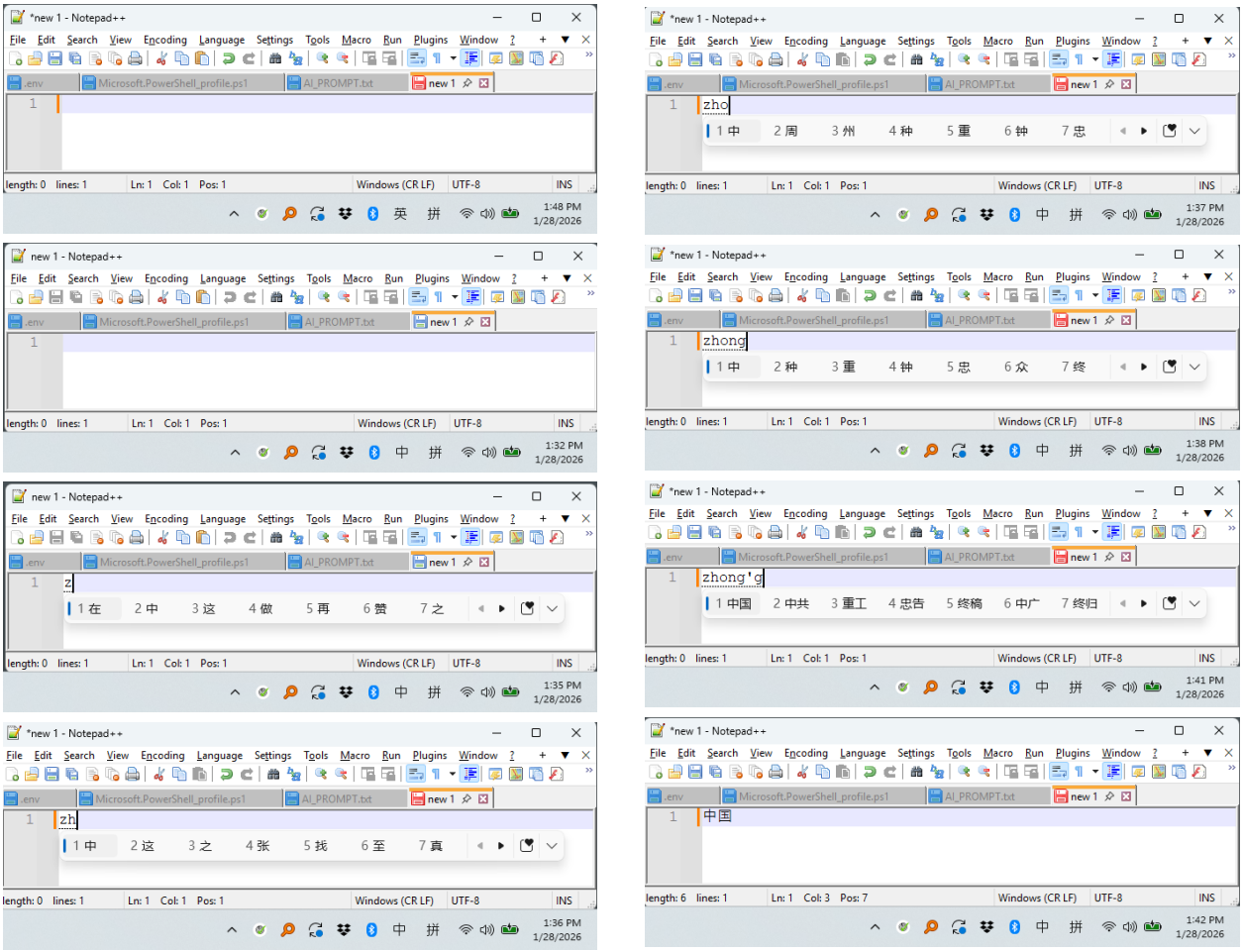

Figure 3.2: Eight consecutive keystrokes for entering the word 中国 “China” using the Microsoft Pinyin IME in Windows: `Ctrl+Space`, `z`, `h`, `o`, `n`, `g`, `g`, `1`.

Figure 3.2 shows a concrete example of the interaction required to enter the word 中国 “China” using the Microsoft Pinyin IME in Notepad++. Starting with an English keyboard layout, the user first presses `Ctrl+Space` to activate Microsoft Pinyin (step 1). They then type the letters `z`, `h`, `o`, `n`, `g`, `g` in sequence (steps 2–7). After each letter, the composition string and candidate bar update to reflect the IME’s current hypothesis about the intended syllable. On the final keystroke, `1`, the user selects candidate 1, 中国, from the candidate list and the IME commits it to the text buffer (step 8). In this example, a single lexical item requires eight keystrokes (G/Lex = 8/1) under a widely used mainstream IME.

Modern Chinese IMEs such as Microsoft Pinyin maintain an internal language model and continuously adapt the ordering of the candidate list based on corpus frequency, local context, and the user’s own selection history. In principle, this maximizes the probability that the desired character or word appears in the first candidate position, thereby reducing average selection cost for short, isolated interactions. However, the same adaptivity makes it impossible for users to rely on a stable mapping from gesture sequence to lexical outcome. A given pinyin string may produce one ordering of candidates on Monday and a different ordering on Friday, or even vary within a session as the IME updates its statistics. From the perspective of touch-typing or chordal input, where users invest effort into building reliable motor programs, this instability is a fundamental obstacle: the movement pattern “type pinyin, then press 1” cannot be learned as a robust shorthand for a particular lexeme if the identity of the first candidate is itself a moving target. In the analysis that follows, I treat such adaptive candidate reordering as a design choice that optimizes for short-term prediction accuracy at the expense of long-term gestural consistency and muscle-memory

formation.

3.4 Example *μLex* app for Chinese lexical input

*μ*短语 (mu-duǎnyǔ) is a family of ***μLex*** Chinese lexical input applications that vary with the frequency lists used to generate them. Figure 3.3 shows the initial grid display for a "small" version based on LCMC, which is used as the basis for this research. A larger version, based on CIC, is used for comparison purposes.

Figure 3.3: *μ*短语 initial grid display for Chinese lexical input (LCMC-based)

3.5 Enabling and disabling *μ*短语 with long-press

For the example use case, an always-visible *μ*短语 grid is unnecessary and potentially distracting. *μ*短语 therefore supports easy **enable** and **disable** behavior (Figures 3.3 and 3.1, red ellipse).

If *μ*短语 is not yet running (e.g., first use since boot), users **activate** it via an OS menu selection. If *μ*短语 is already running, a **long-press** of the / key **toggles it on** (**show**) and **toggles it off** (**hide**). **Deactivation** (fully exiting the application) is performed via mouse-clickable menu

selections accessed through the blue corner cell. Because activation is via the OS, show/hide is via long-press, and deactivation is via *μ*短语's own UI, none of these gestures interfere with the GFAs of concurrently running applications.

3.6 Using *μ*短语 for selecting lexemes

3.6.0.1 Select the most frequent lexemes with one gesture

The initial *μ*短语 display shows the most frequently used lexemes according to the source corpus. When the grid is visible, users select lexemes in one of two ways. They can click the cells directly or **long-press** the corresponding QWERTY keys indicated in blue (Figure 3.3, red circle). For example, tapping `K` in the usual manner inputs `k`, while long-pressing `K` inputs the lexeme 汉字 这 (pīnyīn *zhè*). Because selection uses long-press, *μ*短语 does not interfere with existing word-processing GFAs. Visual feedback (Figure 3.3, yellow highlight) indicates when the long-press threshold has been reached.

As another example, to input 汉字的 (pīnyīn *de*), users long-press `A` until the cell highlights; the lexeme is inserted upon release.

The GE for any single-character lexeme on the initial *μ*短语 grid is 1.00, and the GE for 汉字 一个 (*yī gè*) is 2.00. By comparison, in the Microsoft Pinyin IME the number of gestures to input 汉字 的 varies from 2 (`d` + `Enter`) to 3 (`d` + `e` + `1`), with corresponding GEs of 0.50 and 0.33 (not including the gestures required to switch between English and Chinese input). The lexemes on the initial display account for 18.69% of written Chinese according to LCMC.

3.6.0.2 Select the next most frequent lexemes with two gestures

To input lexemes not on the initial display, users **Shift+long-press** the first letter of the lexeme′ s pīnyīn to update the grid to that initial. For example, to input 汉字 时候 (*shí hòu*, "when"), users:

- **Shift+long-press** S to update the display to the s- grid (see Figure 3.4).
- long-press P to insert the lexeme.

S	A 时	B 使	C 所	D 什么	E 社会	F 三	G 生活
H 生产	I 世界	J 事	K 时间	L 四	M 市场	N 水	O 岁
P 时候	Q 声	R 谁	S 社会主义	T 思想	U 所以	V 手	W 上海
X 水平	Y 十分	Z 使用	1 少	2 死	3 受	4 实现	选项

Figure 3.4: *μ*短语 grid of frequently used lexemes with pīnyīn beginning with s-

There are 658 frequently used lexemes that can be selected with two gestures in this manner. The average GE for two-gesture lexemes is 0.80, and two-gesture lexemes account for 25.78% of written Chinese. Combined with the one-gesture lexemes, 44.47% of written Chinese can be input with two or fewer gestures.

3.6.0.3 Toward four or fewer gestures

REFERENCES

Baroni, Marco and Adam Kilgarriff (Apr. 2006). “Large Linguistically-Processed Web Corpora for Multiple Languages”. In: *Proceedings of the 11th Conference of the European Chapter of the Association for Computational Linguistics (EACL 2006): Demonstrations*. Demo track at EACL 2006. The main conference ran Apr 3–7, 2006; some sources list posters/demos as Apr 5–6. Trento, Italy: Association for Computational Linguistics, pp. 87–90.
URL: https://aclanthology.org/E06-2001/.

Central Intelligence Agency, Center for the Study of Intelligence (May 8, 2007). *The Progress of Pinyin*. Center for the Study of Intelligence.
URL: https://www.cia.gov/resources/csi/static/The-Progress-of-Pinyin.pdf (visited on 01/26/2026).

Da, Jun (2004). “A Corpus-Based Study of Character and Bigram Frequencies in Chinese E-Texts and Its Implications for Chinese Language Instruction”. In: *The Studies on the Theory and Methodology of the Digitalized Chinese Teaching to Foreigners: Proceedings of the Fourth International Conference on New Technologies in Teaching and Learning Chinese*. Ed. by Pu Zhang, Tianwei Xie, and Juan Xu. Publication details taken from the ″Appeared in″ statement in the author-posted PDF. Beijing: Tsinghua University Press, pp. 501–511.
URL: https://lingua.mtsu.edu/academic/dajun-4thtech.pdf.

Francis, Graham and Chris Oxtoby (2006). “Building and Testing Optimized Keyboards for Specific Text Entry”. In: *Human Factors* 48.2, pp. 279–287.
DOI: 10.1518/001872006777724390.

Hendy, James (2009). “Graphically Enhanced Keyboard Accelerators for GUIs”. MA thesis. Vancouver, BC, Canada: The University of British Columbia.
DOI: 10.14288/1.0051504.
URL: https://open.library.ubc.ca/collections/ubctheses/24/items/1.0051504.

Library of Congress (Nov. 19, 1997). *Library of Congress Will Convert to Pinyin for Romanization of Chinese*. Library of Congress.
URL: https://www.loc.gov/catdir/pinyin/announce.html (visited on 01/26/2026).

Light, Lissa W. and Peter G. Anderson (Sept. 1993). “Typewriter Keyboards via Simulated Annealing”. In: *AI Expert* 8.9, pp. 20–27. ISSN: 0888-3785.
URL: https://repository.rit.edu/cgi/viewcontent.cgi?article=1729&context=article.

Masini, Francesca (Sept. 30, 2019). *Multi-Word Expressions and Morphology*. In: *Oxford Research Encyclopedia of Linguistics*. Ed. by Mark Aronoff. Oxford University Press.
DOI: 10.1093/acrefore/9780199384655.013.611.
URL: https://oxfordre.com/linguistics/display/10.1093/acrefore/9780199384655.001.0001/acrefore-9780199384655-e-611.

McEnery, Anthony and Zhonghua Xiao (May 2004). "The Lancaster Corpus of Mandarin Chinese: A Corpus for Monolingual and Contrastive Language Study". In: *Proceedings of the Fourth International Conference on Language Resources and Evaluation (LREC'04)*. Ed. by Maria Teresa Lino et al. Lisbon, Portugal:
European Language Resources Association (ELRA), pp. 1175–1178.
URL: https://aclanthology.org/L04-1117/.

Noyes, J. (Mar. 1983). "Chord Keyboards". In: *Applied Ergonomics* 14.1, pp. 55–59.
DOI: 10.1016/0003-6870(83)90221-1.

Onsorodi, Ali H. H. and Orhan Korhan (2020). "Application of a Genetic Algorithm to the Keyboard Layout Problem". In: *PLOS ONE* 15.1, e0226611.
DOI: 10.1371/journal.pone.0226611.

Pausch, Randy, Jeffrey H. Leatherby, and Theodore Hall (1991). *Voice Input vs. Keyboard Accelerators: A User Study*. Tech. rep. TR-91-22. University of Virginia, Department of Computer Science.
DOI: 10.18130/V3GN3B.
URL: https://doi.org/10.18130/V3GN3B.

Rosenberg, R. and Mel Slater (1999). "The Chording Glove: A Glove-Based Text Input Device". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 29.2, pp. 186–191.
DOI: 10.1109/5326.760563.

Rosenberg, Richard F. (1998). "Computing Without Mice and Keyboards: Text and Graphic Input Devices for Mobile Computing". PhD thesis. University College London.
URL: https://discovery.ucl.ac.uk/id/eprint/10103046/.

Rude, Mark (2019). "Using the QWERTY Keyboard as a Chord Keyboard: Syllabic Typing by Multi-Key Strokes for Language Learning & More". In: *AI and Machine Learning in Language Education: Selected Papers from the JALTCALL2019 Conference, Tokyo, Japan*. Ed. by Robert Chartrand and Edo Forsythe. Tokyo, Japan: JALT CALL SIG, pp. 168–180.
URL: https://jaltcall.org/wp-content/uploads/2019/06/AI-and-Machine-Learning-in-Language-Education.pdf.

Seibel, Robert (1962). "Performance on a Five-Finger Chord Keyboard". In: *Journal of Applied Psychology* 46.3, pp. 165–169.
DOI: 10.1037/h0047948.

Sharoff, Serge (2006). "Creating General-Purpose Corpora
Using Automated Search Engine Queries". In: *WaCky! Working Papers on the Web as Corpus*. Ed. by Marco Baroni and Silvia Bernardini. Bologna, Italy: Gedit, pp. 63–98.
URL: https://wacky.sslmit.unibo.it/doku.php?id=publications.

Tarniceriu, Adriana D., Pierre Dillenbourg, and Bertrand Rimoldi (2013). “The Effect of Feedback on Chord Typing”. In: *Proceedings of the Seventh International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies (UBICOMM 2013)*. Porto, Portugal: IARIA, pp. 69–74. ISBN: 978-1-61208-289-9.
URL: https://www.thinkmind.org/index.php?view=article&articleid=ubicomm_2013_8_20_20067.

The Unicode Consortium (2025). *The Unicode Standard, Version 17.0.0*. South San Francisco, CA: The Unicode Consortium. ISBN: 978-1-936213-35-1.
URL: https://www.unicode.org/versions/Unicode17.0.0/.

Wu, Fu-Gui, Chia-Yi Chen, and Chih-Hung Chen (2007). “Improvements of Chord Input Devices for Mobile Computer Users”. In: *Universal Access in Human-Computer Interaction: Ambient Interaction*. Ed. by Constantine Stephanidis. Vol. 4555. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer.
DOI: 10.1007/978-3-540-73281-5_66.

Zheng, Ying et al. (2011). “Why Press Backspace? Understanding User Input Behaviors in Chinese Pinyin Input Method”. In: *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies (Volume 2: Short Papers)*. Portland, Oregon, USA: Association for Computational Linguistics, pp. 485–490.
URL: https://aclanthology.org/P11-2085/.

WORKS CONSULTED

Her, One-Soon (Dec. 2005). “Between Globalization and Indigenization: On Taiwan’s Pinyin Issue from the Perspectives of the New Economy”. In: *Journal of Social Sciences and Philosophy* 17.4, pp. 785–822.
URL: https://www.sinica.edu.tw/~zbt/pub/cc1744a.pdf.

Unicode Consortium (2023). *Unicode FAQ*. Accessed 2025-12-23.
URL: https://unicode.org/faq/.

APPENDIX A

MINIMAL BROWSER DEMONSTRATIONS

This appendix preserves two minimal browser demonstrations used in the dissertation. Each one is intentionally small enough to include in print while remaining easy to run in a local browser. The JavaScript files in `appendices/code/` are frozen snapshots of the dissertation demos, so the appendix remains self-contained even if the development copies continue to evolve.

If the optional `minted` switch in the preamble is enabled later, these listings can be syntax highlighted. Otherwise they fall back to a conservative colored monospace presentation.

A.1 µTeclas

The *µTeclas* demo isolates the keyboard timing model. To run it, place the following HTML harness in the same directory as `_muTeclas-dissertation.js`, then open the HTML file in a browser.

```
<!doctype html>
<html lang="en">
  <body>
    <div id="muTeclasPreview"></div>
    <textarea id="muTeclasOutput"></textarea>
    <script src="./_muTeclas-dissertation.js"></script>
  </body>
</html>
```



```
/*
  Minimal standalone browser implementation of µTeclas for dissertation
   ↪  appendix use.

  To run:
    1. Add `<div id="muTeclasPreview"></div>` and
       `<textarea id="muTeclasOutput"></textarea>` to a page.
    2. Load this script after those elements.
    3. The demo auto-initializes.
*/

(function () {
  "use strict";

  const LONG_PRESS_MSECS = 400;
  const MAP = {
    a: { base: "a", special: "á", upper: "Á" },
    e: { base: "e", special: "é", upper: "É" },
    i: { base: "i", special: "í", upper: "Í" },
    o: { base: "o", special: "ó", upper: "Ó" },
    u: { base: "u", special: "ú", upper: "Ú" },
    v: { base: "v", special: "ü", upper: "Ü" },
    n: { base: "n", special: "ñ", upper: "Ñ" },
    c: { base: "c", special: "ç", upper: "Ç" },
    "!": { base: "!", special: "¡" },
    "?": { base: "?", special: "¿" },
    "<": { base: "<", special: "«" },
    ">": { base: ">", special: "»" },
    "$": { base: "$", special: "€" },
    P: { base: "P", special: "₧" }
  };

  function mountMuTeclasDissertation(config) {
    const output = config.output;
    const preview = config.preview;
    if (!(output instanceof HTMLTextAreaElement) || !(preview instanceof
     ↪  HTMLElement)) {
      throw new Error(
        "mountMuTeclasDissertation requires a textarea output and a preview
         ↪  element."
      );
    }

    injectStyles();

    const state = {
      pendingKey: null,
      pendingSince: 0,
      shift: false,
```



```
    caps: false,
    frame: 0
  };

  function normalizeKey(key) {
    if (typeof key !== "string" || key.length === 0) {
      return "";
    }
    return key === "P" ? "P" : (key.length === 1 ? key.toLowerCase() :
    ↪  key);
  }

  function isMappedKey(key) {
    return Object.prototype.hasOwnProperty.call(MAP, normalizeKey(key));
  }

  function useUpper(mods) {
    return Boolean(mods.shift) !== Boolean(mods.caps);
  }

  function baseChar(key, upper) {
    const entry = MAP[normalizeKey(key)];
    if (!entry) {
      return "";
    }
    return upper && /^[a-z]$/.test(entry.base) ? entry.base.toUpperCase() :
    ↪  entry.base;
  }

  function specialChar(key, upper) {
    const entry = MAP[normalizeKey(key)];
    if (!entry) {
      return "";
    }
    return upper ? (entry.upper || entry.special) : entry.special;
  }

  function pendingText(key, upper, longPress) {
    const special = specialChar(key, upper);
    return longPress ? special : `{${special}}`;
  }

  function insertText(text) {
    const start = output.selectionStart ?? output.value.length;
    const end = output.selectionEnd ?? output.value.length;
    output.value = `${output.value.slice(0,
    ↪  start)}${text}${output.value.slice(end)}`;
    output.selectionStart = start + text.length;
    output.selectionEnd = start + text.length;
    output.focus();
```



```
}

function passThroughText(event) {
  if (event.key === ”Enter”) {
    return ”\n”;
  }
  if (event.key === ”Tab”) {
    return ”\t”;
  }
  return event.key.length === 1 ? event.key : ””;
}

function clearPending() {
  state.pendingKey = null;
  state.pendingSince = 0;
}

function isLongPress(now) {
  return state.pendingKey !== null && now - state.pendingSince >=
  ↪  LONG_PRESS_MSECS;
}

function currentUpper() {
  return useUpper({ shift: state.shift, caps: state.caps });
}

function emitPending(now) {
  if (!state.pendingKey) {
    return;
  }
  insertText(
    isLongPress(now)
      ? specialChar(state.pendingKey, currentUpper())
      : baseChar(state.pendingKey, currentUpper())
  );
  clearPending();
}

function startPending(key) {
  state.pendingKey = normalizeKey(key);
  state.pendingSince = performance.now();
  render();
}

function scheduleRender() {
  if (state.frame) {
    return;
  }
  state.frame = requestAnimationFrame(function () {
    state.frame = 0;
```



```
    render();
  });
}

function render() {
  const now = performance.now();
  const visualState = state.pendingKey ? (isLongPress(now) ? "posttimer"
  ↪  : "pretimer") : "idle";
  preview.dataset.visualState = visualState;
  preview.textContent = state.pendingKey
    ? pendingText(state.pendingKey, currentUpper(), visualState ===
    ↪  "posttimer")
    : "µTeclas";
  if (state.pendingKey) {
    scheduleRender();
  }
}

function onKeyDown(event) {
  state.shift = Boolean(event.shiftKey);
  state.caps = Boolean(event.getModifierState &&
  ↪  event.getModifierState("CapsLock"));

  if (event.repeat && isMappedKey(event.key)) {
    event.preventDefault();
    render();
    return;
  }
  if (event.repeat || event.ctrlKey || event.altKey || event.metaKey) {
    render();
    return;
  }

  const key = normalizeKey(event.key);
  if (state.pendingKey && key !== state.pendingKey) {
    emitPending(performance.now());
  }

  if (!isMappedKey(event.key)) {
    const text = passThroughText(event);
    if (text) {
      event.preventDefault();
      insertText(text);
    }
    render();
    return;
  }

  if (key !== state.pendingKey) {
    startPending(event.key);
```



```
    }

    event.preventDefault();
    render();
  }

  function onKeyUp(event) {
    state.shift = Boolean(event.shiftKey);
    state.caps = Boolean(event.getModifierState &&
    ↪  event.getModifierState("CapsLock"));

    if (!isMappedKey(event.key) || normalizeKey(event.key) !==
    ↪  state.pendingKey) {
      render();
      return;
    }

    emitPending(performance.now());
    render();
  }

  document.addEventListener("keydown", onKeyDown);
  document.addEventListener("keyup", onKeyUp);
  output.focus();
  render();

  return {
    destroy: function () {
      document.removeEventListener("keydown", onKeyDown);
      document.removeEventListener("keyup", onKeyUp);
      if (state.frame) {
        cancelAnimationFrame(state.frame);
      }
    }
  };
}

function injectStyles() {
  if (document.getElementById("muTeclasDissertationStyles")) {
    return;
  }

  const style = document.createElement("style");
  style.id = "muTeclasDissertationStyles";
  style.textContent = `
    #muTeclasPreview {
      min-height: 8rem;
      border: 2px solid #ccb892;
      border-radius: 16px;
      background: linear-gradient(135deg, #f3efe4, #ece2cc);
```



```
        display: grid;
        place-items: center;
        font: 700 clamp(2rem, 6vw, 3.4rem)/1 Georgia, serif;
        color: #2e2618;
      }

      #muTeclasPreview[data-visual-state='pretimer'] {
        background: linear-gradient(135deg, #f8e8aa, #ecd786);
        border-color: #c0982a;
        color: #7c2f10;
      }

      #muTeclasPreview[data-visual-state='posttimer'] {
        background: linear-gradient(135deg, #d9e6ce, #bdd09c);
        border-color: #78965c;
        color: #21492c;
      }

      #muTeclasOutput {
        width: min(460px, 100%);
        min-height: 10rem;
        margin-top: .8rem;
        padding: .75rem .9rem;
        border: 1px solid #ccb892;
        border-radius: 10px;
        font: 1rem/1.4 Georgia, serif;
      }
    `;
    document.head.appendChild(style);
  }

  window.mountMuTeclasDissertation = mountMuTeclasDissertation;

  function autoInit() {
    const preview = document.getElementById("muTeclasPreview");
    const output = document.getElementById("muTeclasOutput");
    if (preview && output) {
      mountMuTeclasDissertation({ preview: preview, output: output });
    }
  }

  if (document.readyState === "loading") {
    document.addEventListener("DOMContentLoaded", autoInit, { once: true });
  } else {
    autoInit();
  }
})();
```


A.2 µLetras

The *µLetras* demo preserves the same timing model while adding the visual grid that links each key to its special character. To run it, place the following HTML harness in the same directory as `_muLetras-dissertation.js`, then open the HTML file in a browser.

```
<!doctype html>
<html lang="en">
  <body>
    <div id="muLetrasGrid"></div>
    <textarea id="muLetrasOutput"></textarea>
    <script src="./_muLetras-dissertation.js"></script>
  </body>
</html>
```

```
/*
  Minimal standalone browser implementation of µLetras for dissertation
  ↪  appendix use.

  To run:
    1. Add `<div id="muLetrasGrid"></div>` and
       `<textarea id="muLetrasOutput"></textarea>` to a page.
    2. Load this script after those elements.
    3. The demo auto-initializes.
*/

(function () {
  "use strict";

  const LONG_PRESS_MSECS = 400;
  const ORDER = ["a", "e", "i", "o", "u", "v", "n", "c", "!", "?", "<", ">",
  ↪  "$", "P"];
  const MAP = {
    a: { base: "a", special: "á", upper: "Á" },
    e: { base: "e", special: "é", upper: "É" },
    i: { base: "i", special: "í", upper: "Í" },
    o: { base: "o", special: "ó", upper: "Ó" },
    u: { base: "u", special: "ú", upper: "Ú" },
    v: { base: "v", special: "ü", upper: "Ü" },
    n: { base: "n", special: "ñ", upper: "Ñ" },
    c: { base: "c", special: "ç", upper: "Ç" },
    "!": { base: "!", special: "¡" },
    "?": { base: "?", special: "¿" },
    "<": { base: "<", special: "«" },
```



```
  ">": { base: ">", special: "»" },
  "$": { base: "$", special: "€" },
  P: { base: "P", special: "₧" }
};

function mountMuLetrasDissertation(config) {
  const output = config.output;
  const grid = config.grid;
  if (!(output instanceof HTMLTextAreaElement) || !(grid instanceof
  ↪  HTMLElement)) {
    throw new Error("mountMuLetrasDissertation requires a textarea output
    ↪  and a grid container.");
  }

  injectStyles();
  buildGrid(grid);

  const state = {
    pendingKey: null,
    pendingSince: 0,
    pendingSource: null,
    pendingMouseButton: null,
    pendingPhysicalButton: null,
    pendingUpper: false,
    hoveredKey: null,
    shift: false,
    caps: false,
    frame: 0
  };

  function normalizeKey(key) {
    if (typeof key !== "string" || key.length === 0) {
      return "";
    }
    return key === "P" ? "P" : (key.length === 1 ? key.toLowerCase() :
    ↪  key);
  }

  function isMappedKey(key) {
    return Object.prototype.hasOwnProperty.call(MAP, normalizeKey(key));
  }

  function useUpper(mods) {
    return Boolean(mods.shift) !== Boolean(mods.caps);
  }

  function baseChar(key, upper) {
    const entry = MAP[normalizeKey(key)];
    if (!entry) {
      return "";
```



```
  }
  return upper && /^[a-z]$/.test(entry.base) ? entry.base.toUpperCase() :
  ↪  entry.base;
}

function specialChar(key, upper) {
  const entry = MAP[normalizeKey(key)];
  if (!entry) {
    return "";
  }
  return upper ? (entry.upper || entry.special) : entry.special;
}

function pendingText(key, upper, longPress) {
  const special = specialChar(key, upper);
  return longPress ? special : `{${special}}`;
}

function insertText(text) {
  const start = output.selectionStart ?? output.value.length;
  const end = output.selectionEnd ?? output.value.length;
  output.value = `${output.value.slice(0,
  ↪  start)}${text}${output.value.slice(end)}`;
  output.selectionStart = start + text.length;
  output.selectionEnd = start + text.length;
  output.focus();
}

function passThroughText(event) {
  if (event.key === "Enter") {
    return "\n";
  }
  if (event.key === "Tab") {
    return "\t";
  }
  return event.key.length === 1 ? event.key : "";
}

function clearPending() {
  state.pendingKey = null;
  state.pendingSince = 0;
  state.pendingSource = null;
  state.pendingMouseButton = null;
  state.pendingPhysicalButton = null;
  state.pendingUpper = false;
}

function isLongPress(now) {
  return state.pendingKey !== null && now - state.pendingSince >=
  ↪  LONG_PRESS_MSECS;
```



```
  }

  function currentUpper() {
    return state.pendingSource === "mouse"
      ? state.pendingUpper
      : useUpper({ shift: state.shift, caps: state.caps });
  }

  function emitPending(now) {
    if (!state.pendingKey) {
      return;
    }
    insertText(
      isLongPress(now)
        ? specialChar(state.pendingKey, currentUpper())
        : baseChar(state.pendingKey, currentUpper())
    );
    clearPending();
  }

  function startPending(key, source, mods, mouseButton, physicalButton) {
    state.pendingKey = normalizeKey(key);
    state.pendingSince = performance.now();
    state.pendingSource = source;
    state.pendingMouseButton = mouseButton || null;
    state.pendingPhysicalButton = typeof physicalButton === "number" ?
    ↪  physicalButton : null;
    state.pendingUpper = useUpper(mods);
    render();
  }

  function pointerMode(event) {
    return event.button === 2 || (event.button === 0 && event.shiftKey) ?
    ↪  "right" : "left";
  }

  function scheduleRender() {
    if (state.frame) {
      return;
    }
    state.frame = requestAnimationFrame(function () {
      state.frame = 0;
      render();
    });
  }

  function render() {
    const now = performance.now();
    const visualState = state.pendingKey ? (isLongPress(now) ? "posttimer"
    ↪  : "pretimer") : "idle";
```



```
  grid.querySelectorAll("[data-key]").forEach(function (cell) {
    const key = cell.dataset.key;
    const glyph = cell.querySelector(".muLetras-glyph");
    const upper = state.pendingKey === key
      ? currentUpper()
      : useUpper({ shift: state.shift, caps: state.caps });

    cell.className = "muLetras-cell";

    if (state.pendingKey === key) {
      if (state.pendingSource === "mouse") {
        const mouseClass = [
          "mouse",
          state.pendingMouseButton,
          visualState
        ].join("-");
        cell.classList.add(mouseClass);
      } else {
        cell.classList.add(visualState);
      }
      glyph.textContent = pendingText(key, upper, visualState ===
      ↪  "posttimer");
    } else {
      cell.classList.add("idle");
      if (state.hoveredKey === key) {
        cell.classList.add("hover");
      }
      glyph.textContent = specialChar(key, upper);
    }
  });

  if (state.pendingKey) {
    scheduleRender();
  }
}

function onKeyDown(event) {
  state.shift = Boolean(event.shiftKey);
  state.caps = Boolean(event.getModifierState &&
  ↪  event.getModifierState("CapsLock"));

  if (event.repeat && isMappedKey(event.key)) {
    event.preventDefault();
    render();
    return;
  }
  if (event.repeat || event.ctrlKey || event.altKey || event.metaKey) {
    render();
    return;
```



```
    }

    const key = normalizeKey(event.key);
    const now = performance.now();
    const mods = { shift: state.shift, caps: state.caps };

    if (state.pendingKey && key !== state.pendingKey) {
      emitPending(now);
    }

    if (!isMappedKey(event.key)) {
      const text = passThroughText(event);
      if (text) {
        event.preventDefault();
        insertText(text);
      }
      render();
      return;
    }

    if (key !== state.pendingKey) {
      startPending(event.key, "keyboard", mods);
    }

    event.preventDefault();
    render();
  }

  function onKeyUp(event) {
    state.shift = Boolean(event.shiftKey);
    state.caps = Boolean(event.getModifierState &&
    ↪  event.getModifierState("CapsLock"));

    if (!isMappedKey(event.key) || normalizeKey(event.key) !==
    ↪  state.pendingKey) {
      render();
      return;
    }

    emitPending(performance.now());
    render();
  }

  function onMouseDown(event) {
    const cell = event.target.closest("[data-key]");
    if (!cell || !grid.contains(cell)) {
      return;
    }

    event.preventDefault();
```



```
  output.focus();

  const key = cell.dataset.key;
  const mods = { shift: pointerMode(event) === "right", caps: false };

  if (state.pendingKey && key !== state.pendingKey) {
    emitPending(performance.now());
  }

  startPending(key, "mouse", mods, pointerMode(event), event.button);
}

function onMouseUp(event) {
  if (state.pendingSource !== "mouse" || event.button !==
  ↪  state.pendingPhysicalButton) {
    return;
  }
  event.preventDefault();
  emitPending(performance.now());
  render();
}

function onMouseOver(event) {
  const cell = event.target.closest("[data-key]");
  state.hoveredKey = cell ? cell.dataset.key : null;
  render();
}

function onMouseLeave(event) {
  if (!grid.contains(event.relatedTarget)) {
    state.hoveredKey = null;
    render();
  }
}

function onContextMenu(event) {
  if (event.target.closest(".muLetras-cell")) {
    event.preventDefault();
  }
}

document.addEventListener("keydown", onKeyDown);
document.addEventListener("keyup", onKeyUp);
document.addEventListener("mouseup", onMouseUp);
grid.addEventListener("mousedown", onMouseDown);
grid.addEventListener("mouseover", onMouseOver);
grid.addEventListener("mouseleave", onMouseLeave);
grid.addEventListener("contextmenu", onContextMenu);
output.focus();
render();
```



```
  return {
    destroy: function () {
      document.removeEventListener("keydown", onKeyDown);
      document.removeEventListener("keyup", onKeyUp);
      document.removeEventListener("mouseup", onMouseUp);
      grid.removeEventListener("mousedown", onMouseDown);
      grid.removeEventListener("mouseover", onMouseOver);
      grid.removeEventListener("mouseleave", onMouseLeave);
      grid.removeEventListener("contextmenu", onContextMenu);
      if (state.frame) {
        cancelAnimationFrame(state.frame);
      }
    }
  };
}

function buildGrid(grid) {
  grid.classList.add("muLetras-grid");
  grid.replaceChildren();
  grid.appendChild(makeCell("muLetras-title", "µLetras"));

  ORDER.forEach(function (key) {
    const cell = makeCell("muLetras-cell idle");
    cell.dataset.key = key;
    cell.appendChild(makeSpan("muLetras-key", key));
    cell.appendChild(makeSpan("muLetras-glyph", MAP[key].special));
    grid.appendChild(cell);
  });

  grid.appendChild(makeCell("muLetras-cell muLetras-options", "options"));
}

function makeCell(className, text) {
  const cell = document.createElement("div");
  cell.className = className;
  if (text) {
    cell.textContent = text;
  }
  return cell;
}

function makeSpan(className, text) {
  const span = document.createElement("span");
  span.className = className;
  span.textContent = text;
  return span;
}

function injectStyles() {
```



```
if (document.getElementById("muLetrasDissertationStyles")) {
  return;
}

const style = document.createElement("style");
style.id = "muLetrasDissertationStyles";
style.textContent = `
  .muLetras-grid {
    display: grid;
    grid-template-columns: repeat(4, minmax(0, 1fr));
    gap: .16rem;
    padding: .18rem;
    border: 1px solid #c7b089;
    border-radius: 7px;
    background: linear-gradient(180deg, #d9c49d, #ccb07e);
    max-width: 460px;
  }

  .muLetras-cell,
  .muLetras-title {
    position: relative;
    aspect-ratio: 1.35/1;
    border: 1px solid #ccb892;
    border-radius: 3px;
    background: linear-gradient(135deg, #f5f1e7, #eee3cb);
    display: flex;
    align-items: center;
    justify-content: center;
    padding: .18rem;
    overflow: hidden;
    box-shadow: inset 0 1px 0 rgba(255, 255, 255, .45);
    user-select: none;
  }

  .muLetras-cell.idle {
    border-color: #ccb892;
  }

  .muLetras-cell.hover {
    background: linear-gradient(135deg, #f3e4c6, #ecd6ab);
    border-color: #b88d44;
    color: #7b4914;
  }

  .muLetras-cell.pretimer {
    background: linear-gradient(135deg, #f8e8aa, #ecd786);
    border-color: #c0982a;
    color: #7c2f10;
  }
```



```
.muLetras-cell.posttimer {
  background: linear-gradient(135deg, #d9e6ce, #bdd09c);
  border-color: #78965c;
  color: #21492c;
}

.muLetras-cell.mouse-left-pretimer {
  background: linear-gradient(135deg, #ead8f1, #d9bce7);
  border-color: #8a63a5;
  color: #5d3577;
}

.muLetras-cell.mouse-left-posttimer {
  background: linear-gradient(135deg, #cfe9ff, #a9cff0);
  border-color: #5a94c5;
  color: #1b4f79;
}

.muLetras-cell.mouse-right-pretimer {
  background: linear-gradient(135deg, #ffd6d9, #f2b0b9);
  border-color: #c46a78;
  color: #8b2636;
}

.muLetras-cell.mouse-right-posttimer {
  background: linear-gradient(135deg, #d4f0d5, #afd9b2);
  border-color: #619067;
  color: #244f2d;
}

.muLetras-title {
  border-color: #b48b4d;
  background: linear-gradient(135deg, #efdcae, #ddb96a);
  color: #8a3b12;
  font: 700 .85rem/1 Georgia, serif;
  display: flex;
  justify-content: center;
  align-items: center;
}

.muLetras-options {
  border-style: dashed;
  color: #989184;
  font-size: .72rem;
}

.muLetras-key {
  position: absolute;
  top: .22rem;
  left: .28rem;
```



```
      color: #b03321;
      font-size: .62rem;
      font-weight: 700;
      line-height: 1;
    }

    .muLetras-glyph {
      max-width: calc(100% - .5rem);
      max-height: calc(100% - .42rem);
      font-size: clamp(1.7rem, 3.15vw, 2.3rem);
      line-height: .92;
      text-align: center;
      transform: translateY(.04rem);
    }

    .muLetras-cell.pretimer .muLetras-glyph,
    .muLetras-cell.mouse-left-pretimer .muLetras-glyph,
    .muLetras-cell.mouse-right-pretimer .muLetras-glyph {
      font-size: clamp(1.22rem, 2.15vw, 1.62rem);
      line-height: .98;
      letter-spacing: -.01em;
    }

    #muLetrasOutput {
      width: min(460px, 100%);
      min-height: 10rem;
      margin-top: .8rem;
      padding: .75rem .9rem;
      border: 1px solid #ccb892;
      border-radius: 10px;
      font: 1rem/1.4 Georgia, serif;
    }
  `;
  document.head.appendChild(style);
}

window.mountMuLetrasDissertation = mountMuLetrasDissertation;

function autoInit() {
  const grid = document.getElementById("muLetrasGrid");
  const output = document.getElementById("muLetrasOutput");
  if (grid && output) {
    mountMuLetrasDissertation({ grid: grid, output: output });
  }
}

if (document.readyState === "loading") {
  document.addEventListener("DOMContentLoaded", autoInit, { once: true });
} else {
  autoInit();
```



```
    }
})();
```


VITA

Dan Mailman is a PhD candidate in Computer Science at the University of Tennessee at Chattanooga. His doctoral work focuses on gesture-efficient global language input methods: how people can enter Unicode-rich text—including Mandarin, IPA, and other multilingual writing systems—quickly and accurately on modern touch devices. His dissertation, *Gesture-Efficient Global Language Input Methods*, develops a practical engineering and evaluation framework for high-efficiency input, emphasizing measurable productivity, learnability for touch typing, and reduced gesture effort.

Alongside the research, he builds end-to-end prototypes that connect theory to usable software. These projects include a family of grid-based input interfaces (including the *μLex* framework and the *muCiYu* Mandarin prototype), with careful attention to interaction design, error handling, and reproducible evaluation. He also maintains a structured writing and tooling pipeline for the dissertation using LaTeX, XeLaTeX, and Quarto, and routinely develops supporting analyses and visualizations in R and Python.

Professionally, Dan works in education technology and has been affiliated with Great Minds PBC while completing the degree. He is also exploring commercialization paths for his input-method research through Language Genies, Inc., and he publishes technical and creative work

under the Studio MindStride umbrella. Outside of his research and writing, he enjoys cycling, rowing, and hosting film discussions.

