CommandBoard: Creating a General-Purpose Command Gesture Input Space for Soft Keyboards

Jessalyn Alvina, Carla F Griggio, Xiaojun Bi, Wendy E. Mackay

CommandBoard: Creating a General-Purpose Command Gesture Input Space for Soft Keyboards

Jessalyn Alvina[1] **Carla F. Griggio**[1] **Xiaojun Bi**[2] **Wendy E. Mackay**[1]
[1]LRI, Univ. Paris-Sud, CNRS
Inria, Université Paris-Saclay
F-91400 Orsay, France
[2]Department of Computer Science
Stony Brook University
Stony Brook, New York, USA
{alvina, griggio, mackay}@lri.fr; xiaojun@cs.stonybrook.edu

Figure 1. ***CommandBoard*** **creates a new** ***command gesture input space*** **above a soft keyboard. Users can:** ***a)*** **type 'happy' and use a dynamic guide to style it as bold;** ***b)*** **type 'brightn', draw an execute gesture and adjust the brightness slider;** ***c)*** **type 'sans', choose 'sans mono' and draw an execute gesture to change the font;** ***d)*** **type 'color', select yellow in the marking menu to change the brush color.**

ABSTRACT

CommandBoard offers a simple, efficient and incrementally learnable technique for issuing gesture commands from a soft keyboard. We transform the area above the keyboard into a *command-gesture input space* that lets users draw unique command gestures or type command names followed by execute. Novices who pause see an in-context dynamic guide, whereas experts simply draw. Our studies show that *CommandBoard*'s inline gesture shortcuts are significantly faster (almost double) than markdown symbols and significantly preferred by users. We demonstrate additional techniques for more complex commands, and discuss trade-offs with respect to the user's knowledge and motor skills, as well as the size and structure of the command space.

UIST 2017, October 22–25, 2017, Quebec City, QC, Canada
 ISBN 978-1-4503-4981-9/17/10…$15.00
DOI: https://doi.org/10.1145/3126594.3126639

ACM Classification Keywords

H.5.2. Information Interfaces, e.g. (HCI): User Interfaces

Author Keywords

Gesture Shortcuts; Gesture Typing; Mobile; Soft keyboards.

INTRODUCTION

Today's mobile devices offer a variety of functionality with the emphasis on communication, games, and information consumption. Text entry comprises about 40% of mobile activity [4], and is usually accomplished with a soft keyboard. Originally designed to imitate physical keyboards, soft keyboards consist of a set of keys that can be tapped to input text. Gesture keyboards [24] offer a significantly faster alternative by letting users draw through each successive letter of the word. The resulting gesture is interpreted by a sophisticated recognition algorithm, which, when combined with the relevant dictionary, suggests the mostly likely word completions.

Although very effective for producing text, these keyboards are not designed to issue commands. Instead, mobile devices rely on buttons, menus and dialog boxes, which restricts the available command set to what fits on a tiny screen. These

recognition-based command techniques are easy to learn, but rarely offer a path toward recall-based expert use, even though many users regularly spend hours interacting with their mobile devices. An exception is a markdown language, which styles text by surrounding it with special symbols, such as _hello_ to italicize *hello*. This approach is efficient on a physical keyboard, since it avoids leaving the keyboard to move the mouse, but requires two keyboard swaps on a soft keyboard. Worse, users have no easy way to learn the symbol mappings.

CommandBoard

Our goal is to offer users a simple, yet powerful method of issuing commands from a mobile device. We introduce *CommandBoard*, which transforms a soft keyboard into an efficient, yet learnable command-entry tool. We build on a key insight from the gesture keyboard, i.e. that the system can recognize users' gestures as they cross over the keys, and interpret them as text. *CommandBoard* generalizes this idea by creating an additional space, above the keyboard, for interpreting free-form gestures. We can think of this as extending a transparent interaction layer above the keyboard, where users can still see the usual display, but also issue gesture commands. This creates a general-purpose *gesture command input space* that supports a variety of command entry techniques.

CommandBoard takes full advantage of the limited screen real estate on a smartphone. Figure 2 shows four discrete interaction spaces. As with gesture keyboard, the lower space is dedicated to generating text input or emoticons via tapping, crossing or dwelling on keys. Users can also swap keyboards, e.g. numeric or emoticon. *CommandBoard* includes additional features (marked in green), which let users specify command names for later execution via a gesture.

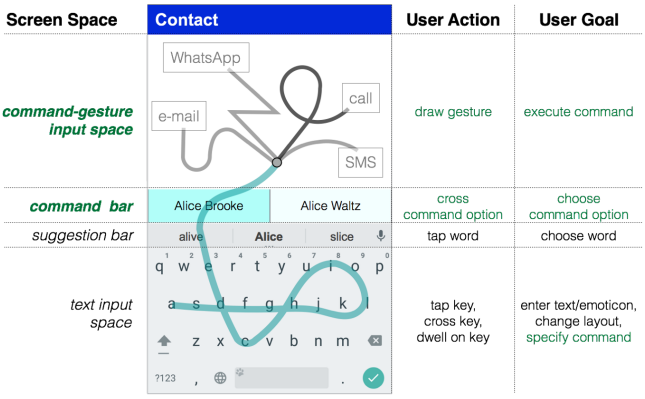

Figure 2. ***CommandBoard*** **specifies multiple command entry spaces. New methods (in green) include typing a command name followed by an** EXECUTE **gesture; crossing a suggested command in the** ***command bar*****; or executing a unique gesture in the upper** ***command gesture input space*****. An in-context dynamic guide shows gesture-command mappings.**

The gesture keyboard provides an optional *suggestion bar* in the middle where users tap to choose among suggested words. *CommandBoard* offers a similar *command bar* that users cross to choose among suggested commands. Finally, *CommandBoard* transforms the upper display area at the top of the screen into a *command-gesture input space* where users can draw an execute gesture to issue the current command name, or draw a unique command gesture. Since this overlay is transparent, users can see the underlying display, such as the current chat conversation.

Our goal is to transform a universal text-entry device – a soft keyboard – into a universal command generation device. We first describe related work on discrete and gesture-based command selection. We then describe the design and implementation of the *CommandBoard*, which comprises a set of techniques for issuing commands from a gesture keyboard. We then present the results of an experiment that compares expert-level performance with INLINE GESTURE SHORTCUTS to MARKDOWN SYMBOLS, and a qualitative study to assess user preferences. We demonstrate additional techniques, based on this idea, and conclude with directions for future research.

RELATED WORK

Selecting Commands via Discrete Actions

Graphical user interfaces usually offer menus and toolbars to execute commands. This allows non-expert users to quickly learn which commands are available, but makes large or complex command sets difficult to access. Small menus and tool bars allow users to quickly access common items, but do not help with large command sets, which may require extensive search and multiple physical operations to find the desired item [20]. Accessing a multi-level hierarchical menu forces the user to move through a multi-step process of selecting the appropriate category before finding the desired leaf.

Keyboard shortcuts let users select commands via a sequence of key presses. Although very efficient for experts, the process of transitioning from novice to expert can be very slow [14]. In fact, research from engineering psychology shows that the most commonly forgotten cognitive skill is performing multi-step action sequences [23].

Selecting Commands via Gestures

In contrast to performing a sequence of discrete actions, drawing a gesture is a perceptual motor skill that involves a continuous response, with little memory loss over long periods of time [23]. Researchers have explored leveraging continuous gestures to select commands. A classic example is a Marking Menu [12], which supports executing commands via directional strokes. FlowMenu [8] extends the hierarchical marking menu to include parameter adjustment of an item. For example, a user can select a zoom command and specify the zoom value in the sub-menu, or even type the value (when the desired number does not exist) without lifting the pen. Li's [15] world-wide deployment of a gesture-search system for smartphones demonstrated that users can successfully access their data via gestures, in their day-to-day mobile activities.

Appert & Zhai [2] investigated using gestures as an alternative to keyboard shortcuts. They found that gesture shortcuts are easier to learn and recall thanks to their spatial and iconic properties. OctoPocus [3] offers better support for learning gesture shortcuts, acting as a dynamic guide to help users follow the correct gesture template: If the user hesitates, OctoPocus appears, showing the remaining possible ways to finish the gesture. This highlights the need for progressive feedforward and feedback to support incremental learning, to help novices transition to expert users.

Augmenting Soft Keyboards

In response to the large demand for text entry on mobile phones, phone manufacturers are developing keyboard extensions that offer new capabilities, from suggesting emoticons to general search. The latest version of Google Keyboard (now called Gboard) [1] includes an in-context search engine. Users tap a button on the top-left of the keyboard to access the search engine, where they can directly type the search keyword, see the results, and share it. The TapBoard 2 [9] enables pointing via a soft keyboard, adding support to bimanual interaction. Arpege [7] supports multi-finger chord interaction, with dynamic guides to show novices where to place their fingers.

Previous research also explored ways of supporting expressivity with soft keyboards: KeyStrokes [19] visualizes the unique typing style of the user on a colorful canvas; Buschek et al. [5] render the user's typing variations into dynamic handwritten-looking output; and Expressive Keyboards [1] "recycle" users' gesture-typing variations to generate and control rich, expressive output.

As in any multitasking environment, switching between typing and issuing commands incurs interruption costs [22]. To reduce these costs, researchers have explored augmenting the keyboard with gesture-based commands. For example, Fuccella et al. [6] propose using a two-finger touch gesture directly on top of a soft keyboard which lets the user move the caret and thus select a specified text. Command Strokes [11] employ additional buttons, e.g. COMMAND to enable keyboard shortcuts on gesture keyboards [10, 24]. Users can simulate using control keys on a physical keyboard, e.g. drawing a gesture that passes through COMMAND then C to perform COMMAND+C. *CommandBoard* moves one step further by turning the space above the keyboard into a general-purpose command gesture space, to support more sophisticated command generation.

COMMANDBOARD TECHNIQUES

We are interested in extending the interaction capabilities of gesture keyboards. By taking advantage of the otherwise-unused input space above the keyboard, *CommandBoard* significantly increases the keyboard's power, letting users execute commands from the current command set, even if they are not visible on the screen.

Note that we do not seek to define a single 'best' method of issuing commands, since different commands perform better in different contexts [16], but rather to create a keyboard that offers users a choice, based on their cognitive and motor skills, as well as the size and organization of the current command set. *CommandBoard* exists in harmony with existing command-generation techniques, such as menus and buttons, but also offers novices the opportunity to transition into power users, to execute commands fluidly at their fingertips. Before describing the *CommandBoard* concept, we first describe the properties of gesture keyboards. We then show how *CommandBoard* leverages these to provide users with a variety of simple, yet powerful command invocation techniques.

[1] https://play.google.com/store/apps/details?id=com.google.android.inputmethod.latin&hl=en

Gesture Keyboards

Gesture keyboards let users either tap each letter to enter text or gesture-type by drawing a line that connects all the letters (a word-gesture). Word-gesture recognition requires a multi-channel recognition engine [10], where the drawn shape is first compared to an "ideal" shape, i.e. from middle-point to middle-point of each key. The recognition engine then produces a list of word candidates. This list is shortened based on the actual location of the drawn word-gesture on the keyboard and weighted based on the language information. The recognition engine also considers temporal features: If the user slows down at a letter, the recognizer weights the word candidate higher.

This recognition process is conducted progressively as the user moves her finger: at each touch, the gesture keyboard generates a list of at least four suggested words. The first is treated as the final result, the next three are displayed in the suggestion bar. The gesture keyboard may also auto-complete the current word-gesture, even before the user reaches the last letter of the intended word.

If a word-gesture is drawn outside the keyboard space, the gesture keyboard captures the touch event but stops recognizing the word. When the user's finger is released, the word output is cancelled and not rendered. By contrast, *CommandBoard* interprets a wide variety of gestures drawn above the keyboard. The next sections describe the two most basic techniques: TYPE-AND-EXECUTE and INLINE GESTURE SHORTCUTS.

Type-and-Execute Commands

Although novices may need to search through menus to discover the available commands, frequent users are usually familiar with both the commands and their names. Navigating through menus can be time-consuming, especially if the user forgets where the desired command is classified within a hierarchical menu. Some graphical user interfaces offer a *search bar* where typing the keyword or command name displays its location in a pull-down menu, if the command exists. Clicking on the search result issues the command, as if it had been selected from the menu. *CommandBoard* offers a similar function by letting the user type any command name from the keyboard, and then execute it directly by drawing the execute gesture in the display area above the keyboard.

Keyword Search Since *CommandBoard* co-exists with traditional menus, we have prior knowledge about the current set of command names or *command-gesture input space*. When a user gesture-types a word, *CommandBoard*'s TYPE-AND-EXECUTE technique examines the first four words suggested by the keyboard recognition engine to see if any is a keyword in the command space. The TYPE-AND-EXECUTE technique treats each element of a compound command name as a search keyword. For example, both "line" and "spacing" can be used to find LINE SPACING. Users need only type the first unique letters of a long command name and the system will suggest the full command. For example, typing 'brightn' produces the BRIGHTNESS command in the command bar, which can then be invoked by performing the execute (/\) gesture (Figure 1b).

Command Preview If the keyword search is successful, the TYPE-AND-EXECUTE technique displays a preview: the full command name appears at the top of the screen. The keyboard continues to recognize the word-gesture as the user types. Thus, when the preview appears, the TYPE-AND-EXECUTE technique stores the keyword so that even if the recognized word changes as the user slides her finger upward, the command keyword remains the same. If the user continues gesture-typing, the preview disappears. If the user releases her finger within the keyboard space, the word appears as normal text.

Command Execution If, after typing a recognized command name, the user continues to slide her finger upward, she enables the command-gesture input space. If she now performs a /\ gesture, the TYPE-AND-EXECUTE technique will execute the corresponding command. This allows the user to perform any command directly from the keyboard, as long as she already knows the command name. She need not learn any special commands beyond the execute gesture.

We designed the execute gesture specifically so that it would not interfere with the GBoard's technique for cancelling gestures. (The user cancels the current word by sliding her finger into the space above the keyboard and releasing it.) By contrast, *CommandBoard*'s execute gesture is designed to move up and then down, explicitly change direction, to reduce the risk of issuing unintended commands. The following examples illustrate various applications of *CommandBoard*'s TYPE-AND-EXECUTE technique:

Text Editor Application

Most text editing applications for mobile devices, such as Google Docs, offer only a limited number of commands. The process is also cumbersome: Selecting a menu command requires hiding the keyboard, navigating to and executing the command, then closing the menu and bringing back the keyboard, all before continuing to type.

The TYPE-AND-EXECUTE technique simplifies command selection for text editors. The user can type the name of any menu item, as if it were a search word, and then execute it directly. For example, Figure 1c shows the user typing the word 'sans', then sliding her finger above the keyboard to perform the execute gesture, at which point TYPE-AND-EXECUTE applies the SANS MONO font to the selected text.

CommandBoard's TYPE-AND-EXECUTE technique also lets users type sub-menu names, and display their items in the command bar located above the suggestion bar. This is particularly useful when the menu item cannot be typed, for example the numbers shown in Figure 3. The user types 'line' and a preview for the LINE SPACING sub-menu appears (Figure 3a). The menu items then appear on the command bar. She sets the LINE SPACING value to 1.2 by crossing through it in the command bar and then performing the execute gesture (Figure 3c).

Doodle Application

Many mobile applications, such as iMessage and SnapChat, let users 'doodle' on their messages. *CommandBoard*'s TYPE-AND-EXECUTE technique lets users specify brush properties, such as changing the color or brush type, with a marking menu [13]. For example, in Figure 1d, the user types 'color'. She slides her finger upward to reveal the COLOR marking menu, which brings up various brush color, and then moves down-left to select YELLOW.

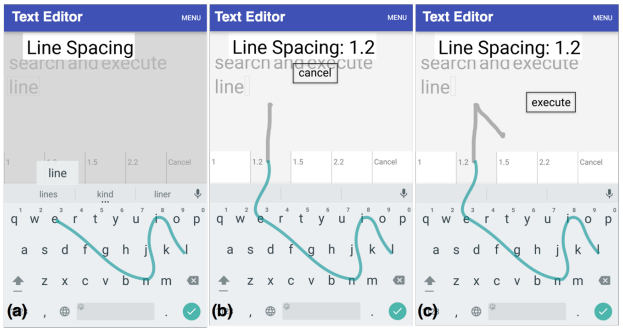

Figure 3. Typing an existing command name a) displays a preview. Here, LINE SPACING is a sub-menu, so its menu items appear in the command bar. After sliding across the command bar, b) performing a | gesture cancels the current word, whereas c) performing a /\ gesture executes the command.

Note that a challenge in combining *CommandBoard* with a marking menu is deciding when a gesture should be interpreted as a 'mark'. One solution is to require users to begin from the middle of the screen, which, given the phone's limited screen real estate, would ensure sufficient space to move in all directions.

Parameterization

CommandBoard's TYPE-AND-EXECUTE technique can also be combined with sliders to parameterize commands. For example, in Figure 1b, the user begins gesture-typing the 'brightness' command. Upon performing the /\ gesture, a slider appears, with the handle under her finger. Moving along the x-axis (left and right) adjusts the screen's brightness, whereas moving along the y-axis (up and down) moves the slider's position on the screen.

Inline Gesture Shortcuts

CommandBoard's INLINE GESTURE SHORTCUTS let users invoke gesture shortcuts from the keyboard *as they type*. Instead of typing the command name, the user types the object of the command, for example, the text to be styled. The user then slides her finger above the keyboard, pausing to bring up the dynamic guide that shows the current set of possible commands (see Figure 1a). Users can benefit from motor memory to recall these gestures. As they become experts, they can perform the command gesture directly, without pausing for the guide. The following examples illustrate various applications of *CommandBoard*'s INLINE GESTURE SHORTCUTS technique:

Chat Application

Although soft keyboards are not specifically designed to support command input, users often use markdown languages to issue text styling commands. For example, typing an asterisk before and after a word (*help!*) produces (**help!**). Markdown commands are effective keyboard shortcuts when using physical keyboards, because users avoid lifting their hands to move the mouse. On soft keyboards, however, markdown languages force users to switch from the alphabetic to the symbolic keyboard, disrupting their writing flow.

Unfortunately, issuing styling commands as text can be cumbersome, especially if done often or multiple times in a row. *CommandBoard*'s INLINE GESTURE SHORTCUTS offer a more efficient alternative, by executing a specialized gesture directly from the keyboard. In Figure 1a, the user wants to style the word 'happy'. After writing it, she moves into the upper area, pauses to see several styling alternatives, and then follows the pigtail to execute the **bold** command. As before, the dynamic guide offers a path to help users develop their motor memory, becoming expert over time.

CommandBoard's markdown language is designed to be similar to those in existing applications, where the user writes a symbol before and after the word. Thus, writing 'happy' followed by a pigtail gesture generates '*happy*' on the text field buffer, which then will be rendered as **happy**. This enables users to style more than one word, by moving the caret in between the markdown symbols and insert more words.

Contacts Application

Most phones have a CONTACTS application that lets users tap on a contact to view the person's details and then call, send an SMS, or use another communication app to communicate with that person. Users can also access a person's details by typing her name in the search bar.

CommandBoard's INLINE GESTURE SHORTCUTS let users issue commands from within the search bar, as soon as the desired result appears. For example, if 'Mom' exists in the contacts list, the user can gesture-type 'Mom', then slide up to the upper space and draw a pigtail gesture to call her. If the search produces multiple contacts the command bar displays the alternatives. For example, Figure 2 shows two contacts: *Alice Brooke* and *Alice Waltz*. Here, the user crosses through the *Alice Brooke* contact and then draws a pigtail gesture to call her.

Note that both TYPE-AND-EXECUTE and INLINE GESTURE SHORTCUTS are designed for efficiency, and rely on an experienced user's ability to either recall the command name, or the associated gesture. Each technique provides scaffolding to help novice users learn, including the TYPE-AND-EXECUTE's command bar and the INLINE GESTURE SHORTCUTS's dynamic guides. However, these techniques can only display a small number of commands, which makes them most useful when the current context significantly limits the command space.

EVALUATION

Standard mobile devices use icons, buttons and menus to access functionality, because these are easy for novice users to recognize and use. However, many experts prefer the efficiency of command-line interfaces, even though they require learning and subsequent recall of command names and syntax. One of the goals of *CommandBoard* is to bridge the gap between these two approaches, by supporting both recognition-*and* recall-based interaction, with a smooth transition between novice and expert use.

We begin by examining "expert" behavior, with a focus on the efficiency of the technique. We use a common experimental strategy for simulating expert performance: we show the participant the correct action so that we measure only performance, not confounded by unmeasured memory issues. We sought an ecologically valid domain for testing *CommandBoard*'s ability to support both recognition and recall. We chose the markdown commands available in chat applications such as *WhatsApp* and *Slack*, since users can style their text by typing markdown symbols before and after the text (recall), with a "cheatsheet" in the menu if they forget the symbols (recognition). For evaluating expert behavior, MARKDOWN SYMBOLS offer a fairer, more realistic comparison than standard pull-down menus, which would be even slower. In the INLINE GESTURE SHORTCUTS condition, users write a word and then draw a command gesture directly from the keyboard to style it, whereas in the MARKDOWN SYMBOLS condition, users type markdown symbols before and after the word to be styled. Although not a primary goal, we are also interested in whether or not users begin to learn gesture-command mappings, simply by using the technique. Our research questions include:

1. Are INLINE GESTURE SHORTCUTS faster and more accurate than text-based MARKDOWN SYMBOLS?
2. Do users prefer *CommandBoard*'s INLINE GESTURE SHORTCUTS?

METHOD

We conducted a two-part study, using a within-participants design, to compare *CommandBoard*'s INLINE GESTURE SHORTCUTS technique to MARKDOWN SYMBOLS (see Figure 5). Part A is a one-factor experiment that compares speed and accuracy of expert users using these two techniques. Part B is a qualitative study designed to assess participants' preferences as well as incidental learning with respect to each technique. Part B follows Part A, with the same participants, hardware and software.

Participants

We recruited 12 right handed participants (4 women, 8 men), aged 23-41. All use mobile phones daily. Two gesture-type daily; the others are non-users. Three sometimes use markdown symbols in existing chat applications; the rest do not.

Hardware and Software

We used two LG Nexus 5X (5.2" display) smartphones, running Android 7.1.

We implemented *CommandBoard* as an Android application that lets users issue text-styling commands with INLINE GESTURE SHORTCUTS, using the native Android gesture recognizer. The INLINE GESTURE SHORTCUTS technique requires the user to draw through the letters of the indicated word on the keyboard. *CommandBoard* recognizes the word, and renders it on the screen. If the user continues the stroke above the keyboard, a semi-transparent overlay appears and the stroke is interpreted as a command gesture. The overlay displays an OctoPocus-like [3] dynamic guide indicating the gestures associated with possible styling commands. Lifting the finger applies the recognized gesture-command to the word output and the overlay disappears. Note: We removed OctoPocus' dwell delay in the experiment to avoid confounding time measures. We also implemented the MARKDOWN SYMBOLS technique, which requires the user to type a specified symbol before and after the word to be styled.

Command-Set Design: We created a command set consisting of six text-styling commands: *underline*, *monospace*, *big*,

small, *outline*, and *gradient color* and mapped them to INLINE GESTURE SHORTCUTS and MARKDOWN SYMBOLS. The INLINE GESTURE SHORTCUTS set consists of six gestures chosen from [2] (see Figure 4). We ensured that these gestures do not overlap when displayed together in an OctoPocus-style dynamic guide using [17]. The MARKDOWN SYMBOLS set consists of six characters chosen from the second row of the symbol keyboard: @, #, $, %, &, and +. We ensured that none overlap with existing chat symbols from, e.g., *WhatsApp* and *Slack*. Mappings between gestures and markdown symbols are counter-balanced across participants using a Latin square.

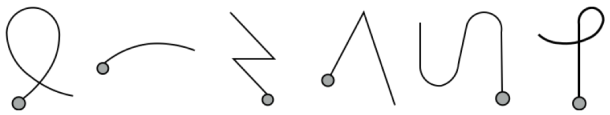

Figure 4. Gesture set: Grey circles indicate where to begin drawing.

Phrase Set Design: We constructed two sets of 24 three-word phrases drawn from the Oxford Dictionary[2]. The middle words are each four-five letters long, and end in 24 different letters of the alphabet (we exclude 'j' and 'q'), to ensure gesture starting points are distributed evenly across the keyboard. We also balanced angles between stroke segments across the sets, to avoid unwanted performance effects [21, 1]. Eight words include acute angles, e.g. "menu"; eight include at least one obtuse angle, e.g. the 'agi' in "magic"; and eight include only 0° or 180° angles, e.g. "power".

We used the 24 middle words to create two sets of 24 three-word phrases. We created two phrases around each middle word, using three-to-six letter surrounding words that make sense when read together as a phrase. For example, the first set includes 'play *video* games', and the second set includes 'some *video* clips'. We distributed the first set of 24 phrases across the practice and experimental conditions of the experiment, and distributed the second set across the pre- and post-test conditions of the study. We counter-balanced for order within and across participants using a Latin square.

Procedure

Figure 5 shows the study design. Part A consists of four conditions, each comprised of two blocks of six trials, grouped by TECHNIQUE. Part B consists of a single recomposition task where users can freely choose the desired technique.

	PART A			PART B	PART A
Condition	Practice	Experimental	Pre-test	Recombination Task	Post-Test
Feedback?	yes	yes	no	yes	no
Gestures	6 trials	6 trials [x3]	6 trials	12 trials	6 trials
Symbols	6 trials	6 trials [x3]	6 trials	(user chooses technique)	6 trials

Figure 5. Part A (Experiment): Each condition (Practice, Experiment, Pre-test, and Post-test) includes two blocks of six trials, one per technique, with three replications in the experimental condition. Part B (Study): Participants recompose 12 of their own messages, with free choice of technique.

Part A: Trial Description

Each trial begins by displaying a three-word phrase, with a styled middle word, e.g. play *video* games. The participant presses START, then retypes the phrase, using the indicated technique to style the middle word. This simulates the process of issuing styling commands during the flow of writing. To simulate "expert" behavior, each trial includes explicit instructions as to how to execute the command, removing the need for recall memory. Participants may preview styling results.

[2]https://en.oxforddictionaries.com/

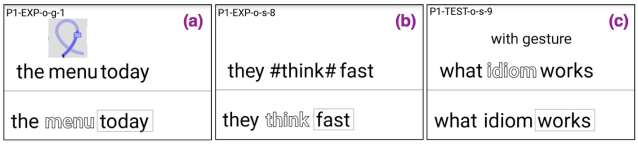

Figure 6. Each trial presents instructions above the line, and the result below the line. Practice and Experimental conditions present a) INLINE GESTURE SHORTCUTS to draw, or b) MARKDOWN SYMBOLS to type, to issue the specified styling command. Pre- and Post-Test conditions present c) the styled text to reproduce with the specified technique, with no feedback.

Practice and experimental trials display the correct styling command, either the gesture to draw (Figure 6a, INLINE GESTURE SHORTCUTS condition) or the symbols to type (Figure 6b, MARKDOWN SYMBOLS condition). This simulates expert performance by eliminating errors due to forgetting a gesture shape or markdown symbol. Conditions are separated by short breaks.

Practice Condition

Participants are exposed to two practice blocks, one per TECHNIQUE (INLINE GESTURE SHORTCUTS and MARKDOWN SYMBOLS). Each block involves typing six three-word phrases, and styling the middle word. Each trial shows which INLINE GESTURE SHORTCUTS or MARKDOWN SYMBOLS to use. In the INLINE GESTURE SHORTCUTS condition, the gesture template appears as soon as the participant's finger leaves the keyboard. Participants can retype phrases as often as they like, until they are comfortable performing the task quickly and reliably. An error message appears if they forget to apply the style or make a typing or styling error. Pressing CLEAN restarts the trial; DONE moves to the next trial.

Experimental Condition Participants are exposed to two six-trial blocks, one per TECHNIQUE (INLINE GESTURE SHORTCUTS and MARKDOWN SYMBOLS), for a total of 12 trials. Experimental trials are identical to practice trials, except that participants retype and style each three-word phrase three times (three replications), to provide a stable performance measure.

Pre- and Post-test Conditions Participants begin with two blocks of six trials, one for each TECHNIQUE (INLINE GESTURE SHORTCUTS and MARKDOWN SYMBOLS), counter-balanced for order within and across participants. Each trial displays the phrase to be typed including the styled the middle word (see Figure 6c). Participants reproduce the styled phrase with each technique, with no feedback. This serves as a baseline measure of styling command recall.

The pre- and post-test conditions are identical, but use phrases from the alternate phrase set. The pre-test offers an initial assessment of learning, how much they remember immediately after their first exposure to each technique. The post-test offers a second assessment, based on more extensive practice during the recomposition task.

Part B: Recomposition Task
After completing the Pre-Test condition in Part A, participants are asked to perform a more open-ended set of tasks, in order to assess their overall preferences for each technique. For greater ecological validity, we asked participants to check their smart phones and choose 12 recent messages to retype, avoiding ones they felt were too personal. Participants were free to change the text as they liked. We then asked them to recompose these 12 messages, using either technique to style at least one word. We provided a 'cheat sheet' with the relevant markdown symbols for the MARKDOWN SYMBOLS technique, and displayed a dynamic guide with the relevant gestures for the INLINE GESTURE SHORTCUTS technique.

Measures

Input Time We measure *INPUT TIME* in seconds for the phrase and each word-output, refered to as: WO_1, WO_2, and WO_3. Note that WO_2 includes inserting the two markdown symbols. This measure allows us to assess the gesture-typing time for both INLINE GESTURE SHORTCUTS and MARKDOWN SYMBOLS.

Gesture-Typing and Command Selection Time The participant must gesture-type the middle word and style it using INLINE GESTURE SHORTCUTS or MARKDOWN SYMBOLS (i.e. WO_2). We capture the times spent in each sub-activity. We measure Command Selection Time (*COMMAND TIME*) and Gesture-Typing Time (*TYPING TIME*).

INLINE GESTURE SHORTCUTS: We measure the time spent leaving the keyboard and drawing the gesture (*COMMAND TIME*). If a participant crosses the top border of the keyboard, below the suggestion bar, at $event_k$, then *COMMAND TIME* and *TYPING TIME* are as follows:

$$\textit{COMMAND TIME} = t(event_N) - t(event_k)$$
$$\textit{TYPING TIME} = t(event_k) - t(event_0)$$

MARKDOWN SYMBOLS: We measure the time spent writing the symbols before and after the word (*COMMAND TIME*) for WO_2. Given an input I is a sequence of touch *event*s, where $I = \langle event(x,y,t,\texttt{action})_{0\ldots N}\rangle$, if a participant starts gesture-typing the word at $event_i$ (tagged as `down`) and lifts her finger at $event_j$ (tagged as `up`) in WO_2, then *COMMAND TIME* and *TYPING TIME* are as follows:

$$\textit{COMMAND TIME} = t(event_i) - t(event_0) + t(event_N) - t(event_j)$$
$$\textit{TYPING TIME} = t(event_j) - t(event_i)$$

Gap Time We assess how long participants spend switching from writing a regular word (WO_1) to a styled word (WO_2) and back again (WO_3). Given that an input I is a sequence of touch *event*s, $I = \langle event(x,y,t,\texttt{action})_{0\ldots N}\rangle$ where t is the timestamp, we measure *gap* time between each word-output as follows:

$$gap(WO_i, WO_{i+1}) = t(WO_{i+1}.event_0) - t(WO_i.event_N)$$

Errors We count three types of error: typographical errors (*TYPING ERRORS*), incorrect symbols or gestures (*STYLING ERRORS*), or forgetting to style the middle word (*MISSING ERRORS*). Note that *TYPING ERRORS* and *STYLING ERRORS* can occur in the same trial. A trial is considered correct when it has no errors.

Data Collection

We log all touch events and the recognized word output for each trial. We tag each touch event with one of five `actions`: `shift`, `tap`, `down`, `move`, and `up`. `tap` involves pressing a key and `shift` involves holding down the keyboard shift key. The remaining actions identify the start (`down`), drawing phase (`move`) and completion (`up`) of a gesture. These measures allow us to compute speed, movement time and errors for each technique.

Participants answer a five-point Likert-style questionnaire to assess their perceived accuracy, speed, ease-of-use, confidence, comfort, and enjoyment of each technique. We also take observational notes and debrief participants, with a particular focus on what the participants liked and disliked about the techniques and their strategies for styling their text.

RESULTS

Experiment

We collected a total of 432 experimental trials (12 PARTICIPANT × 2 TECHNIQUE × 6 trials × 3 replications). We removed one trial (P4) who gave up after repeated typing errors on the third word of one phrase. After determining we had no unwanted significant effects from the word sets, we ran a one-way repeated-measures analysis of variance for factor TECHNIQUE, followed by Tukey HSD tests for post-hoc comparisons.

Input Time
The overall *INPUT TIME* (trial completion time) is significantly affected by TECHNIQUE ($F_{1,11} = 86.9$, $p < 0.0001$). This is due primarily to styling the middle word (WO_2), as shown in Figure 7.

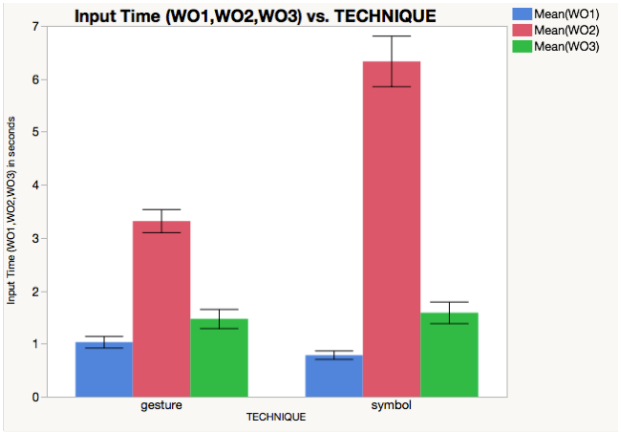

Figure 7. Average time spent entering each word. WO_2 is the styled word. commands are significantly faster: almost double.

Gesture-Typing and Command Selection Time
On average, participants spent significantly more time styling words with MARKDOWN SYMBOLS (mean 6.3s) than with INLINE GESTURE SHORTCUTS (3.3 seconds), with $F_{1,11} = 71.1$, $p < 0.0001$. When we break apart *INPUT TIME* for WO_2 into time to select the command (*COMMAND TIME*) and time to gesture-type it (*TYPING TIME*), we find that participants spend significantly

longer writing symbols (mean *COMMAND TIME* =5.8s) than drawing gestures (mean *COMMAND TIME* =1.5s) [$F_{1,11} = 177.6$, $p < 0.0001$] (Figure 8).

However, they spend more time gesture-typing the styled word when using INLINE GESTURE SHORTCUTS (mean *TYPING TIME* =1.8s) than MARKDOWN SYMBOLS (mean *TYPING TIME* =0.6s) [$F_{1,11} = 68.3$, $p < 0.0001$]. This may be an artifact of the experimental design, since participants slowed down to check that they had gesture-typed the correct word, before drawing the styling gesture. In the long run, this may actually benefit the INLINE GESTURE SHORTCUTS technique, because slowing down improves the recognition process with gesture keyboards [10]. Recognized words are less likely to change when users slide into the command gesture input space.

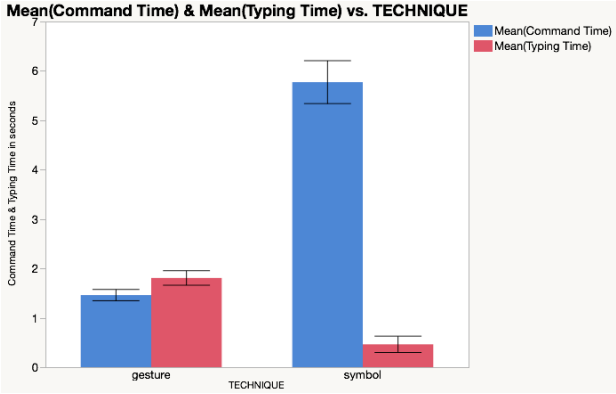

Figure 8. Average time spent gesture-typing (*TYPING TIME*) and issuing the command (*COMMAND TIME*). Participants drew quickly with INLINE GESTURE SHORTCUTS, but took significantly longer inserting MARKDOWN SYMBOLS.

Gap Time

When the participants switch from writing the first word to applying a styling command to the second word, the gap duration (*GAP* (WO_1, WO_2)) is significantly longer for MARKDOWN SYMBOLS (mean=1.9s) than for INLINE GESTURE SHORTCUTS (mean=1.2s) [$F_{1,11} = 49.7$, $p < 0.0001$]. This suggests that participants needed more time to consider which key to press when selecting markdown symbols, i.e. searching and pre-planning. However, when participants finish applying the styling command to the middle word, they spend significantly less time writing the third word when using MARKDOWN SYMBOLS (mean *GAP* (WO_2, WO_3) 0.9s) than when using INLINE GESTURE SHORTCUTS (mean *GAP* (WO_2, WO_3) 1.5s) [$F_{1,11} = 128.4$, $p < 0.0001$]. In the MARKDOWN SYMBOLS condition, they can already see if they have applied the correct command as they press the SPACE bar, whereas with INLINE GESTURE SHORTCUTS, they must check again after releasing their finger. This would be improved by displaying a progressive preview at the end of the dynamic guide, but was not made available during the experiment.

Errors

Participants made significantly fewer styling errors with INLINE GESTURE SHORTCUTS (mean *STYLING ERRORS* = 0.09) than with MARKDOWN SYMBOLS (mean *STYLING ERRORS* = 0.36), [$F_{1,11} = 13.7$, $p = 0.0035$]. However participants using INLINE GESTURE SHORTCUTS were somewhat more likely to forget to actually style the word – INLINE GESTURE SHORTCUTS (mean *MISSING ERRORS* = 0.3) versus MARKDOWN SYMBOLS (mean *MISSING ERRORS* = 0.04), [$F_{1,11} = 26.7$, $p = 0.0003$]. This is probably an artifact of the experimental setting, since in actual use, users would not 'forget' to style a word if they wanted to. We did not find a significant effect of TECHNIQUE on accuracy [$F_{1,11} = 49.7$, $p = 0.47$]. which suggests that using gestures to style text does not interfere with typing accuracy.

Preferences Study

Pre- and Post-test Results

We ran a one-way repeated measures analysis of variance for factor TECHNIQUE to compare *STYLING ERRORS* during the Pre- and Post-test conditions. We found a significant interaction effect [$F_{1,11} = 4.4$, $p = 0.0375$] for *STYLING ERRORS*. In the Pre-test, the average *STYLING ERRORS* for INLINE GESTURE SHORTCUTS and MARKDOWN SYMBOLS are 0.52 and 0.32, respectively. In the Post-test, the average *STYLING ERRORS* for INLINE GESTURE SHORTCUTS and MARKDOWN SYMBOLS are 0.35 and 0.38, respectively.

Prior to the pre-test condition, participants had practiced both techniques, but always with a direct indication of how to perform the gesture or what symbols to type. The pre-test was the first time that participants had ever tried executing the commands without help. Participants remembered half the gestures and two thirds of the symbols from the previous practice and experiment condition. The post-test was given after participants had experimented with their choice of technique to recompose their own text, and participants remembered almost two thirds of the gestures. This suggests that we should study longer term use of *CommandBoard*'s INLINE GESTURE SHORTCUTS technique, to see how well it supports incremental learning over time.

Recomposition Task Results

Although given a choice between using MARKDOWN SYMBOLS or INLINE GESTURE SHORTCUTS, all participants chose gestures. They ignored the cheat-sheet showing all markdown symbols and their resulting styles. P11 was the exception, but he only looked at the cheat-sheet to get inspiration from the style examples. We observed three strategies when styling words with gestures: thinking of a style first, and then using OctoPocus to follow the corresponding gesture; activating OctoPocus first, and then deciding on a style from the options; and performing a learned gesture to apply a style with no hesitation.

A few participants explained the rationale behind their styling. P2 recomposed a text message to his wife with a shopping list, and he used all available styles to highlight the ingredients they had to buy for a salad. P8 associated word categories to styles: big meant positive or a lot, small meant negative or uncertain, underline was important or certain, outline and gradient were for special words. P12 also assigned meanings to different styles: gradient for opinions, outline for time-related words, underline for important words, and big for emphasisis in general: "*Big is the most useful.*" P11 on the other hand cared less about the different styling options, and mostly focused on emphasizing important words: "*I think I didn't really want to choose a specific [style], I just wanted to add an effect on it so it looks different from other words.*"

Self-reported Quantitative Measures

Participants were asked to rate six statements on a 5-point Likert scale, from strongly disagree (1) to strongly agree (5). The statements asked whether the current technique helped them to style text: a) accurately, b) quickly, c) easily, d) confidently, e) comfortably, and f) enjoyably. Table 1 lists the medians of each question for both techniques. An analysis using a Friedman test showed that participants reported significantly stronger agreement for INLINE GESTURE SHORTCUTS compared to MARKDOWN SYMBOLS on five statements: ACCURATELY ($p = .34$, $\chi^2(1) = 4.5$), QUICKLY ($p = .007$, $\chi^2(1) = 7.36$), EASILY ($p = .11$, $\chi^2(1) = 6.4$), COMFORTABLY ($p = .002$, $\chi^2(1) = 10$) and ENJOYABLY ($p = .001$, $\chi^2(1) = 11$).

Statement	Symbols	Gestures
ACCURATELY*	2.5	4.0
QUICKLY*	2.0	4.0
EASILY*	2.0	4.0
CONFIDENTLY	2.5	4.0
COMFORTABLY*	2.0	4.0
ENJOYABLY*	2.0	4.5

Table 1. Participant ratings of how each technique helped them to style text (median values; * indicates a significant difference). Participants significantly preferred gestures in all categories except 'confidently'.

User Preferences and Debriefing

The final questionnaire asked participants to rate their preference between the two techniques on a 5-point scale (from strong preference for MARKDOWN SYMBOLS to strong preference for INLINE GESTURE SHORTCUTS). All participants preferred *gestures*: 10 indicated a strong preference, 2 indicated some preference.

Six participants expressed their preference in terms of *typing flow*, explaining that INLINE GESTURE SHORTCUTS best supported styling without interrupting their text composition process. P2 commented "*I didn't use the symbols at all in the chat. It's troublesome to have to switch the keyboard, doing it in the beginning and at the end. It really breaks the flow of the writing. While with the gesture, it's always there, I can pick what I want on the go.*" P9 wrote "*It's enjoyable to use and in coherent with using gestures to type words.*"

Participants differed with respect to recognition and recall. Four participants found INLINE GESTURE SHORTCUTS easier to recall than MARKDOWN SYMBOLS: "*I used big, small, underline in the recomposition task, so I remember them*" (P1). However, four participants had difficulty recalling the INLINE GESTURE SHORTCUTS mappings: P9 said "*the paths of gestures are difficult to link with their meanings*", and P6 said "*If the gestures are well designed or designed by the user himself, it could be quite natural.*" Two participants felt more comfortable creating mnemonics for MARKDOWN SYMBOLS rather than INLINE GESTURE SHORTCUTS, despite their overall preference for INLINE GESTURE SHORTCUTS: "*It's easier to remember the symbols for each type (+ for big; $ for the underlined because of the line in the S).*" Three participants also appreciated the convenience of recognizing gestures with OctoPocus rather always having to recall them: "*this is nice, I don't have to remember and just follow [the OctoPocus guideline].*"

Finally, we asked participants to suggest other applications for INLINE GESTURE SHORTCUTS. Four participants suggested using gestures to add emojis: "*I have 5-10 smileys that I always use, so I think it'd be nice if I can use the gesture to get it. Because it's bothersome having to change to another keyboard view (emoticon), so if I can do it with the gesture it'd be cool.*" (P3). Two thought of command shortcuts: "*If you like a webpage, you could do a special gesture to bookmark it. To refresh the page, you could use a circular gesture, etc.*" (P2). Other suggested applications were changing lines, replacing the enter key, taking notes and changing fonts.

DISCUSSION

On mobile devices, users issue commands via buttons, menus and dialog boxes and enter text with soft keyboards. Given the sheer amount of time users spend with their smartphones and other mobile devices, it seems odd that they willingly accept such limited forms of interaction. *CommandBoard* provides an additional set of interaction techniques, offering users both power and simplicity when executing commands. *CommandBoard* repurposes the unused output space above the keyboard to accept gestures that invoke commands; extends gesture keyboards with command gestures, without disrupting existing command invocation techniques; and makes it easy for users to discover gesture-command mappings.

We view *CommandBoard* as a strategy for transforming mobile devices into powerful, personalized tools, with which users can benefit from a variety of new command entry techniques, using text, gestures or both. By building upon the gesture keyboard, we leverage its powerful machine-learning algorithms, and offer an easy way to incorporate successful gesture-based command invocation techniques from the research literature.

CommandBoard offers a variety of alternatives, depending upon the task, the user's cognitive and motor skills, and the size and structure of the current command space. *CommandBoard* can also handle parametric commands, such as typing 'brightness' followed by the execute gesture (/\) displays a slider with continuous control of the screen brightness level. It could also be combined with the *Expressive Keyboard* [1], which would allow gestures to dynamically modify command parameters.

Although we expected that *CommandBoard* would perform better than current markdown commands, we were surprised by the size of the effect (approximately twice as fast) and by how much the participants preferred it over standard markdown commands. We believe this is because users can fluidly style their text without interrupting the flow of their typing. Users not only avoid switching modes, but also avoid selecting text, the most time-consuming aspect of text editing [6].

One important issue is how best to support the transition from novice to expert use. Expert users must not only know that a command exists, but must also be able to recall either the command name, or the associated gesture shortcut. We provide several types of dynamic guides to help novices learn, and to help intermittent users when they forget. For example, we present likely commands in the command bar, or show gesture

paths, either Marking Menu-style directions [12] or free-form OctoPocus-style gestures [3].

The pre- and post-test results from the experiment indicate that users can easily learn gesture commands simply through the process of using them. We expected relatively low post-test scores, since users had only limited experience with the gestures during the practice and experimental conditions. Even so, users clearly made fewer errors in the post-test, which suggests that even limited experience can improve gesture recall. We should be able to further reduce learning time and enhance the transition from novice to expert performance by letting users define their own memorable, yet recognizable gestures [18], e.g. with [17]. In future work, we plan to conduct a longitudinal field study of *CommandBoard*, in order to more thoroughly investigate this transition from novice to expert.

The experiment restricted *CommandBoard*'s INLINE GESTURE SHORTCUTS to styling one word at a time. For example, the 'happy'+pigtail gesture generates **happy**. However, sometimes users want to apply a style to multiple words. One option would be to combine *CommandBoard* with other advanced text selection techniques, such as selecting a phrase with a two-finger gesture on top of the keyboard [6]. Gesture grammars can also combine command gestures with selection-scope gestures. For example, in TYPE-AND-EXECUTE, after sliding her finger to the input space above the keyboard, the user could specify the scope of the selection with a marking menu that includes last word, last sentence, last paragraph, and select all.

All gesture-based menu systems, including Marking Menus and OctoPocus, run into visual overload problems when forced to display more than about 16 menu items at a time. This is commonly addressed by creating hierarchical menus or by restricting the command set to a more limited context. *CommandBoard* faces these same limitations, but they can be partially mitigated when *CommandBoard* is used in conjunction with other recognition-based techniques. On the other hand, using *CommandBoard* to type commands on the keyboard and then select a parameter in the gesture-input space can help users access the full range of available commands.

Ideally, mobile application developers should be able to use *CommandBoard* as a service i.e. library when developing an application. The gesture keyboard captures the gesture input from the users and recognizes the word, which is then processed by the underlying application. The developers define the command set from the current set of menu items, the *CommandBoard* technique implementation, and the basic gesture-command mappings in the underlying application.

CONCLUSION

We present *CommandBoard*, which lets users gesture-type commands directly from a soft keyboard on a mobile device. We transform the otherwise unused area *above* the keyboard into an alternate, gesture-based input space. *CommandBoard* is a general approach for adding gesture-based commands to a soft keyboard, that builds upon gesture-typing to offer several different techniques for generating commands.

This paper proposes two basic techniques that address different trade-offs. The user can: 1. gesture-type a known command name followed by an execute gesture; or 2. move from the gesture-type keyboard directly to the *command-gesture input space* above, to execute a unique command gesture.

When practiced by experts, both techniques require the user to recall either the command name, or its associated gesture. However, *CommandBoard* also provides a path from novice to expert use, by offering two types of dynamic guides. The command bar offers suggested command names that users can select by crossing through, and the OctoPocus-style dynamic guide offers progressive feedforward, to suggest alternative command-gesture mappings.

We ran an experiment to compare *CommandBoard*'s command invocation to a conventional markdown language for styling text. We found that participants are not only significantly faster with *CommandBoard* (almost double), but also participants significantly preferred *CommandBoard* (unanimously).

We also demonstrate how we can leverage the gesture keyboard to extend its functionality while preserving its accuracy, simplicity, and accessibility. We implemented several applications of each technique to illustrate the variety of ways that it can be incorporated into different context. Finally, we show how *CommandBoard* can be combined with traditional command selection techniques, including pull-down menus and tool bars, as well as adopting more innovative gesture-based menu techniques, such as Marking Menus and OctoPocus.

In future work, we plan a longitudinal study to see how easily users learn *CommandBoard* over time and how they balance the trade-offs it offers between recognition-based novice performance, and recall-based expert performance. We also seek new ways of helping users personalize their mobile devices, by letting them define their own gestures and customize their commands. Ultimately, we see *CommandBoard* as offering a path towards simpler, yet significantly more powerful personal devices.

ACKNOWLEDGMENTS

This work was supported by European Research Council (ERC) grant n° 321135 CREATIV: Creating Co-Adaptive Human-Computer Partnerships.

REFERENCES

1. Jessalyn Alvina, Joseph Malloch, and Wendy E. Mackay. 2016. Expressive Keyboards: Enriching Gesture-Typing on Mobile Devices. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology (UIST '16)*. ACM, New York, NY, USA, 583–593. DOI: http://dx.doi.org/10.1145/2984511.2984560
2. Caroline Appert and Shumin Zhai. 2009. Using Strokes As Command Shortcuts: Cognitive Benefits and Toolkit Support. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '09)*. ACM, New York, NY, USA, 2289–2298. DOI: http://dx.doi.org/10.1145/1518701.1519052
3. Olivier Bau and Wendy E. Mackay. 2008. OctoPocus: A Dynamic Guide for Learning Gesture-based Command

Sets. In *Proceedings of the 21st Annual ACM Symposium on User Interface Software and Technology (UIST '08)*. ACM, New York, NY, USA, 37–46. DOI: http://dx.doi.org/10.1145/1449715.1449724

4. Barry Brown, Moira McGregor, and Donald McMillan. 2014. 100 Days of iPhone Use: Understanding the Details of Mobile Device Use. In *Proceedings of the 16th International Conference on Human-computer Interaction with Mobile Devices & Services (MobileHCI '14)*. ACM, New York, NY, USA, 223–232. DOI:http://dx.doi.org/10.1145/2628363.2628377
5. Daniel Buschek, Alexander De Luca, and Florian Alt. 2015. There is More to Typing Than Speed: Expressive Mobile Touch Keyboards via Dynamic Font Personalisation. In *Proceedings of the 17th International Conference on Human-Computer Interaction with Mobile Devices and Services (MobileHCI '15)*. ACM, New York, NY, USA, 125–130. DOI: http://dx.doi.org/10.1145/2785830.2785844
6. Vittorio Fuccella, Poika Isokoski, and Benoit Martin. 2013. Gestures and Widgets: Performance in Text Editing on Multi-touch Capable Mobile Devices. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '13)*. ACM, New York, NY, USA, 2785–2794. DOI: http://dx.doi.org/10.1145/2470654.2481385
7. Emilien Ghomi, Stéphane Huot, Olivier Bau, Michel Beaudouin-Lafon, and Wendy E. Mackay. 2013. ArpèGe: Learning Multitouch Chord Gestures Vocabularies. In *Proceedings of the 2013 ACM International Conference on Interactive Tabletops and Surfaces (ITS '13)*. ACM, New York, NY, USA, 209–218. DOI: http://dx.doi.org/10.1145/2512349.2512795
8. François Guimbretiére and Terry Winograd. 2000. FlowMenu: Combining Command, Text, and Data Entry. In *Proceedings of the 13th Annual ACM Symposium on User Interface Software and Technology (UIST '00)*. ACM, New York, NY, USA, 213–216. DOI: http://dx.doi.org/10.1145/354401.354778
9. Sunjun Kim and Geehyuk Lee. 2016. TapBoard 2: Simple and Effective Touchpad-like Interaction on a Multi-Touch Surface Keyboard. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems (CHI '16)*. ACM, New York, NY, USA, 5163–5168. DOI: http://dx.doi.org/10.1145/2858036.2858452
10. Per-Ola Kristensson and Shumin Zhai. 2004. SHARK2: A Large Vocabulary Shorthand Writing System for Pen-based Computers. In *Proceedings of the 17th Annual ACM Symposium on User Interface Software and Technology (UIST '04)*. ACM, New York, NY, USA, 43–52. DOI:http://dx.doi.org/10.1145/1029632.1029640
11. Per Ola Kristensson and Shumin Zhai. 2007. Command Strokes with and Without Preview: Using Pen Gestures on Keyboard for Command Selection. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '07)*. ACM, New York, NY, USA, 1137–1146. DOI: http://dx.doi.org/10.1145/1240624.1240797
12. Gordon Kurtenbach and William Buxton. 1991. Issues in Combining Marking and Direct Manipulation Techniques. In *Proceedings of the 4th Annual ACM Symposium on User Interface Software and Technology (UIST '91)*. ACM, New York, NY, USA, 137–144. DOI: http://dx.doi.org/10.1145/120782.120797
13. Gordon Kurtenbach and William Buxton. 1993. The Limits of Expert Performance Using Hierarchic Marking Menus. In *Proceedings of the INTERACT '93 and CHI '93 Conference on Human Factors in Computing Systems (CHI '93)*. ACM, New York, NY, USA, 482–487. DOI: http://dx.doi.org/10.1145/169059.169426
14. David M. Lane, H. Albert Napier, S. Camille Peres, and Aniko Sandor. 2005. Hidden Costs of Graphical User Interfaces: Failure to Make the Transition from Menus and Icon Toolbars to Keyboard Shortcuts. *Int. J. Hum. Comput. Interaction* 18, 2 (2005), 133–144. http://dblp.uni-trier.de/db/journals/ijhci/ijhci18.html#LaneNPS05
15. Yang Li. 2010. Gesture Search: A Tool for Fast Mobile Data Access. In *Proceedings of the 23Nd Annual ACM Symposium on User Interface Software and Technology (UIST '10)*. ACM, New York, NY, USA, 87–96. DOI: http://dx.doi.org/10.1145/1866029.1866044
16. Wendy E. Mackay. 2002. Which Interaction Technique Works when?: Floating Palettes, Marking Menus and Toolglasses Support Different Task Strategies. In *Proceedings of the Working Conference on Advanced Visual Interfaces (AVI '02)*. ACM, New York, NY, USA, 203–208. DOI: http://dx.doi.org/10.1145/1556262.1556294
17. Joseph Malloch, Carla F. Griggio, Joanna McGrenere, and Wendy E. Mackay. 2017. Fieldward and Pathward: Dynamic Guides for Defining Your Own Gestures. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems (CHI '17)*. ACM, New York, NY, USA, 4266–4277. DOI: http://dx.doi.org/10.1145/3025453.3025764
18. Miguel A. Nacenta, Yemliha Kamber, Yizhou Qiang, and Per Ola Kristensson. 2013. Memorability of Pre-designed and User-defined Gesture Sets. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '13)*. ACM, New York, NY, USA, 1099–1108. DOI: http://dx.doi.org/10.1145/2470654.2466142
19. Petra Neumann, Annie Tat, Torre Zuk, and Sheelagh Carpendale. 2007. KeyStrokes: Personalizing Typed Text with Visualization. In *Proceedings of the 9th Joint Eurographics / IEEE VGTC Conference on Visualization (EUROVIS'07)*. Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 43–50. DOI: http://dx.doi.org/10.2312/VisSym/EuroVis07/043-050

20. Richard C. Omanson, Craig S. Miller, Elizabeth Young, and David Schwantes. 2010. Comparison of Mouse and Keyboard Efficiency. *Proceedings of the Human Factors and Ergonomics Society Annual Meeting* 54, 6 (2010), 600–604. DOI: http://dx.doi.org/10.1177/154193121005400612

21. Robert Pastel. 2006. Measuring the Difficulty of Steering Through Corners. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '06)*. ACM, New York, NY, USA, 1087–1096. DOI: http://dx.doi.org/10.1145/1124772.1124934

22. Dario D. Salvucci, Niels A. Taatgen, and Jelmer P. Borst. 2009. Toward a Unified Theory of the Multitasking Continuum: From Concurrent Performance to Task Switching, Interruption, and Resumption. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '09)*. ACM, New York, NY, USA, 1819–1828. DOI: http://dx.doi.org/10.1145/1518701.1518981

23. Christopher D Wickens, Justin G Hollands, Simon Banbury, and Raja Parasuraman. 2015. *Engineering psychology & human performance*. Psychology Press. 197–244 pages.

24. Shumin Zhai and Per Ola Kristensson. 2012. The Word-gesture Keyboard: Reimagining Keyboard Interaction. *Commun. ACM* 55, 9 (Sept. 2012), 91–101. DOI:http://dx.doi.org/10.1145/2330667.2330689