HotStrokes: Word-Gesture Shortcuts on a Trackpad

Wenzhe Cui
Department of Computer Science
Stony Brook University
Stony Brook, New York, USA
wecui@cs.stonybrook.edu

Jingjie Zheng
Google
Kitchener, Ontario, Canada
jingjie@acm.org

Blaine Lewis
Cheriton School of Computer Science
University of Waterloo
Waterloo, Ontario, Canada
blaine.lewis@uwaterloo.ca

Daniel Vogel
School of Computer Science
University of Waterloo
Waterloo, Ontario, Canada
dvogel@uwaterloo.ca

Xiaojun Bi
Department of Computer Science
Stony Brook University
Stony Brook, New York, USA
xiaojun@cs.stonybrook.edu

ABSTRACT

Expert interaction techniques like hotkeys are efficient, but poorly adopted because they are hard to learn. HotStrokes removes the need for learning arbitrary mappings of commands to hotkeys. A user enters a HotStroke by holding a modifier key, then gesture typing a command name on a laptop trackpad as if on an imaginary virtual keyboard. The gestures are recognized using an adaptation of the *SHARK*^{2} algorithm with a new spatial model and a refined method for dynamic suggestions. A controlled experiment shows HotStrokes effectively augments the existing "menu and hotkey" command activation paradigm. Results show the method is efficient by reducing command activation time by 43% compared to linear menus. The method is also easy to learn with a high adoption rate, replacing 91% of linear menu usage. Finally, combining linear menus, hotkeys, and HotStrokes leads to 24% faster command activation overall.

CCS CONCEPTS

• **Human-centered computing** → **Interaction devices**; **Interaction techniques**;

ACM Reference Format:
Wenzhe Cui, Jingjie Zheng, Blaine Lewis, Daniel Vogel, and Xiaojun Bi. 2019. HotStrokes: Word-Gesture Shortcuts on a Trackpad. In *CHI Conference on Human Factors in Computing Systems Proceedings (CHI 2019), May 4–9, 2019, Glasgow, Scotland UK*. ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3290605.3300395

1 INTRODUCTION

It is a common belief that user interfaces should be easy to learn, provide high-performance mechanisms for power users, and ideally, support the progression from ease to efficiency [9, 23, 43]. In personal computing, menus and hotkeys (or keyboard shortcuts) are the most common ways of activating commands, representing the two opposite ends

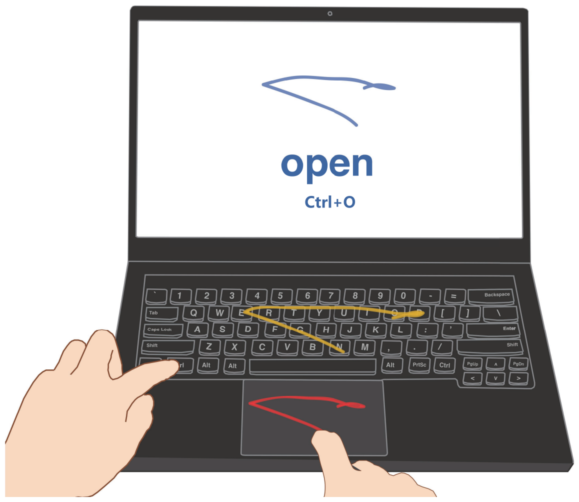

Figure 1: HotStrokes activates a command ("*Open*") by holding down the Ctrl key, then tracing through letters of that command on the trackpad ("*O-P-E-N*") as if gesturing on a soft keyboard. The red trace on the trackpad shows the finger motion, the yellow trace on the keyboard shows the equivalent gesture typing shape, and the blue trace on the display provides feedback. Red and yellow traces are for illustration only and invisible in real use.

of the ease-to-efficiency spectrum. On the one end, graphics-based menu interaction is easy for novices, but can be inefficient when performing repetitive actions [48]. On the other, hotkeys are fast but require a conscious and effortful switch from menu interaction [15, 21]. Research shows that hotkeys are largely underused, and many attributes to the arbitrary command-to-key mappings caused by limited keyboard input space [9, 15, 27, 48, 49].

We present HotStrokes, a command activation method that removes the need for memorizing arbitrary key assignment (Figure 1). It works by transforming a laptop trackpad into an imaginary word-gesture keyboard: a user enters a command by holding down a modifier key (e.g., Ctrl), then inputs the command label on the trackpad by gesture typing. The user learns a word gesture by referring to the physical keyboard on the laptop for static visual guidance. Then, with repetitive use of the same command, they may gradually transition from depending on the visual guidance to directly recalling the gesture from memory.

We implement HotStrokes using an adaption of the *SHARK*^{2} algorithm [19] with a new spatial model and a refined method for dynamic suggestions. Our evaluation shows that HotStrokes is efficient by reducing command activation time by 43% compared to linear menus. The method is also easy to learn with a high adoption rate, replacing 91% of linear menu usage. Finally, combining linear menus, hotkeys, and HotStrokes leads to 24% faster command activation overall.

2 RELATED WORK

HotStrokes relates to the word-gesture keyboard, techniques for promoting hotkey usage, and gestural command input.

Word-gesture Keyboard

Word-gesture keyboard [19, 34, 42, 45], also variably known as gesture typing or shape writing, is a text entry method that has been widely adopted on touchscreen mobile devices. Users may trace through letters of a word as a single gesture instead of tapping on individual letters. As users gain expertise, they may gradually transition to directly recalling the gesture without relying on the visual guidance provided by the keyboard. Past research has extended word-gesture keyboard to supporting bi-manual input [8], mid-air text entry [31], head-movement based [41], and device-tilt based text input [40]. Building on the success of word-gesture input, HotStrokes extends this input paradigm to command activation on a trackpad.

Promoting Hotkey Usage

Hotkeys are viewed as the gold standard in efficient command input [27, 34, 35]. They enable parallel interaction by combining keyboard and mouse input, and allow users to focus on primary tasks by reducing visual distraction. However, research shows that except for very frequent commands like “copy” and “paste”, users fail to transition from using graphical interfaces to using shortcuts [1, 18, 27, 39, 48].

The failure to adopt hotkeys may be ascribed to factors like poor visibility [14, 15, 39], cumbersome hand movement [22, 32], lack of user motivation [9, 26, 38], and the large cost to learn more efficient interaction strategies [21, 23]. Yet, one primary issue, as identified by many [15, 18, 22, 27, 39] and reiterated by a recent survey by Zheng et al. [48], is the requirement for memorizing an often arbitrary mapping from command labels to hotkeys.

Various techniques have been proposed to promote hotkey usage. ExposeHK [30] displays hotkey labels on UI elements when a modifier key is pressed to enable rehearsing hotkey actions with visual guidance. FingerArc and FingerChord [48] provide similar guidance, but reduce the need for pressing modifier keys by detecting special hand gestures. Grossman et al. [15] show providing auditory feedback about shortcut information when interacting with menus may significantly improve hotkey adoption. IconHK [14] increases the visibility of hotkeys by artistically incorporating hotkey letters into toolbar icons.

Despite these techniques, requiring an arbitrary mapping from a command to key presses is still a fundamental challenge for the wide adoption of hotkeys, partly due to the limited keyboard space compared to the relatively large number of commands on desktop computing [49]. HotStrokes proposes to alleviate this issue by leveraging the interaction space of the trackpad for command activation. Because HotStrokes directly associate the command with its word gesture, it removes the need for learning arbitrary mappings, which may substantially ease the learning procedure.

Gestural Command Input

Taking advantage of human’s ability in well-remembering pictorial information [33], many techniques have explored using gestures for command input [44]. Gestural commands may address the arbitrary mapping problem by using gestures with semantic meanings. For example, Gesture Search [28] triggers actions on a smartphone by drawing related characters, and similarly, Gesture Avatar [29] activates on-screen UI widgets by drawing their icons.

Many other interfaces use somewhat arbitrary gestures, but provide visual guidance for novices to facilitate learning. On touchscreen devices, Marking Menu [21, 24, 25] and its variants [5, 6, 11, 12, 46, 47] allow novices to “press and hold” to expand a hierarchical menu and trace through menu labels to form a gesture, while experts may approximate the same gesture without depending on any guidance. OctoPocus [7] expands this idea, but only shows what gestures are possible when users pause their fingers. On trackpads, Markpad [13] triggers commands by gesturing on a predefined grid while referring to the display for visual guidance.

More relevant are Command Strokes [20] and Command-Board [2], both supporting entering a command by gesture typing its label on a *touchscreen* keyboard. HotStrokes transfers this idea to a laptop trackpad. It significantly deviates from these techniques by letting users refer to the physical keyboard for static visual guidance and providing gesture feedback on a separate display. Anderson and Bischof [3] show that static guidance, in contrary to common belief, may actually foster better motor performance when recalling gestures, and the limited guidance with HotStrokes compared to showing an on-screen keyboard directly under the finger may have certain benefits.

3 HOTSTROKES: WORD-GESTURE SHORTCUTS

To strike a balance between efficiency and ease of use, we design and implement *HotStrokes* to enable entering a command by gesture typing its label on a laptop trackpad as if on an imaginary virtual keyboard, while also holding a modifier key (Figure 1). The gesture stroke is visualized on the laptop screen, providing real-time action feedback. The command is executed as soon as the finger lifts off the trackpad. The activated command label, as well as the hotkey option, is displayed on the screen after the execution. Since HotStrokes is aimed to augment, rather than replace the existing command input paradigm, displaying hotkeys serves as calm notification [38] to remind users of the existence of another efficient command input method.

HotStrokes has the following theoretical advantages. First, a known command can be triggered directly without having to search in linear menus or toolbars. Second, it inherits the same benefits of gesture typing, supporting a gradual transition from relying on visual guidance to direct memory recall. Third, it reduces the need for memorizing arbitrary key combinations by naturally mapping command labels to corresponding word gestures. The physical keyboard above the trackpad also serves as a reference to help users recall gesture shapes and locations, easing the learning procedure.

In the rest of the paper, we first investigate how to decode HotStrokes input, then study the pros and cons of using HotStrokes in combination with the existing "linear menu and hotkey" command input paradigm.

4 DECODING HOTSTROKES INPUT

Decoding HotStrokes input on a trackpad is substantially different from regular touchscreen keyboards. First, the input signal is expected to be noisy due to the lack of on-screen keyboard visual where locating key positions can be challenging to users. Second, no language context exists for command input, so the decoding algorithm can not leverage a language model to disambiguate words like regular typing.

In this section, we review the *SHARK*^{2} gesture decoding algorithm [19], analyze users gesture typing behavior on a trackpad, and optimize *SHARK*^{2} for decoding HotStrokes input.

Decoding Principle

We developed a gesture decoding algorithm from scratch, following principles outlined in *SHARK*^{2} [19] with two augmentations explained later. Since command input has no language context, our algorithm relied on the shape channel and location channel in *SHARK*^{2} for decoding.

Shape Channel. This channel classifies gestures based on the shape information. It takes a gesture stroke u as input, and outputs the shape matching distance $D_s(u, v)$ between the gesture input u and the template pattern v of a candidate command w. The template pattern is formed by connecting key centers of the command with straight lines on a Qwerty layout whose dimensions are identical with the trackpad. We followed the same procedure outlined in [19] to calculate $D_s(u, v)$.

Location Channel. This channel examines the absolute location of the user's gesture trace on the keyboard. It outputs average the location distance $D_l(u, v)$ between the input gesture u and the template pattern of a command candidate v. The procedure of calculating $D_l(u, v)$ is also detailed in *SHARK*^{2} [19]. We followed the same procedure. The only difference was that $D_l(u, v)$ was divided by the diagonal of a keyboard whose dimensions were identical to the trackpad, which normalized its value between $[0, 1]$, in the similar scale with $D_s(u, v)$.

Finally, we combined the shape distance $D_s(u, v)$ and location distance $D_l(u, v)$ to obtain the matching distance $D(u, v)$ between an input gesture u and a template pattern v as:

$$D(u, v) = \alpha D_s(u, v) + (1 - \alpha) D_l(u, v), \tag{1}$$

where α is the weight between D_s and D_l which was later trained from the collected data.

Applying Equation (1), the algorithm obtains the matching distance $D(u, v)$ between an input gesture u with every command candidate w whose template pattern is denoted by v. The one with the shortest $D(u, v)$ is the intended command while the candidates with 2nd and 3rd shortest $D(u, v)$ are the 2nd and 3rd suggestions.

To investigate whether this algorithm can decode HotStrokes and optimize its performance, we conducted a study to collect HotStrokes input data, and used it to train and test the algorithm parameters.

Collecting HotStrokes Input Data

Task. The study was to study the users' HotStrokes gesturing behavior and collect HotStrokes input data. The subjects

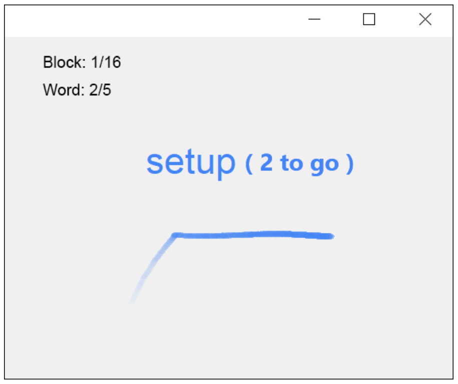
Block: 1/16
Word: 2/5
setup (2 to go)

Figure 2: Screenshot of the data collection application. The user was instructed to gesture type *setup* three times in a row.

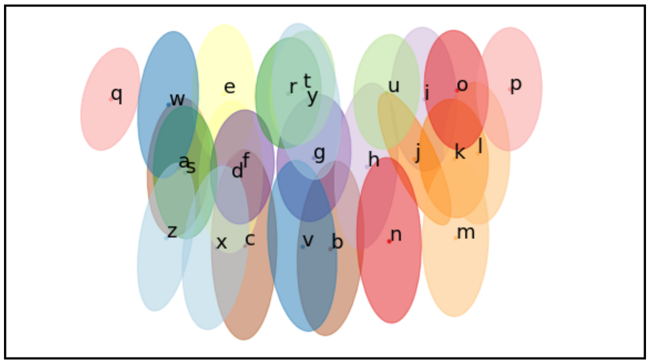

Figure 3: Imaginary key position distribution with 95%confidence ellipses. The black boundaries were the trackpad borders.

were instructed to gesture type commands displayed on the screen via the trackpad of a laptop, imagining that there was an invisible keyboard superimposed on the trackpad and could successfully decode their input. We used this Wizard of Oz keyboard approach to collect neutral input data which was not biased toward any algorithm. Participants were asked to gesture as naturally as possible. The keyboard above the trackpad worked as an input aid to help participants who were not familiar with QWERTY keyboard layout.

Design. The command set contained 80 single word commands chosen from menu items from Apple OS X. These words covered all 26 English letters from 'a' to 'z'. These commands were extracted from Safari menus and systems applications of Apple OS X. Most of the commands were single words. For those commands containing multiple words that were hard to input in one continuous gesture, we picked either the verb or the core word in the command phrase. An 80-word dictionary was set up using these commands for gesture input recognition.

Each command appeared 3 times consecutively, while the Ctrl key was pressed. In total, each participant input 240 gestures. The orders of all words were randomized across participants. The gesture traces were recorded for later training and testing. The gesture traces were displayed on the laptop screen as feedback.

Participants and Apparatus. We recruited 12 participants (3 females, 25%), aged from 22 to 39. All were right-handed. Three of them had gesture input experiences. The others had never used gesture typing before. The subjects were instructed to use their preferred finger and hand. In the experiment, all the users used right hand and index finger. All the experiments conducted on a Lenovo Thinkpad X1 Carbon 2017. (CPU:i7-7500U, RAM:8G) with Synaptics HID-Compliant Touch pad Device. In total, 2880 gestures of 80 words were collected from 12 participants.

Data Analysis – Imagined Key Position Distribution

After collecting the input data, we first investigated whether users could recall key positions without key visuals, and whether the imagined key positions differed from a regular Qwerty keyboard. Such information would be critical in designing the decoding algorithm.

We located the imagined key positions of a command w from its corresponding gesture input u. It was straightforward to locate the first and last letter positions: the first and last touch points in u corresponded to the first and last letter in w, respectively. However, it was not easy to match letters in the middle of w with touch points in the gesture u. We achieved it using the Dynamic Time Warping (DTW) algorithm [37]. We performed DTW between u and the pattern template of w (denoted by v). For a letter c in w, assuming its key center in the template pattern is v_c, the DTW algorithm would output its matching point u_c in u, which is the imagined key position of letter c on the trackpad. Note that DTW matched the template pattern (u) with a user-entered gesture (v) based on touch point coordinates of these two traces, not the time stamp. So matching results were unlikely to be affected by the gesturing time from letter to letter. More specifically, it first sampled u and v into 100 equidistant points, and obtained the optimal match between u and v. This procedure resulted in good matches for letters at the beginning and end of a word (they will be matched to starting and ending points on a gesture trace), and also middle letters where the gesture exhibited sharp turns.

Figure 3 shows the distribution of imagined key positions after this procedure. As shown, the relative imagined key centers approximately followed a Qwerty layout, which showed

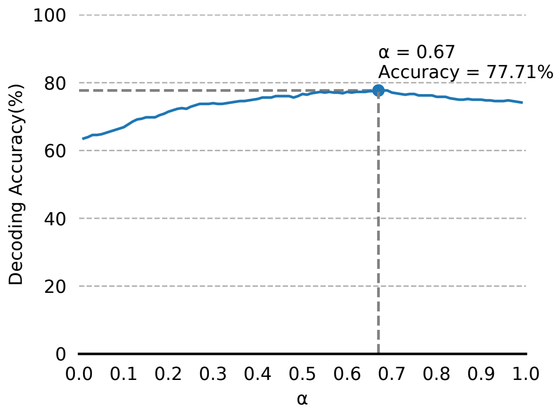

Figure 4: Performing a linear search to obtain the optimal α in Equation (1) on training data set.

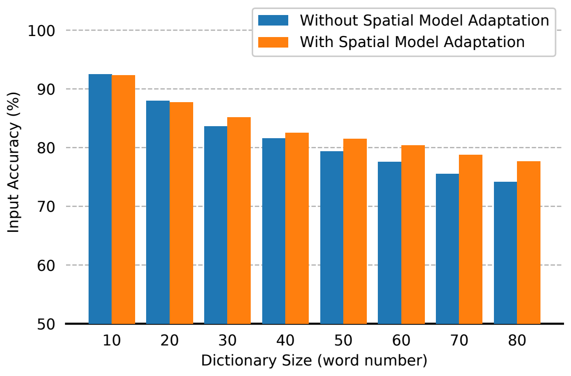

Figure 5: Decoding accuracy by the size of the command set.

that users could to a certain degree recall relative Qwerty key locations without key visuals shown on the trackpad. Previous research in imaginary interfaces [16, 17] has shown that users' memory on interface location is strong enough to help them draw single-stroke characters and simple sketches and perform interaction without interface visuals. Our work advances the understanding to word-gesture input on a trackpad, showing that such ability also holds for gesture typing on an imaginary keyboard on a trackpad.

Figure 3 also showed the touch point distribution clustered near the central region of the trackpad, suggesting that the width of the imaginary keyboard was smaller than the width of the trackpad, and the imagined key positions did not evenly distribute over the trackpad. This finding led us to hypothesize that adapting the key positions to the imagined key positions (a.k.a, spatial model adaption) may improve the accuracy of recognizing HotStrokes input. We investigate the effectiveness of this method as follows.

Enhancing Decoding 1. Spatial Model Adaption

In this section, we evaluate the effectiveness of *spatial model adaption*: we learned the key positions from the collected data, used them in decoding and compared its performance with a regular Qwerty layout.

To avoid over-fitting, we randomly selected 10 out of 12 users' data as the training set, and the rest as the test set. We first computed the center of each key from the training data, following the procedure described in the previous section. The new key locations were then used in both shape and location channels [19] to form a template pattern for a command, and used in both shape and location distance calculation.

We also performed a linear search to obtain the optimal α value in Equation (1), to maximize the decoding accuracy on training data. It worked as follows. We first initiated α to 0, and gradually increased it to 1, with 0.01 as the step length. For each α value, we plugged it into Equation (1) and obtained the overall decoding accuracy on the training data. The α that led to the greatest accuracy was the optimal value. Figure 4 showed how the accuracy varied as α changed. The optimal α was 0.67 and the accuracy on the training data was 77.71%.

The algorithm was then evaluated over the test data and compared with a baseline condition. The baseline condition was using a regular Qwerty layout in decoding: we equally divided the trackpad into a 3×10 grid, and superimposed a Qwerty layout onto it. We also performed an identical linear search to obtain the optimal α value for the baseline, which was 0.88. The accuracy on the training data was 74.17%.

To investigate how the decoding accuracy was affected by the size of the command set, we created 8 test command sets from the test data, with the size of 10, 20, 30, $\cdots$, 80. For command set with size X, we randomly chose X commands from test data set, used these X commands as the dictionary in the decoding algorithm, and fed the gestures corresponding to these X commands into the algorithm for decoding.

Figure 5 shows the decoding accuracy by the size of the command set, with and without spatial model adaption. As shown, adapting key centers in general improved the decoding accuracy, especially for command set greater than 30.

Enhancing Decoding 2. Suggesting Multiple Commands When Necessary

Figure 5 showed that the top 1 decoding accuracy was above 90% on small-sized command set, but the accuracy drop as

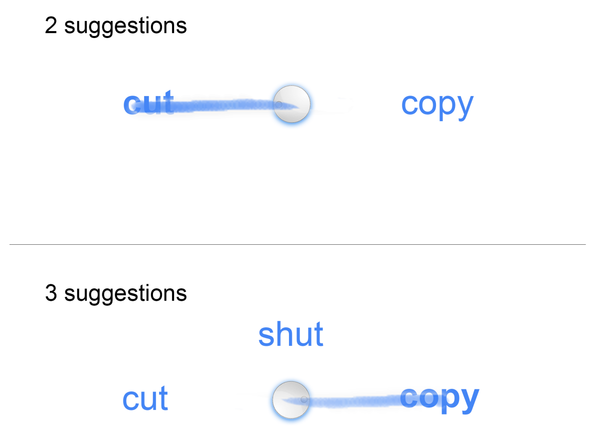

Figure 6: Dynamic suggestion shows 2 or 3 suggestions if the input command is ambiguous, determined by Equation (2), and Equation (6). Sliding the finger to the left, right, or up selects the corresponding command. Sliding down cancels the input.

the command set size increases, suggesting users will likely benefit from suggesting two or three commands especially on large command set. On the other hand, always providing multiple suggestions may introduce unnecessary cost for gestures where the top 1 suggestion was already the intended word, which was the majority across all the conditions (the top 1 accuracy was above 77% for across command set sizes).

To address this problem, we developed a *dynamic suggestion* feature. The decoding algorithm will suggest two command candidates if

$$D_{top2} - D_{top1} < \sigma_1, \tag{2}$$

where D_{top1} is the matching distance between the top 1 command candidate and the input gesture, and D_{top2} is the matching distance between the top 2 candidate and the input gesture.

The rationale behind this feature is that if both top 1 and top 2 candidates have close matching distance to the input gesture, it means both have high probabilities to be the intended command and the algorithm should present both to the user, and request the user to select the suggestion by performing a sliding gesture (Figure 6). If none of the suggestion is the intended command, sliding down will cancel the current input. If the algorithm produces only one suggestion, the command would automatically be executed upon the finger lifting off the trackpad.

To make the dynamic suggestion effective, we need to determine an optimal σ_1 value, which should produce 2 suggestions only if the intended command is the 2nd suggestion, and show one suggestion only if the intended command is the 1st suggestion.

We used recall and precision rates to quantify whether σ_1 is optimal. The recall rate is defined as

$$r = \frac{tp}{tp + fn}, \tag{3}$$

where tp is the number of true positive (i.e., the intended command was the 2nd suggestion and two suggestions were displayed), and fn is the number of false negative (i.e., the intended command was the 2nd suggestion but the algorithm only showed 1 suggestion).

The precision rate was defined as

$$p = \frac{tp}{tp + fp}, \tag{4}$$

where fp is the number of false positive (i.e., the intended command was the 1st suggestion but two suggestions were displayed) and tp is the number of true positive as previously explained.

We weighed recall rate more than precision because a false negative is more costly than a false positive. Missing a hit (false negative) means the user needs to redraw the gesture, while falsely reporting the 2nd suggestion (false positive) means the user needs to do a follow-up sliding to select the intended command.

To balance precision and recall rates, and weigh recall higher than precision, we performed a linear search to find the optimal σ_1 value that maximized the commonly used F_2 score [36] on the training data:

$$F_2 = 5 \cdot \frac{p \cdot r}{4 \cdot p + r}, \tag{5}$$

where p is the precision rate and r is the recall rate.

We first initiated σ_1 to 0, gradually increased it to 0.25 (max value in experiment data) with an increment of 0.0001, and calculated the F_2 score. $\sigma_1 = 0.0468$ led to the highest F_2 score of 0.6823.

Similarly, we expanded the dynamic suggestion feature to include the 3rd suggestion. If the matching distance of top 1 and top 3 command candidates satisfy the following equation, the decoding algorithm will show three candidates:

$$D_{top3} - D_{top1} < \sigma_2, \tag{6}$$

where E_{t_3} is the matching distance between the 3rd command candidate and the input gesture. The user will then slide the finger in the corresponding direction to select the candidate command (Figure 6). Similar to determining optimal σ_1, we searched for the optimal σ_2 to maximize the corresponding

Table 1: The intended command Coverage per command set size without and with dynamic suggestion, and false positive rates for 2 suggestions (2-FP) and 3 suggestions (3-FP) with dynamic suggestion.

size	Coverage (without)	Coverage (with)	2-FP	3-FP
10	92.31%	96.00%	13.05%	6.92%
20	87.76%	93.33%	18.03%	10.53%
30	85.20%	92.11%	22.33%	15.89%
40	82.56%	90.25%	25.51%	18.62%
50	81.52%	89.18%	29.52%	22.60%
60	80.42%	88.25%	32.95%	24.49%
70	78.81%	87.71%	35.94%	25.30%
80	77.71%	86.88%	38.07%	27.47%

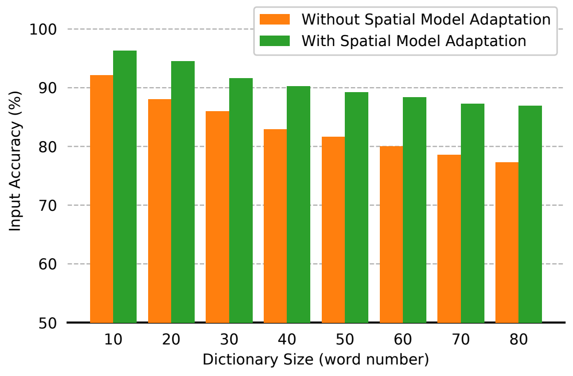

Figure 7: Intended command coverage without and with dynamic suggestions.

F_2 score, and found $\sigma_2 = 0.0318$ led to the highest F_2 score of 0.7155 on the training data set.

Equation (2) with $\sigma_1 = 0.0468$ and Equation (6) with $\sigma_2 = 0.0318$ were our solution for dynamically showing multiple commands when necessary. We applied it on the test data set to evaluate its effectiveness.

Figure 7 shows the intended command coverage after adopting the dynamic suggestion. Intended command coverage is the percentage of gestures in which the intended command was correctly recognized, or it is one of the presented suggestions when 2 or 3 suggestions are displayed. As shown in Figure 7 and Table 1, using dynamic suggestion brought the intended command coverage to above or near 90% for command set less or equal to 50. Table 1 showed the intended command coverage, and the false positive rates for 2, and 3 suggestions.

Discussion

Our investigation showed both spatial model adaption and dynamic suggestions were effective in recognizing HotStrokes input.

First, as shown in Figure 5, adapting the spatial model substantially improved the decoding accuracy, especially for command set greater than 30 commands. It was probably because users exhibited distinct gesturing behavior when drawing HotStrokes. For example, they underused the regions near the left and right boundaries. Accounting for their gesturing behavior improved the decoding performance.

Second, the dynamic suggestion method was effective in recommending the necessary suggestion when there was ambiguity in decoding. If the algorithm only showed the top 1 suggestion, the recognition accuracy dropped from 92.21% (10-command set) to around 80% as the command set size increases to 60 or 70. Dynamic suggestions brought the intended command coverage to above or near 90% especially for command sets with 50 or fewer commands. (Figure 7).

5 EVALUATION

After investigating how to support HotStrokes, we studied the utility of HotStrokes. We conducted a user study to investigate whether and to what degree providing HotStrokes as an alternative improved the performance of the existing command input paradigm (e.g., menus and hotkeys).

Experiment Setup

Design. The study was a command activation task, and had two conditions:

- Hotkeys + linear menus (KM). This represented the existing command activation paradigm. A user could freely choose either hotKey or use linear menus.
- HotStrokes + hotkeys + linear menus (SKM). HotStrokes were added as an option for triggering command. A user could choose any of these three methods for triggering a command, according to her preference.

We adopted a between-subject design to eliminate the potential carryover learning effect, since participants may learn hotkey combinations during the study. There was a single independent variable, technique, with two levels (KM, and SKM).

Task. The study consisted of multiple command activation trials. At the start of each trial, a start button appeared at the center of the screen. Participants moved the cursor to click on the start button, which would then be replaced by a stimulus icon at the same position with a size of 200 × 200 pixels. The icon was a highly familiar graphical object, such as cat, apple, or hat. The associations among all the stimulus objects were trivial and unique. Choosing iconic stimuli was

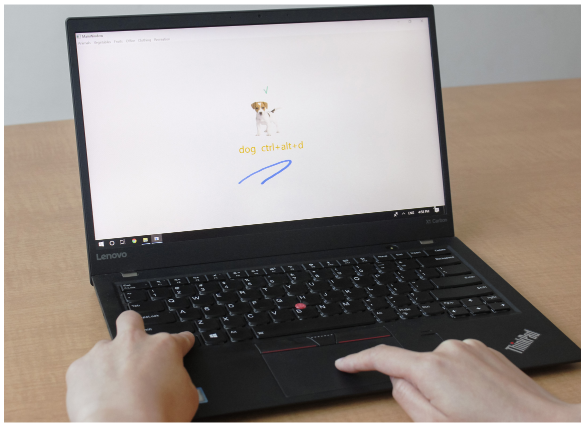

Figure 8: A participant just finished inputting the command *dog* with HotStrokes in Experiment. The green ✓indicated the input was correct. The blue trace was the feedback for HotStrokes input. The command label (e.g., *dog*) and corresponding hotkey option (e.g., *ctrl + alt + d*) were displayed after the command was successfully entered.

based on the fact that verbal or textural objects reduce the connections between the stimulus and the command. It is the same strategy used in previous research [4, 15].

Participants were then instructed to input the command according to the stimulus image on the icon. The participant could use either hotkeys or linear menus in the KM condition, and linear menus, hotkeys or HotStrokes in the SKM condition. In both conditions, participants were instructed to only use the trackpad for cursor movement, a common usage on laptop.

If the input was correct, the command label and the corresponding hotkey would appear under the stimulus image. It was consistent with the HotStrokes design: command feedback including hotkey option was provided after it was triggered. If the input was incorrect, the window would freeze for 3 seconds as a penalty, preventing participants from randomly guessing hotkeys. The user then repeated the same trial until the command was correctly triggered.

A post-study questionnaire was administered at the end of the study to collect participants' subject rating about mental demand, physical demand, temporal demand, the effort of performing the task, and their overall preference of each technique.

Command Set. The task included six categories of 72 commands, each category containing 12 commands. A 72-word dictionary was created based on these commands for recognition. For each command, a hotkey was assigned after the following rules designed to simulate the common hotkey assigning strategies:

1) If the first letter of a command word was the only letter appears in initial letters of all the commands, it would be associated with 'Ctrl' button. For example, 'g' was the first letter of 'gloves', and there was no other words in 30 commands start with 'g', 'Ctrl + G' would be assigned to glove as a hotkey.

2) If there were several words with the same first letter, we assigned the 7 combinations of 'Ctrl', 'Shift', 'Alt' buttons to each of them. In an order as 'Ctrl', 'Shift', 'Alt', 'Ctrl + Shift', 'Ctrl + Alt', 'Alt + Shift', 'Ctrl + Alt + Shift'. For example, if there are only three words 'pig', 'pea' and 'printer' in all 72 commands start with 'p', we will only use the first 3 of 7 combinations, 'Ctrl + P', 'Alt + P' and 'Shift + P'. To ensure all the hotkeys were designed with the first letter of the corresponding command, for all the words in our study, the number of words that start with the same letter is less than or equal to 7 in the study.

A command could also be activated via the menu bar. The height of the menu bar was 20 pixels (7mm on the laptop), approximately the same size of menu bars in common applications.

We randomly selected 14 commands from the 6 categories as target commands. The same target commands were used across participants. Since previous research showed that the distribution of commands could be approximated with a Zipfian distribution, the frequencies of the command were generated according to Zipfian distribution with exponent 1 (relative frequency = 1/rank) based on 30 random inputs of 7 items.

The generated frequencies are (12, 6, 4, 3, 2, 2, 1). Since we had 14 target commands, each frequency value was assigned to two commands. The frequencies of target commands in a 60-trial block were (12, 12, 6, 6, 4, 4, 3, 3, 2, 2, 2, 2, 1, 1). The command to frequency mapping was counterbalanced across all participants in each condition(KM and SKM). Therefore, each command was mapped to each frequency an equal number of times. This strategy was the same as used in the previous research [4, 15]. Each participant performed 20 trials (in 3 minutes) as a warm-up prior to the formal study. Each condition contained 2 blocks of 60 trials (120 trials in total) for each participant. The orders of commands were randomized in each block, with the constraint on the frequencies of each command. For instance, the two commands mapped to the frequency 6, would each appear exactly 12 times over both blocks.

Participants and Apparatus. 28 subjects (6 female), aged from 18 to 36 participated in this experiment. The median of self-reported familiarity with the Qwerty keyboard (1: extremely unfamiliar; 5: extremely familiar) was 5. The experiment was conducted on a Thinkpad X1 2017 running Windows 10 OS.

In total, our experiment included:

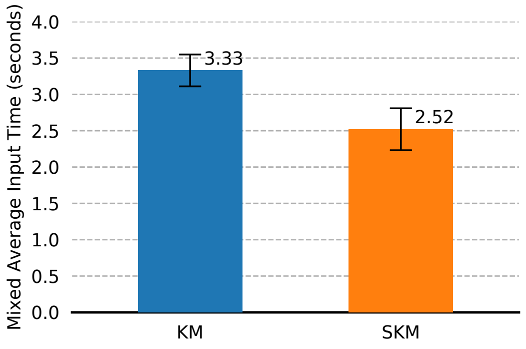

Figure 9: Average command input time (95% CI) in two conditions.

Table 2: The average input time(seconds) of each command activation method in two conditions.

method	KM - mean (std)	SKM - mean (std)
HotStrokes	-	2.93 (0.69)
linear menus	5.27 (1.02)	6.27 (0.85)
hotkeys	2.14 (0.37)	1.59 (0.52)

28 (subjects) × 2 (blocks) × 60 (trials) = 3360 trials.

Results

Command Input Time. This metric measured the efficiency of each technique. It was defined as the elapsed duration between the moment the start button was clicked and the moment a command was triggered.

Figure 9 shows the average command input time in KM and SKM conditions. The means (std) (in seconds) were 3.32($SD = 0.60$) for KM and 2.52($SD = 0.50$) for SKM. An independent t-test showed the difference was significant ($t_{26} = 3.87, p < 0.05$). Adding HotStrokes as an option shortened the average activation time by 24.1%.

Table 2 shows the break-down command activation time for each technique. To further compare the performance between linear menus and HotStrokes, we examined commands that were entered by linear menus in KM, and also entered by HotStrokes in SKM. HotStrokes reduced the average command input time of these commands by 43%, from 4.96 seconds (linear menus in KM) to 2.84 seconds (HotStrokes in SKM).

To investigate whether users became faster at repeating the same HotStrokes, we selected HotStrokes which had been repeated at least 4 times and analyzed their input time. Figure 10 showed the mean input time of those HotStrokes by the repetition number. As shown, the average input time

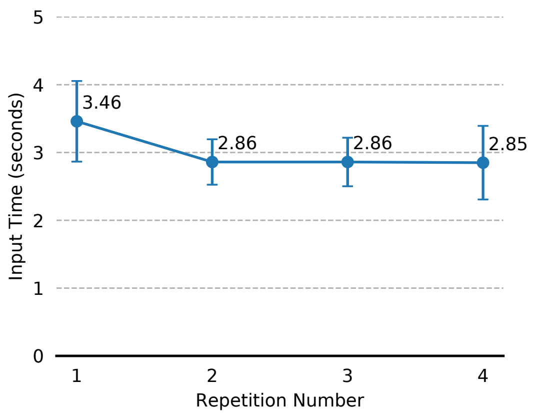

Figure 10: The mean (95% CI) input time of HotStrokes which had been repeated at least 4 times.

dropped by 17.3% after the first repetition and stayed stable afterwards, indicating that users could quickly improve the HotStrokes input efficiency by practicing it only once.

To further understand the strength and weakness of Hotstrokes, we made a comparison of left and right side hotkeys. The result showed that 64.3% of hotkeys used left-hand side letters (keys to the left of T, G, B inclusive). For commands that can be entered with left-hand side hotkeys, the mean input time (in HotStrokes+hotkeys+linear menus condition) was 1.83 seconds ($SD = 0.84$) for hotkeys and was 2.81 seconds ($SD = 0.72$) for HotStrokes (i.e. using a hotkey was 1 second faster).

Accuracy. The accuracy was the ratio of correctly triggered commands over the total number of commands. The mean accuracy was 91.85% ($SD = 5.00\%$) for KM, and 89.46% ($SD = 4.08\%$) for SKM. An independent t-test did not show significant difference the between KM and SKM ($t_{26} = 1.38, p = 0.17$). Within KM, the accuracy of was 87.98% ($SD = 7.40\%$) for hotkeys, and 97.82% ($SD = 2.33\%$) for linear menus, while within SKM, the accuracy was 86.35% ($SD = 5.68\%$) for HotStrokes, 86.06% ($SD = 13.26\%$) for hotkeys, and 90.24% ($SD = 14.84\%$) for linear menus. As a recall-based method, the accuracy of HotStrokes was slightly higher than its counterpart on the keyboard, and lower than the visual-guided linear menus. It was not unexpected: recall-based actions were efficient but harder to perform than visual-guided actions

Adoption Rate. Adoption rate shows how many percentages of commands were activated by a particular technique. As shown in Figure 11, in SKM where HotStrokes is enabled the majority of commands (64.7%) were triggered by HotStrokes;

Table 3: The repetition index of each input method in KM and SKM.

method	KM - mean (std)	SKM - mean (std)
HotStrokes	-	6.31 (5.17)
linear menus	4.46 (4.56)	2.16 (2.75)
hotkeys	9.84 (6.47)	10.95 (6.46)

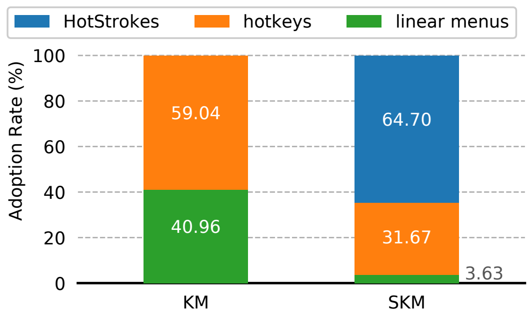

Figure 11: Adoption rate of each command activation method.

the usage of linear menus was only 3.63%, drastically lower than that in KM (40.96%).

To further understand the adoption pattern of each technique, we measured the *repetition index*, which was the number of times a command had been repeated when it was presented. Table 3 shows the mean (std) of this measure across conditions. The repetition index for the linear menus dropped from 4.46 (KM) to 2.16 (SKM), while this measure of hotkeys remained almost unchanged from KM to SKM. It showed that when HotStrokes was enabled, users substituted a large number of linear menu usages with HotStrokes, while hotkey usage was hardly affected by HotStrokes.

As repetition indices may be affected by frequencies of commands, to remove the potential impact caused by command frequencies, we examined repetition indices for commands with highest frequencies only (commands that were repeated 24 times in total across 2 blocks). For these commands, the mean (std) repetition indices for linear menus were: 8.24 (7.15) in KM, 3.77 (5.47) in SKM; repetition indices for hotkeys were 13.76 (6.34) in KM, 13.61 (6.42) in SKM; this metric was 11.67 (7.14) for HotStrokes in SKM. The data showed a similar pattern as data based on all commands. The repetition index for the linear menus dropped from 8.24 (KM) to 3.77 (SKM), while this measure for hotkeys remained almost unchanged from KM to SKM.

Subjective Rating. Subjects were asked to provide a continues numerical rating (1-least demanding, 10-most demanding) on mental demand, physical demand, temporal demand, and effort of completing the task in each condition. Figure 12 showed the mean subjective ratings. As shown, subjects' subjective ratings were unanimously in favor of SKM across all the questions.

Each subject was also asked to give an overall impression of the condition: "Overall how do you like activating commands in this condition?" (1-least preferred, 5-most preferred). The median rating was 5 for SKM and 4 for KM, showing that participants preferred to have HotStrokes as an extra command activation method.

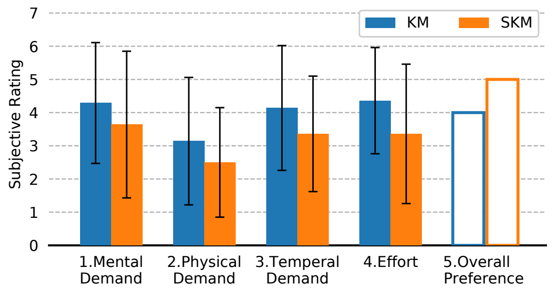

Figure 12: Mean(SD) of subjective ratings and median of overall preference. For measure 1-4, a lower rating means lower mental, physical, temporal demand, and smaller effort. For measure 5 (1 - least, 5-most preferred), a higher score means the method is more preferred. SKM received favorable ratings in all categories.

Discussion

Efficiency and adoption rate. The study showed promising results of adding HotStrokes as a command activation method.

First, HotStrokes is more efficient than the traditional linear menus. It shortened the average command input time by 43% over the linear menus, from 4.96 seconds to 2.84 seconds. This substantial time saving makes HotStrokes a great complement to the existing command input paradigm. Overall, providing HotStrokes as an option reduced the average command issuing time by 24%, compared with using linear menus and hotkeys only.

Second, HotStrokes is also very easy to adopt. Once it was enabled, users substituted 91% of linear menu usage with HotStokes (linear menu usage dropped from 40.96% in KM to 3.63% in SKM). HotStrokes adoption rate was also the highest among the three options in SKM, at 64.7%. Users could quickly improve the input efficiency after entering a HotStrokes only once. As shown in Figure 10, the mean input time dropped from 3.46 to 2.86 seconds (17%) after using a HotStrokes command once.

In contrast, the adoption rate of hotkeys was only 31.67% in SKM, much lower than that of HotStrokes. Note that the adoption rate of hotkeys was subject to its design strategy. In our experiment, we used two components to make hotkey commands easier to remember: using the first letters of commands and using increasingly complex modifiers. It was somewhat artificial, but was one of the common practices. In fact, the majority of hotkey commands (78.6%) executed by participants used a single modifier key, indicating that most of the hotkeys were easy to execute. Should other hotkey design strategy be adopted, it might affect the hotkey adoption rate.

Integration with existing command activation methods and interaction workflow. In addition to being efficient and easy-to-adopt, HotStrokes is also highly compatible with the existing hotkey trackpad commands and current unimanual trackpad usage. There are currently two types of existing hotkey trackpad commands: combining a modifier key with a click (e.g., Ctrl+click to open a link in a new tab), and multi-finger gestures (e.g., 3-finger swipe to switch workspace on a Mac). Since HotStrokes is executed by a combination of a modifier key with singer-finger touch gesture, it does not conflict with these common gestures.

Although in our experiment HotStrokes was executed bimanually, we believe unimanual usage of Hotstrokes is possible, depending on trackpad size and keyboard layout. Simple HotStrokes could be performed with one hand (e.g., pinky finger presses Ctrl while the thumb draws a short stroke on the trackpad.). Note the distance from the Cmd key to the farthest corner of a 15" MacBook Pro trackpad is about 19cm, comparable to the 95th percentile maximum hand span for a woman (also 19cm) [10].

We envision that HotStrokes could facilitate the existing interaction workflow via at least two approaches: (1) HotStrokes serves as a global command activation method, an alternative to linear menus and hotkeys; or (2) HotStrokes could be integrated with current one-handed hotkeys like pressing and holding down Ctrl-V in Excel, then drawing a word-gesture with the other hand to select a paste variation, such as 'text', 'transposed', or 'values'. The left-hand hotkey press serves as a single modifier press in the standard HotStrokes method. We focus on investigating approach #1 in this paper.

6 CONCLUSION AND FUTURE WORK

We present HotSrokes, word-gesture shortcuts on a trackpad. Our investigation showed that HotStrokes significantly improved the command activation performance, augmenting the existing command activation paradigm. Providing it as a command input alternative shortened the input time by 24%. It is more efficient than using the typical linear menus, reducing command activation time by 43%. It is also easy for adoption. Once enabled, users substituted over 90% of commands previously triggered with linear menus with HotStrokes; its overall adoption rate is 64.7%, higher than hotkeys or linear menus.

Decoding HotStrokes is challenging due to the lack of language context and the absence of key visuals on the trackpad. We have presented spatial model adaption and dynamic suggestion to address it. Both were effective and made HotStrokes a practical command activation method. Spatial model adaption improved the decoding accuracy by accounting for gesture behaviors on an invisible keyboard, while dynamic suggestion shows multiple command candidates when the input is ambiguous.

We have open-sourced HotStrokes and the installation file is available for download[1]. It is interesting to carry out longitudinal studies to investigate HotStrokes in a large interaction workflow with more complex tasks. Interesting topics include under which circumstances a user will replace typical menus or hotkeys with HotStrokes, and how quickly a user will adopt HotStrokes in real-world tasks. The open-sourced code and file provide tools for conducting such studies. We believe such understanding will complement the findings and insights gained from lab experiments reported in this paper.

ACKNOWLEDGEMENT

We thank the anonymous reviewers for their insightful comments. We thank our user study participants. This work was supported in part by NSF CHS-1815514. This work was done as part of the Ph.D. dissertation of Wenzhe Cui, a Stony Brook Ph.D. student.

REFERENCES

[1] Jason Alexander and Andy Cockburn. 2008. An Empirical Characterisation of Electronic Document Navigation. In *Proceedings of Graphics Interface 2008 (GI '08)*. Canadian Information Processing Society, Toronto, Ont., Canada, Canada, 123–130. http://dl.acm.org/citation.cfm?id=1375714.1375736

[2] Jessalyn Alvina, Carla F. Griggio, Xiaojun Bi, and Wendy E. Mackay. 2017. CommandBoard: Creating a General-Purpose Command Gesture Input Space for Soft Keyboard. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology (UIST '17)*. ACM, New York, NY, USA, 17–28. https://doi.org/10.1145/3126594.3126639

[3] Fraser Anderson and Walter F. Bischof. 2013. Learning and Performance with Gesture Guides. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '13)*. ACM, New York, NY, USA, 1109–1118. https://doi.org/10.1145/2470654.2466143

[4] 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*

[1]https://github.com/cuiwenzhe/HotStrokes

'09). ACM, New York, NY, USA, 2289–2298. https://doi.org/10.1145/1518701.1519052

[5] Gilles Bailly, Eric Lecolinet, and Laurence Nigay. 2007. Wave Menus: Improving the Novice Mode of Hierarchical Marking Menus. In *Human-Computer Interaction âĂŞ INTERACT 2007*, CÃĺcilia Baranauskas, Philippe Palanque, Julio Abascal, and Simone Diniz Junqueira Barbosa (Eds.). Number 4662 in Lecture Notes in Computer Science. Springer Berlin Heidelberg, 475–488. https://doi.org/10.1007/978-3-540-74796-3_45

[6] Gilles Bailly, Eric Lecolinet, and Laurence Nigay. 2008. Flower Menus: A New Type of Marking Menu with Large Menu Breadth, Within Groups and Efficient Expert Mode Memorization. In *Proceedings of the Working Conference on Advanced Visual Interfaces (AVI '08).* ACM, New York, NY, USA, 15–22. https://doi.org/10.1145/1385569.1385575

[7] 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. https://doi.org/10.1145/1449715.1449724

[8] Xiaojun Bi, Ciprian Chelba, Tom Ouyang, Kurt Partridge, and Shumin Zhai. 2012. Bimanual Gesture Keyboard. In *Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology (UIST '12).* ACM, New York, NY, USA, 137–146. https://doi.org/10.1145/2380116.2380136

[9] Andy Cockburn, Carl Gutwin, Joey Scarr, and Sylvain Malacria. 2014. Supporting Novice to Expert Transitions in User Interfaces. *ACM Comput. Surv.* 47, 2, Article 31 (Nov. 2014), 36 pages. https://doi.org/10.1145/2659796

[10] Sarah M. Donelson and Claire C. Gordon. 1996. 1995 Matched Anthropometric Database of US Marine Corps Personnel: Summary Statistics.

[11] JÃľrÃľmie Francone, Gilles Bailly, Eric Lecolinet, Nadine Mandran, and Laurence Nigay. 2010. Wavelet Menus on Handheld Devices: Stacking Metaphor for Novice Mode and Eyes-free Selection for Expert Mode. In *Proceedings of the International Conference on Advanced Visual Interfaces (AVI '10).* ACM, New York, NY, USA, 173–180. https://doi.org/10.1145/1842993.1843025

[12] Jeremie Francone, Gilles Bailly, Laurence Nigay, and Eric Lecolinet. 2009. Wavelet Menus: A Stacking Metaphor for Adapting Marking Menus to Mobile Devices. In *Proceedings of the 11th International Conference on Human-Computer Interaction with Mobile Devices and Services (MobileHCI '09).* ACM, New York, NY, USA, 49:1–49:4. https://doi.org/10.1145/1613858.1613919

[13] Bruno Fruchard, Eric Lecolinet, and Olivier Chapuis. 2017. MarkPad: Augmenting Touchpads for Command Selection. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems (CHI '17).* ACM, New York, NY, USA, 5630–5642. https://doi.org/10.1145/3025453.3025486

[14] Emmanouil Giannisakis, Gilles Bailly, Sylvain Malacria, and Fanny Chevalier. 2017. IconHK: Using Toolbar Button Icons to Communicate Keyboard Shortcuts. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems (CHI '17).* ACM, New York, NY, USA, 4715–4726. https://doi.org/10.1145/3025453.3025595

[15] Tovi Grossman, Pierre Dragicevic, and Ravin Balakrishnan. 2007. Strategies for Accelerating On-line Learning of Hotkeys. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '07).* ACM, New York, NY, USA, 1591–1600. https://doi.org/10.1145/1240624.1240865

[16] Sean Gustafson, Daniel Bierwirth, and Patrick Baudisch. 2010. Imaginary Interfaces: Spatial Interaction with Empty Hands and Without Visual Feedback. In *Proceedings of the 23Nd Annual ACM Symposium on User Interface Software and Technology (UIST '10).* ACM, New York, NY, USA, 3–12. https://doi.org/10.1145/1866029.1866033

[17] Sean Gustafson, Christian Holz, and Patrick Baudisch. 2011. Imaginary Phone: Learning Imaginary Interfaces by Transferring Spatial Memory from a Familiar Device. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology (UIST '11).* ACM, New York, NY, USA, 283–292. https://doi.org/10.1145/2047196.2047233

[18] Jeff Hendy, Kellogg S. Booth, and Joanna McGrenere. 2010. Graphically Enhanced Keyboard Accelerators for GUIs. In *Proceedings of Graphics Interface 2010 (GI '10).* Canadian Information Processing Society, Toronto, Ont., Canada, Canada, 3–10. http://dl.acm.org/citation.cfm?id=1839214.1839217

[19] 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. https://doi.org/10.1145/1029632.1029640

[20] 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. https://doi.org/10.1145/1240624.1240797

[21] Gordon Kurtenbach and William Buxton. 1994. User Learning and Performance with Marking Menus. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '94).* ACM, New York, NY, USA, 258–264. https://doi.org/10.1145/191666.191759

[22] Gordon Kurtenbach, George W. Fitzmaurice, Russell N. Owen, and Thomas Baudel. 1999. The Hotbox: Efficient Access to a Large Number of Menu-items. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '99).* ACM, New York, NY, USA, 231–237. https://doi.org/10.1145/302979.303047

[23] G. Kurtenbach, T. P. Moran, and W. Buxton. 1994. Contextual animation of gestural commands. In *Computer Graphics Forum.* 83–90.

[24] Gordon Paul Kurtenbach. 1993. *The Design and Evaluation of Marking Menus.* Technical Report.

[25] Gordon P. Kurtenbach, Abigail J. Sellen, and William A. S. Buxton. 1993. An Empirical Evaluation of Some Articulatory and Cognitive Aspects of Marking Menus. *Hum.-Comput. Interact.* 8, 1 (March 1993), 1–23. https://doi.org/10.1207/s15327051hci0801_1

[26] Benjamin Lafreniere, Carl Gutwin, and Andy Cockburn. 2017. Investigating the Post-Training Persistence of Expert Interaction Techniques. *ACM Trans. Comput.-Hum. Interact.* 24, 4 (Aug. 2017), 29:1–29:46. https://doi.org/10.1145/3119928

[27] 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. *International Journal of Human Computer Interaction* 18, 2 (2005), 133–144. https://doi.org/10.1207/s15327590ijhc1802_1

[28] Yang Li. 2010. Gesture Search: A Tool for Fast Mobile Data Access. In *UIST'10: Symposium on User Interface Software and Technology.* 87–96.

[29] Hao LÃij and Yang Li. 2011. Gesture Avatar: A Technique for Operating Mobile User Interfaces Using Gestures. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '11).* ACM, New York, NY, USA, 207–216. https://doi.org/10.1145/1978942.1978972

[30] Sylvain Malacria, Gilles Bailly, Joel Harrison, Andy Cockburn, and Carl Gutwin. 2013. Promoting Hotkey Use Through Rehearsal with ExposeHK. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '13).* ACM, New York, NY, USA, 573–582. https://doi.org/10.1145/2470654.2470735

[31] Anders Markussen, Mikkel Rønne Jakobsen, and Kasper Hornbæk. 2014. Vulture: A Mid-air Word-gesture Keyboard. In *Proceedings of the 32Nd Annual ACM Conference on Human Factors in Computing Systems (CHI '14).* ACM, New York, NY, USA, 1073–1082. https://doi.org/10.1145/2556288.2556964

[32] Hugh Mcloone, Ken Hinckley, and Edward Cutrell. 2003. Bimanual interaction on the Microsoft Office Keyboard. In *In Proceedings of the Conference on HumanComputer Interaction (INTERACT*. 49–56.

[33] Douglas L Nelson, Valerie S Reed, and John R Walling. 1976. Pictorial superiority effect. *Journal of Experimental Psychology: Human Learning and Memory* 2, 5 (1976), 523.

[34] Jakob Nielsen. 1992. Finding Usability Problems Through Heuristic Evaluation. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '92)*. ACM, New York, NY, USA, 373–380. https://doi.org/10.1145/142750.142834

[35] Daniel L. Odell, Richard C. Davis, Andrew Smith, and Paul K. Wright. 2004. Toolglasses, Marking Menus, and Hotkeys: A Comparison of One and Two-handed Command Selection Techniques. In *Proceedings of Graphics Interface 2004 (GI '04)*. Canadian Human-Computer Communications Society, School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, 17–24. http://dl.acm.org/citation.cfm?id=1006058.1006061

[36] C. J. Van Rijsbergen. 1979. *Information Retrieval* (2nd ed.). Butterworth-Heinemann, Newton, MA, USA.

[37] H. Sakoe and S. Chiba. 1978. Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing* 26, 1 (February 1978), 43–49. https://doi.org/10.1109/TASSP.1978.1163055

[38] Joey Scarr, Andy Cockburn, Carl Gutwin, and Philip Quinn. 2011. Dips and Ceilings: Understanding and Supporting Transitions to Expertise in User Interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '11)*. ACM, New York, NY, USA, 2741–2750. https://doi.org/10.1145/1978942.1979348

[39] Susanne Tak, Piet Westendorp, and Iris van Rooij. 2013. Satisficing and the Use of Keyboard Shortcuts: Being Good Enough Is Enough? *Interacting with Computers* 25, 5 (2013), 404–416. https://doi.org/10.1093/iwc/iwt016

[40] Hui-Shyong Yeo, Xiao-Shen Phang, Steven J. Castellucci, Per Ola Kristensson, and Aaron Quigley. 2017. Investigating Tilt-based Gesture Keyboard Entry for Single-Handed Text Entry on Large Devices. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems (CHI '17)*. ACM, New York, NY, USA, 4194–4202. https://doi.org/10.1145/3025453.3025520

[41] Chun Yu, Yizheng Gu, Zhican Yang, Xin Yi, Hengliang Luo, and Yuanchun Shi. 2017. Tap, Dwell or Gesture?: Exploring Head-Based Text Entry Techniques for HMDs. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems (CHI '17)*. ACM, New York, NY, USA, 4479–4488. https://doi.org/10.1145/3025453.3025964

[42] Shumin Zhai and Per-Ola Kristensson. 2003. Shorthand Writing on Stylus Keyboard. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '03)*. ACM, New York, NY, USA, 97–104. https://doi.org/10.1145/642611.642630

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

[44] Shumin Zhai, Per Ola Kristensson, Caroline Appert, Tue Haste Andersen, and Xiang Cao. 2012. Foundational issues in touch-screen stroke gesture design-an integrative review. *Foundations and Trends in Human-Computer Interaction* 5, 2 (2012), 97–205. https://hal.inria.fr/hal-00765046/

[45] Shumin Zhai, Per Ola Kristensson, Pengjun Gong, Michael Greiner, Shilei Allen Peng, Liang Mico Liu, and Anthony Dunnigan. 2009. Shapewriter on the Iphone: From the Laboratory to the Real World. In *CHI '09 Extended Abstracts on Human Factors in Computing Systems (CHI EA '09)*. ACM, New York, NY, USA, 2667–2670. https://doi.org/10.1145/1520340.1520380

[46] Shengdong Zhao and Ravin Balakrishnan. 2004. Simple vs. Compound Mark Hierarchical Marking Menus. In *Proceedings of the 17th Annual ACM Symposium on User Interface Software and Technology (UIST '04)*. ACM, New York, NY, USA, 33–42. https://doi.org/10.1145/1029632.1029639

[47] Jingjie Zheng, Xiaojun Bi, Kun Li, Yang Li, and Shumin Zhai. 2018. M3 Gesture Menu: Design and Experimental Analyses of Marking Menus for Touchscreen Mobile Interaction. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems (CHI '18)*. ACM, New York, NY, USA, Article 249, 14 pages. https://doi.org/10.1145/3173574.3173823

[48] Jingjie Zheng, Blaine Lewis, Jeff Avery, and Daniel Vogel. 2018. FingerArc and FingerChord: Supporting Novice to Expert Transitions with Guided Finger-Aware Shortcuts. In *The 31st Annual ACM Symposium on User Interface Software and Technology*. New York, NY, 17p.

[49] Jingjie Zheng and Daniel Vogel. 2016. Finger-Aware Shortcuts. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems (CHI '16)*. ACM, New York, NY, USA, 4274–4285. https://doi.org/10.1145/2858036.2858355