Int. J. Human-Computer Studies 96 (2016) 38–53

Contents lists available at ScienceDirect

Int. J. Human-Computer Studies

journal homepage: www.elsevier.com/locate/ijhcs

# Short links and tiny keyboards: A systematic exploration of design trade-offs in link shortening services

Sandy J.J. Gould [a,*], Anna L. Cox [a], Duncan P. Brumby [a], Sarah Wiseman [b]

[a] UCL Interaction Centre, University College London, 66-72 Gower Street, WC1E 6BT, United Kingdom
[b] Department of Computing, Goldsmiths, University of London, London SE14 6NW, United Kingdom

ARTICLE INFO

*Article history:*
Received 7 March 2016
Received in revised form
23 July 2016
Accepted 25 July 2016
Available online 28 July 2016

*Keywords:*
Link shortening
Short links
Keyboard
Optimization
Typing
Data entry
Mobile
Touchscreen

ABSTRACT

Link-shortening services save space and make the manual entry of URLs less onerous. Short links are often included on printed materials so that people using mobile devices can quickly enter URLs. Although mobile transcription is a common use-case, link-shortening services generate output that is poorly suited to entry on mobile devices: links often contain numbers and capital letters that require time consuming mode switches on touch screen keyboards. With the aid of computational modeling, we identified problems with the output of a link-shortening service, *bit.ly*. Based on the results of this modeling, we hypothesized that longer links that are optimized for input on mobile keyboards would improve link entry speeds compared to shorter links that required keyboard mode switches. We conducted a human performance study that confirmed this hypothesis. Finally, we applied our method to a selection of different non-word mobile data-entry tasks. This work illustrates the need for service design to fit the constraints of the devices people use to consume services.



## 1. Introduction

Semantically meaningful URLs are often long. This makes them tedious to transcribe. QR codes obviate the need to type, but are not human-readable and have a number of issues with usability (Shin et al., 2012) and security (Vidas et al., 2013). Link shortening services like *bit.ly* provide a compromise: the process of typing complex and lengthy URLs is accelerated and human readability is preserved. Unfortunately, many shortening services exhibit little consideration for how links might be made quick and easy to type. The outputs of shortening services usually contain a mix of numbers and mixed-case letters. On space-constrained mobile devices, entering these characters requires changing the keyboard from lowercase mode to number mode or uppercase mode. Making these mode switches to access different characters is particularly time consuming and error prone (Greene et al., 2014). Shorter links, therefore, may not necessarily be faster to type.

Generating short links that do not require keyboard mode switches necessarily means using a smaller selection of characters. For a given link length this means fewer unique links can be generated. This reduction in the number of options for each character of a link can be mitigated by increasing the length of links. But how much longer would such a link need to be? In this paper we model the process of text entry on three widely-used mobile platforms. We use simulations to systematically explore the output of a popular shortening service, *bit.ly*. To see whether the predictions we make based on our model hold true in reality we test them in a game-like human performance study.

We found that link shorteners trade-off link length and entry difficulty. Most shorteners optimize too aggressively for link length; their output is awkward to transcribe. We show that links can be made easier to type with only a modest increase in their length. Given the increasingly limited functional utility of ultra-short links on services like Twitter, link shortening services should prioritize making links easier to type.

### 1.1. Related work

Efforts to make text entry easier for people have mostly focused on improving entry interfaces, like keyboards (e.g., Cheng et al., 2013; Leiva et al., 2015; Oulasvirta et al., 2013). Another way to improve text entry is to adapt input interfaces to accommodate the kinds of input that they are most likely to receive. Wiseman et al. (2013) showed that in hospitals the distribution of digits entered into devices is not random. Input interfaces that are designed for a particular set of possible inputs perform better than standard interfaces that do not take account of the strings that are likely to be entered (Wiseman et al., 2013). The preponderance of third-party

* Corresponding author.
*E-mail address:* s.gould@cs.ucl.ac.uk (S.J.J. Gould).

http://dx.doi.org/10.1016/j.ijhcs.2016.07.009

mobile keyboards designed specifically for entering emoji characters or calendar events also reflects this kind of thinking: make the interface fit the input.

Unfortunately, implementing bespoke data-entry interfaces is a luxury that is rarely available to designers. People do not have custom keyboards installed for every input use case. More often than not, the design of input interfaces is entirely out of the control of service designers. In most circumstances input values should instead fit interfaces (see, e.g., Wiseman et al., 2016).

The entry of certain types of input can be suboptimal when keyboard designs have had to make trade-offs. This is especially the case for keyboards on phone-sized touchscreen devices. To save space, one of the concessions the designers of touchscreen keyboards make is to only show lowercase characters. Accessing uppercase letters and numbers requires multiple taps to change a keyboard's mode.

Gallagher and Byrne (2015) modeled the effect of having to switch modes on touchscreen keyboards in the context of password entry. They found that the interval between typing two lowercase letters was around 500 ms. The interval between typing a lowercase letter and an uppercase letter was approximately 1500 ms, a three-fold increase. Mode switching on small touchscreen keyboards is costly, but implementing a custom keyboard solely for entering passwords is impractical: touchscreens don't have the space to display the full set of numbers, letters and special characters in a single keyboard pane.

If input interfaces are fixed constraints in a system, we should consider how a service might be adjusted so as to better fit those constraints in likely contexts of use. This is not always possible, but there are scenarios in which target information can be substituted or altered without compromising a service. We focus on a particular example that exhibits this property: link shortening services.

### 1.2. Link shortening services

Link shortening services like *bit.ly*, *ow.ly* or *goo.gl* act as intermediaries between users and websites. Users provide a target link, for instance, https://www.elsevier.com/journals/international-journal-of-human-computer-studies/1071-5819/guide-for-authors. A shortening service generates a much shorter link, in this case, http://bit.ly/1OT4BPc. A mapping between these long and short links is stored by the shortening service. When a short URL is requested users are redirected to the original long URL.

Shorter links come at a cost: semantic information is lost from a URL. In the example above, a user reading the long link has a good idea that they'll end up on an Elsevier page. The short link, however, might just lead to Rick Astley's *Never Gonna Give You Up* (http://bit.ly/e4Rt5rr). A user would not know until they had followed it. The loss of semantic information has made shortening services a vector for phishing attacks (Chhabra et al., 2011; Klien and Strohmaier, 2012). Despite these shortcomings, shortening services have a number of benefits.

Short links are useful when space is limited or when characters are restricted. Shortening services also offer social media users methods for tracking engagement. Short links are often easier to copy and paste because they are more compact and generally do not cover multiple lines. Short links can also ease transcription from physical artefacts to digital devices. Short links feature on print advertising (see Fig. 1) and on slides during talks. By using shortened links, labyrinthine directions to a slide deck on a university server can be shortened to a few quick keystrokes.

### 1.3. Improving link shorteners

How well do existing link shortening services meet the requirements of the use cases we have discussed so far? RFC 3986 (Berners-Lee et al., 2005), which defines how URLs work, specifically discusses the tension between the digital and physical use of URLs: "[URL] design considerations," it says, "are not always in alignment". Given that trade-offs are required, do shortening services make reasonable ones?

One of the constraints on link shortening schemas is the need to be able to address a large potential set of links so that each short link can be guaranteed to be unique. Shorteners like *bit.ly* produce seven-character identifiers comprising numbers and mixed-case characters. In this scheme, each character can be any of 62 options: 26 uppercase letters, 26 lowercase letters or 10 digits. This yields a large space of possible identifiers, $(26+26+10)^7$, or around 3.5 trillion.

In the digital domain, unique identifiers could be made much shorter by allowing more than 62 options for each character. Many modern browsers (but not all services) support *percent encoded* URLs. UTF-8, a method of encoding Unicode characters, supports up to 1,112,064 characters. Using the full array of UTF-8 characters, a link shortening schema could exceed the size of the *bit.ly* pool of possible identifiers by orders of magnitude using only three characters (i.e., $1{,}112{,}064^3 \gg 62^7$). This yields links that are physically smaller and use fewer characters. Additionally, a link like *bit.ly/→∴♣* is no more or less meaningful than a link like *bit.ly/E5tF68G*. In the digital domain, where links are clicked and their composition after shortening is immaterial, shorteners could increase their effective compression ratio by using expanded

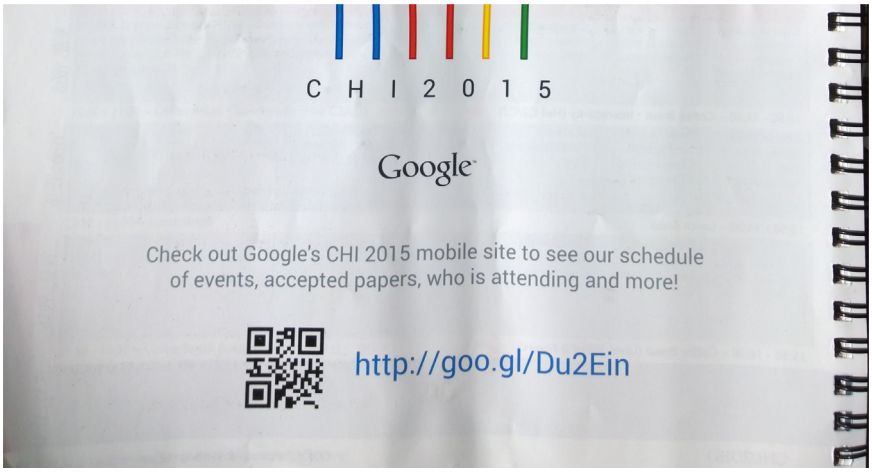


**Fig. 1.** Short links often appear on print advertising. Here the link contains numbers, capitals and lowercase characters.

character sets. Despite being short in length, expanded character sets become a problem if links need to be manually transcribed. Special characters can be difficult to identify in target links and hard to locate and enter on mobile devices.

If entering special characters is difficult, what about the capital letters and numbers used by services like *bit.ly*? On mobile keyboards these require mode switches. Is entering *bit.ly/lqexieb* easier than entering *bit.ly/S9T7R4B*? Shorteners use capital letters and numbers to maintain the size of their unique identifier pool. With an identifier of seven characters, using lowercase letters yields a set of possible links three orders of magnitude smaller ($26^7 < 62^7$, ~8 billion vs ~3.5 trillion). For the ambitious, running out of links might be a concern.

How long would a lowercase-only link need to be in order to offer a sufficiently large set of available links? Due to the nature of exponents, the answer is that only two more characters are required. Nine lowercase letters ($26^9$) afford almost two trillion more permutations than seven mixed-case characters ($62^7$). This might be a design improvement for link shortening services. Longer lowercase-only links might be quicker and easier to enter than shorter ones that use a variety of characters. Of course, a seven-character link from *bit.ly* that happens to be all lowercase will be faster and easier to enter than a nine-character all-lowercase link. But given the huge number of possible links, we need to know, on average, which shortening schema we should expect to enable quicker and easier transcription of links.

## 2. Systematic analysis

### 2.1. General modeling approach

We developed Monte Carlo models[1] that simulate the keypresses required to enter links. Our target strings were the link IDs at the end of a shortened URL (e.g., *S9T7R4B* or *T6 × 54RS*). The space of all possible permutations for a *bit.ly*-like link shortening schema is large, $62^7$, so we used a smaller sample of one hundred million ($10^8$) link IDs that we generated stochastically. The link IDs were generated by randomly and independently picking seven characters from the 62 options (A-Z, a-z, 0-9); repeats were permissible.

We developed two kinds of model. One predicts the number of taps required to type a given link ID. The other predicts how many milliseconds a given link ID will take to type. In our models we assume an *ideal performer* (see Gray et al., 2006). For the tap-optimizing models we assume that the user always uses the fewest number of taps to enter a link ID. This means we assume that the user has perfect knowledge of the taps required to move between character modes and knows how to minimize mode switches across a whole string before typing begins. Our ideal performer does not make errors (but see Banovic et al., 2013; Wobbrock et al., 2008 for details on how pointing errors can be modeled). Likewise, for the time-optimizing model we assume an ideal performer with perfect knowledge. The ideal performer always types a link in the shortest possible time.

We developed both tap- and time-optimizing models because they have complementary strengths and limitations. Tap-based models are straightforward to construct given a description of the keypresses required to enter a string. They do not require the empirical fitting of human performance data (e.g., typing speed). Time-based models are more complex but are better able to capture subtle differences in the time costs of keyboard interactions (e.g., making a double-tap vs. a single long keypress to change between keyboard modes).

[1] Models available from https://www.sjjg.uk/short-links-models/.

Mobile platforms have different keyboards. Each of these works differently. We wanted to know whether our proposal for a nine-character lowercase schema would be superior across a range of platforms. To this end we model the iOS 8 ('iOS'), Android 5.0 ('Android') and Windows Phone 8.1 ('Windows') keyboards.

### 2.2. Tap-optimizing model

Keystrokes per character (KSPC) is a measure of typing performance that affords simple comparisons of keyboards (MacKenzie, 2002; Varcholik et al., 2012). This metric maps well to taps on touchscreen keyboards. The model for our proposed design solution is simple. Any given link ID formed of nine lowercase characters will take nine taps to enter if starting in lowercase mode (see the bottom row of Table 1). This is because there is no mode switching required for entering a string of lowercase characters when starting in lowercase mode.

Developing a model for seven-character link IDs containing mixed-case letters and numbers is more challenging. Typing numbers and capital letters requires extra taps to make mode switches. Links with runs of characters from a particular mode (e.g., *bit.ly/ABCD123*) can be entered more quickly than links that require a mode change for every character (e.g., *bit.ly/A1B2C3D*).

Each keyboard operates differently. For instance, to access capital letters on the three keyboards that we consider in this paper, users can tap the shift key once. If several capital letters in a row are required keyboards can be switched to the caps-lock mode. On the iOS keyboard this is achieved with a double tap of the shift key. On Android and Windows keyboards a long press also enables caps-lock mode. The tap-optimizing model attempts to minimize the number of taps. It chooses single long presses over quick double taps.

To access number-mode users must first tap the '123' key. Once in number mode, users have to tap the 'ABC' key to get back to the lowercase keyboard. Therefore, to enter a number and return to the lowercase keyboard requires three taps. On the iOS keyboard, activating number mode deactivates caps-lock. Conversely, on the Android and Windows keyboards caps-lock remains active even after number mode has been activated. We made every effort to model all keyboard idiosyncrasies of this kind.

The accessibility of particular symbols varies across the platforms. In iOS and Windows the forward slash character (i.e., /) is accessed through the number mode. This means at the moment of entering the first character of a link, a mode switch is required in order to enter any non-number character. To enter our lowercase links, an additional mode switch is required at the start. We ran the iOS and Windows models both with this additional mode switch and without it. We found no practical difference in the results. We therefore keep with the simpler analysis as it provides a consistent baseline for comparison across keyboards. In other words, all models make the assumption that the starting position for entering the link IDs (i.e., the part of the URL after the forward slash) is in the lowercase mode.

The number of mode switches (and therefore taps) required to enter a randomly generated link ID varies. This means that, unlike lowercase links, the number of taps required to enter a mixed-character link is defined by a distribution, rather than a single value. The *bit.ly* schema will produce links that require anywhere between seven taps to enter for a link all in lowercase characters (e.g., *bit.ly/lqexieb*) to seventeen taps for a combination of capitals and numbers (e.g., *bit.ly/S9T7R4B*).

For our proposed nine-character lowercase schema, all possible links take nine characters to enter. Whether the link is *bit.ly/opsnebzms* or *bit.ly/xvcbsdneu* we would expect the link ID to always take nine taps to enter, assuming an ideal performer. We wanted to know how the seven-character mixed-case schema compared.

**Table 1**
Three examples of the patterns of taps required to enter the ID portions of the links on the default iOS 8 keyboard. Total tap counts are only for the link IDs and exclude the 'Go' tap and the domain name.

| Link ID | Taps | Total |
|---|---|---|
| 9ST74ex | 123 9 ABC ⇧ ⇧ S T 123 7 4 ABC E X Go | 13 |
| lqexieb | L Q E X I E B Go | 7 |
| biexeqlbe | B I E X E Q L B E Go | 9 |

### 2.2.1. Results

The results reflect favorably on our nine-character lowercase schema when compared to the seven-character mixed-case schema. Given a randomly generated link ID, we would expect more taps (top row, Table 2) to be required for a mixed-case link ID even though it comprises fewer characters (7 vs 9). Our intuition was that longer lowercase link IDs would require fewer taps. Our modeling allows us to quantify this advantage. Depending on the keyboard, the chance of a seven character mixed-case link ID needing *more* than nine taps to enter is 42-92%; across the platforms a link generated by the mixed-case schema is more likely than not to need more than the nine taps required to enter nine lowercase letters.

How often will the mixed-case schema generate a link that requires fewer taps than a nine-character lowercase link? As can be seen in the bottom row of Table 2: unlikely. On the Android keyboard, which is superior by virtue of having numbers accessible from the QWERTY keyboard through a long press, the mixed-case schema will require fewer than nine taps (i.e., the number of taps to enter a nine-character lowercase ID) on one in five occasions (19%). On the iOS keyboard this is a one-in-one-hundred event (1%).

To summarize, a link produced from the lowercase schema is very likely to require fewer taps than a link generated by the mixed-case schema. This is despite being two characters longer. Measured by taps alone, our nine-character schema is clearly superior to a *bit.ly*-style link.

## 2.3. Time-optimizing model

The tap-optimizing model we developed is easy to build and generalize. It affords simple comparisons. It also makes several assumptions that might hinder its reliability. One is that double taps count as two taps even though the execution of a double tap is faster than two independent single taps. This means that the tap models overestimate the difficulty of entering strings involving double taps. Thus, the tap-based model for the iOS keyboard might make the keyboard appear more onerous to use than it might actually be.

**Table 2**
Mean, standard deviation, range and lowercase model comparisons for results of tap-optimizing model. Nine-character lowercase links are modeled as requiring nine taps to enter.

| | iOS | Android | Windows |
|---|---|---|---|
| $E(X)$ of taps | 12 | 9 | 11 |
| $\sigma$ of taps | 2 | 1 | 2 |
| Range of taps | 7–17 | 7–12 | 7–1 |
| $P > 9$ taps | 92% | 42% | 87% |
| $P < 9$ taps | 1% | 19% | 3% |

Consider, for instance, the final two characters of the link ID *2e45pRF*. If double taps are treated as two independent taps, then entering *Shift*→*Shift*→*R*→*F* using the caps-lock mode uses as many taps as *Shift*→*R*→*Shift*→*F*, where caps-lock is not used. But we would expect that entering the caps-lock mode with a quick double tap would be faster than having to tap shift before each of the characters.

The second assumption that the tap-optimizing model makes is that long presses are equivalent to normal taps. This makes keyboards that use long presses appear quicker to use than they might actually be. In the case of the Android keyboard, long presses can be used to activate caps-lock or use the numbers on the top row of the keyboard (see Fig. 2). These are modeled as single taps, despite being longer in duration. This means that the tap-optimizing model will prefer long presses over double taps. Consider, for instance, the link ID *2345ERF*. To enter the first four characters, the tap-optimizing model will choose to enter *2*→*3*→*4*→*5* by long pressing the *w,e,r* and *t* keys on the keyboard. This comes to four taps. The alternative –and likely faster– option is to change the mode of the keyboard and then enter the numbers, i.e., *?123*→*2*→*3*→*4*→*5*→*ABC*. This sequence comprises six taps and so is ignored by the tap-optimizing model. To overcome some of the limitations of the tap-optimizing model, we developed a time-optimizing model. Using timing parameters from the literature gave us estimates for the transitions between keys and modes. Critically, the parameters we use are for typing non-word strings like passwords (e.g., Gallagher and Byrne, 2015; Greene et al., 2015; Greene et al., 2014; Jakobsson and Akavipat, 2012) that map well to our problem domain. Our timing estimates are from Gallagher and Byrne (2015) unless otherwise stated.

In the time-optimizing model we model double taps and long presses individually. To model long presses we added 500 ms to the duration of a keystroke. This value was obtained from the Android 5.0 source code (DEFAULT_LONG_PRESS_TIMEOUT). Double taps were modeled by adding half of the upper time limit for a double tap (i.e., half of 300 ms) to a keystroke. This upper limit was also obtained from the Android 5.0 source (DOUBLE_TAP_TIMEOUT).

The Android and Windows keyboards give the option of a long press or a double tap to activate caps-lock. With the parameters we used, a double tap is always faster than a long press for activating caps-lock. Therefore, caps-lock was modeled as a double tap for all three keyboards. Long presses are only incorporated into the model for entering numbers on Android using the top row of the QWERTY keyboard (Fig. 2). We model long presses as taking (550 ms + 500 ms), which is quicker than the time to change to the number keyboard (modeled as 1500 ms).

We modeled the inter-key interval between keystrokes in the same mode (e.g., a number followed by a number or a lowercase


**Fig. 2.** iOS, Android and Windows Phone keyboards with respective distributions of tap counts required in order to enter seven-character link IDs (e.g., *2345eRF* or *S9T7R4B*) using numbers and uppercase and lowercase characters. No link IDs required more than 12 taps on the Android keyboard. The iOS and Windows Phone keyboards required between seven and seventeen taps.

letter followed by a lowercase letter) as taking 550 ms. Due to the change of keyboard mode required, entering or leaving the number mode and entering a letter takes 1500 ms. When moving between uppercase and lowercase modes, users may use caps-lock or shift depending on which type of characters follow. Gallagher and Byrne's (2015) data for these transitions are not broken-down by strategy, so we must infer some of the transition times. The time to switch between cases is shorter than the transition time for the changing to the number mode because the keyboard layout does not change when moving between uppercase and lowercase keyboards. We model using the shift key and tapping the target character as two 550 ms taps for a total of 1100 ms. Entering caps-lock requires a further 150 ms for the double tap of the shift key (1250 ms) to engage the mode, plus a further 550 ms to exit it. On the iOS keyboard if caps-lock is enabled and a user changes to the number mode, caps-lock is also removed. On Android and Windows keyboards caps-lock remains turned on until it is turned off again by a user. The models account for these idiosyncrasies and look ahead through the target strings to minimize time costs.

The models assume perfect knowledge of the keyboard and the keystrokes and mode transitions required to minimize typing duration. Human performance would, of course, vary. Our model used a bracketing-like heuristic (Brumby et al., 2007; Kieras and Meyer, 2000), but we only consider the minimum (i.e., fastest) bracket. This represents a *particularly* aggressive pruning of the strategy space; more nuanced approaches exist (Dayan, 2014; Howes et al., 2009; Zhang and Hornof, 2014). But our model is intended to be indicative of the suitability of competing link ID schema options, rather than a complete exploration of the strategy space. Our approach is broadly consistent with several KLM-like models that have been developed for modeling touch input (El Batran and Dunlop, 2014; Greene et al., 2013; Li et al., 2010; Rice and Lartigue, 2014), except that we do not specify generalizable basic operations. Instead we focus just on the combinations of transition times between modes and characters.

#### 2.3.1. Results

The results for our nine-character model are very simple. The time to enter a lowercase letter from the lowercase mode is 550 ms. Nine consecutive lowercase letters yields a model estimate for 4950 ms for any permutation.

The lowercase schema again yields better performance. The distributions of entry times for each of the three keyboards are shown in Fig. 3. The expected time to enter a link generated using the mixed schema is shown in Table 3. Overall the results mirror those of the tap-optimizing model, but are even more weighted in favor of the lowercase schema. Across all keyboards approximately 90% of seven-character mixed-case IDs are expected to be slower to enter than nine-character lowercase IDs.

The fastest possible input is estimated to be 3850 ms for all three keyboards. This is for the entry of seven lowercase characters. This interaction is particularly fast because no mode switches are required. These results are congruent with those of the tap-optimizing models, which showed entering seven lowercase characters requires the fewest taps (i.e., seven). Approximately one-in-a-thousand links generated by the seven mixed character schema are of this variety.

### 2.4. Conclusions from models

The models show that link IDs from commonly used link-shortening services have large input overheads. Depending on the keyboard, we can expect a given *bit.ly* link to take between one and three (i.e., 11–33%, see *E(X)*, Table 2) more taps to enter than a nine-character lowercase link. These extra taps and mode switching translate to average entry being between 777 ms and 1938 ms (i.e., 16–39%, see *E(X)*, Table 3) slower than a nine-character lowercase link. In other words, the output of existing link shortening services, like *bit.ly*, are likely to require *more* taps and *more* time to enter than a service that used solely lowercase characters, even if those links have to be *longer* to maintain the size of the set of links. To corroborate the inferences drawn from our ideal performer models,

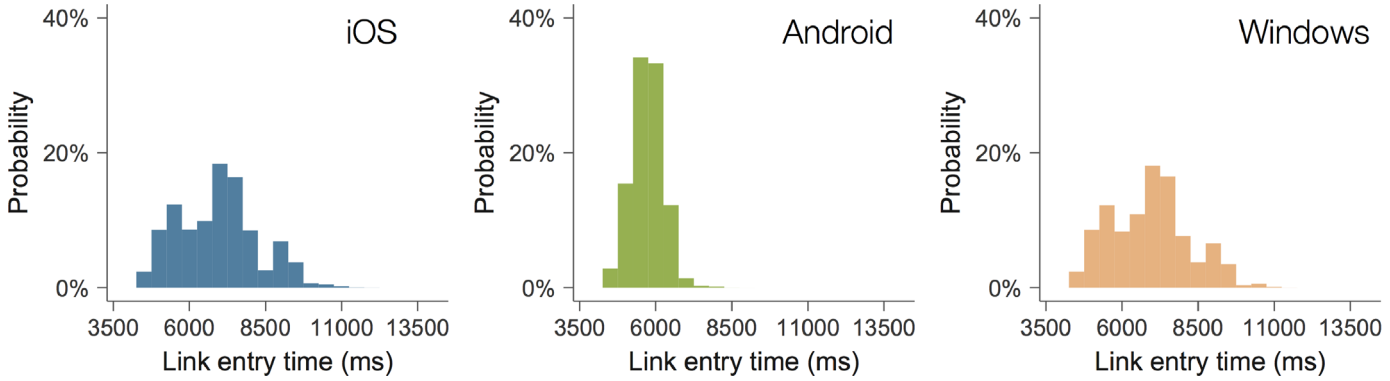


**Fig. 3.** iOS, Android and Windows time distributions for the entry of seven-character link IDs (e.g., *2345eRF* or *S9T7R4B*) using numbers and mixed-case characters. For plotting model output was rounded and binned in 500 ms buckets.

**Table 3**
Mean, standard deviation and range for results of time-optimizing model. The fastest model is entering seven consecutive lowercase letters (3850 ms). The estimate for entering nine lowercase letters is 4950 ms.

| Keyboard | iOS | Android | Windows |
|---|---|---|---|
| $E(X)$ | 6888 ms | 5727 ms | 6868 ms |
| $\sigma$ | 1289 ms | 547 ms | 1264 ms |
| Fastest | 3850 ms | 3850 ms | 3850 ms |
| Slowest | 12150 ms | 9250 ms | 11600 ms |
| $P > 4950$ ms | 93% | 89% | 93% |
| $P < 4950$ ms | 3% | 3% | 3% |

we ran an experiment in which users typed URLs on smartphones.

## 3. Experiment

Conducting an experiment in which users type links on their smartphones allowed us to investigate additional aspects of behavior not captured by our models. In particular, people usually interleave their attention between the device they are typing on and the target link they are copying. Memory limitations mean that target information is often broken-up and copied in smaller chunks (Howes et al., 2015; Janssen et al., 2010, 2012; Payne and Howes, 2013). Longer strings require more chunks (Salthouse, 1986; Smith et al., 2008). It seems reasonable, therefore, to assume that copying a longer nine-character link might require more chunks than copying seven-character links. In practice this might mean entering nine-character links is slower, because each chunk requires attention to be shifted from device to target. We did not model chunking behavior, so our models do not account for potential differences in time spent interleaving as a function of schema type.

Our models were also ideal performer models. They never made mistakes. They were able to look ahead through the links and treat them holistically. Because the models never erred and always knew the optimal combination of keystrokes to enter, they accounted poorly for the costs of switching keyboard modes. Making mode switches involves changes of context that, for instance, increase the chances of errors being made (Gallagher, 2015, p. 62). Given these factors, it is likely that our models also underestimated the costs of entering seven-character links composed of mixed-case letters and numbers.

We ran an experiment to determine whether longer links that are easier to enter provide a meaningful improvement in response speed over more complex but shorter links. We also solicited participants' subjective experiences of entering different links. To help us focus on the effects of link type on performance, the experimental procedure allows us to closely examine the entry of the seven- or nine-digit link IDs.

### 3.1. Gamification

Typing involves a speed-accuracy trade-off. Going quickly increases the likelihood that mistakes are made. Going slowly means spending unnecessary time on a task. A traditional laboratory study would need to incentivize participants to type accurately without inducing them to be unrealistically fastidious. Finding the equilibrium that encourages participants to work quickly while still avoiding typos can be difficult. Indeed, characterizing these trade-offs is itself an area of research (e.g., Healy et al., 2004).

Our focus here is not on exploring speed-accuracy trade-offs. We only wanted an effective way of encouraging participants to reach a satisfactory equilibrium. The study we present here places participants in a competitive situation involving points and leaderboards (i.e., gamification). By making the game competitive, participants are motivated to try to work quickly. High penalties for errors moderate this impulse, though; there is a balance to be struck. This encourages participants to find an equilibrium in the speed-accuracy trade-off that does not put too much emphasis on either (Trommershäuser et al., 2005).

### 3.2. Method

#### 3.2.1. Participants

Nineteen participants (nine male) took part in the study. Participants' ages ranged between 23 and 42 ($M=30$ years, $SD=5$ years). Participants were recruited opportunistically from staff at a research laboratory. We were not concerned about participant non-naïvety: knowing the aims of the study would not have improved participants' performance, which is largely a function of individuals' pre-existing typing ability.

Data were collected over a six-week period. Over this period points were accumulated in each round of the study. A round consisted of the display of one short link that was entered by two or more participants. Scores were awarded for response speed and accuracy. Cash prizes were awarded to the participants with the six highest cumulative scores. Snacks were also made available during each session to encourage participation.

#### 3.2.2. Design

The experiment had one within-subjects factor, link type. There were two conditions, *mixed* and *lowercase*. In the *mixed* condition participants were asked to enter seven-character links containing numbers and mixed-case letters. In the *lowercase* condition participants were asked to enter nine-character links containing only lowercase characters.

There were two dependent variables. The first was *response speed*. We measured the time from the target link being shown on the television to the moment that the request from participants' phones got to our server. The second measure was *response accuracy*. All requests made to the server after the target link appeared were recorded as responses, whether they matched the target link or not.

### 3.2.3. Materials

The experimental procedure was inspired by classroom response systems like *Kahoot!* (Wang, 2015; Wang et al., 2008). These platforms allow several people to participate simultaneously while competing in a lighthearted way that maintains attention.

Participants brought their own phone. The only requirements were that it had a touchscreen keyboard and a browser. Device platform, for example Android or iOS, was recorded. The browser used in the study was also recorded.

As part of the setup of the experiment, we asked participants to tap the address bar in their preferred browser and provide a screenshot of the keyboard that popped up. The screenshot was uploaded as part of the consent process. The experimenter was on hand to aid participants who did not know how to create screenshots on their device.

Target links in the study were entered directly into browser address bars. We chose this approach over using a field in a form because it retains important features. For instance, address bars can invoke browser-specific custom keyboards that are not available in forms. Modern mobile browsers also display potentially distracting autocomplete suggestions when typing URLs. We wanted to emulate the process of opening a blank tab and entering a short link –say, during a lecture– as closely as possible within constraints of an experimental setting.

Three different displays were used in the study. The first was private and gave the experimenter control over the study. The second was a large wall-mounted television (see Fig. 4). This showed the target link during trials and scores between trials. The third kind of output was sent to participants' devices when they navigated to the home page of the experiment. This on-device output controlled participants' progress through the study, showed their scores for each round, and allowed for links to be rated.

After entering each link, participants rated it on a slider. The slider handle started in the middle of the screen (50). The left side was labelled *Easy* (0) and the right side labelled *Hard* (100). Only the position of the handle –and not its value– was visible. Fig. 5 illustrates the rating interface.

Links were pseudo–randomly generated for the *mixed* condition. The objective was to evenly sample simple link IDs (e.g., *utisbnd*) and complex links (e.g., *E9b5T3e*). To do this we used the tap-based iOS model to generate links that would need between seven taps (i.e., simple) and seventeen taps (i.e. complex). When a *mixed* trial was requested the condition allocator examined all *mixed* rounds. It computed the complexity (7–17) that had appeared the fewest times. It then produced a link of that complexity. This approach generated analyzable numbers of trials for all 11 complexities. If we had generated *mixed* links at random, links with uncommon complexities (e.g., 7, 17; see Fig. 2) would have been unlikely to appear in the study.

For the *lowercase* condition, links could only have one complexity: nine taps. Links for the lowercase condition were thus generated randomly. Whether a round was *lowercase* or *mixed* was determined by counting which type had been run in the fewest rounds. That type was run next.

The speed and accuracy of participants' responses were used to calculate scores for each round. If participants entered the link incorrectly they scored zero for the round. The first participant to correctly enter the link scored 1000 for the round. Other participants started at 900 points and had one point deducted for every 20 ms they were behind the fastest player on that round. There was a floor for correct answers of 100 points.

### 3.2.4. Procedure

Participation was in ad-hoc groups of 2-7. Groups were formed opportunistically at short notice. Participants sat in the testing room with a view of the screen (see Fig. 4).

First-time participants were given an introduction, gave consent and entered their demographic information. Participants picked a username of their choice. They were told that it would be visible to others. A cookie was stored on the participant's device; returning participants' details were loaded automatically. Changing devices did not disqualify participants, but data obtained after a change of device were discarded from analysis. If a participant deleted the cookies on their device they would be sent back to the sign-up page. The need to retain cookies for the duration of the study was explained to participants. No participants deleted the cookies used by the task during the course of the study.

Before each round, a button appeared on each participant's phone that opened a blank browser tab. Once participants were ready, a link appeared on the large television. Participants had to type this link into their browsers. After entering the link, the blank tab closed and participants were informed whether they had entered the link correctly. They were also given a score. This information was communicated in the tab still open on their device.

Once all participants in a round had entered the link, the target link being displayed on the television was replaced with a leaderboard. This displayed all participants' scores from the round just completed along with cumulative scores for the whole session. Sessions consistent of multiple rounds and ran until one participant chose to leave.

Prior to target links appearing on the television, participants

**Fig. 4.** Location of experiment. Participation was in ad-hoc groups. Links were shown on a television screen.

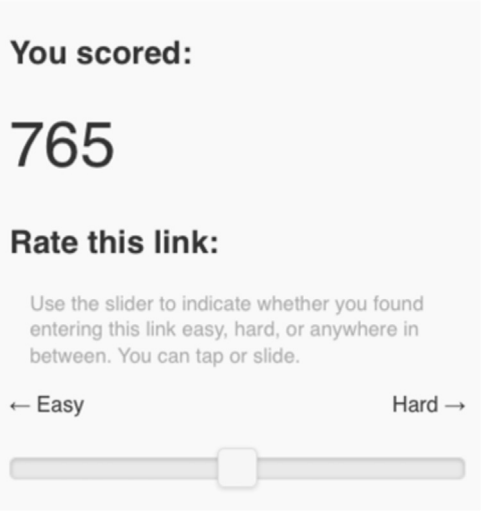


**Fig. 5.** The rating interface. Participants were shown their score and asked to rate how hard or easy they thought the link was to enter. The slider's handle always started in the center of the bar.

were asked to enter the first part of the domain (i.e., *fbr.me/*) into the address bar of their browser. This gave an additional degree of control to the experiment by limiting the scope for individual differences in strategies. For example, some participants may have tried to enter the whole link when it appeared, but other participants may have prepared for the appearance of the target link by typing or copy-and-pasting the static *fbr.me/* part (see Charman and Howes, 2003). This approach also had the advantage of allowing us to make direct comparisons between the models and the empirical data. Removing the static parts of the links made the differences between the most and least complex link IDs more salient in the results.

### 3.3. Results

#### 3.3.1. Participation

Nineteen people participated in the study. A visual inspection of keyboard screenshots revealed three participants using non-English keyboard layouts such as QWERTZ or using keyboards containing non-English characters on the default mode (e.g., Ø). One participant used a custom keyboard with numbers on an extra line above the QWERTY keyboard. As it was our intention that participants use the keyboard they were most familiar with, the use of 'non-standard' keyboards did not disqualify participants from the study or inclusion in subsequent analyses. One participant changed devices during the study, but continued to participate. All data obtained after they switched devices were discarded.

Twenty-nine sessions were run. Sessions comprised 2-7 participants with a mean and median of four ($SD=1$). Sessions lasted 1-24 rounds with a mean of seven rounds and a median of six ($SD=5$). Each participant completed 4-148 rounds, with a mean of 44 and median of 31 rounds ($SD=41$). Across the whole study, a total of 828 individual trials were completed. Participants took part in 1–16 sessions. Differences in degrees of participation are accounted for in our analyses of data. The mean and median number of sessions completed was six ($SD=4$). Six participants were Android users and contributed 337 rounds (40%). The rest of the participants used iOS.

#### 3.3.2. General performance and outliers

Participants submitted an incorrect link on 111 of the trials (~13%). Across all 717 correctly completed trials, the mean response time was 8407 ms ($SD=3885$ ms). The distribution of correct response times is illustrated in Fig. 6. Two trials were affected by technical issues with participants' devices that extended their duration (to 55-s and 63-s, respectively). These trials were labelled as outliers and were discarded from all analyses that appear from this point onward.

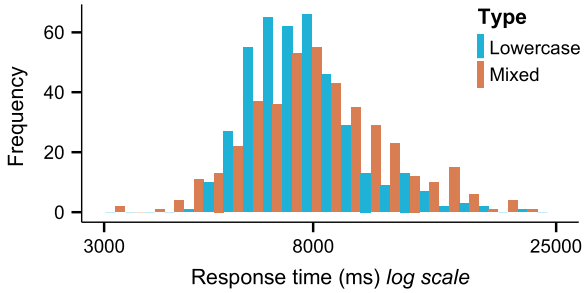


**Fig. 6.** Response times for all trials, correct and incorrect, by link type (lowercase vs mixed).

#### 3.3.3. Effect of link complexity on performance

Our computational modeling showed that mixed-case links require varying numbers of mode switches –and therefore taps– to enter. Some links have low complexity and are simple to enter (e.g., *lqexieb* requires seven taps). Others are more complex and require many mode switches (e.g., *S9T7R4B* requires seventeen taps on an iOS keyboard). As part of the *mixed* condition, this experiment presented links with complexities of 7–17. This range was a prediction of our ideal performer model typing on the iOS keyboard. We wanted to confirm that the links that our model predicted would require more taps actually took participants longer to enter.

We took a linear mixed effects modeling (LMEM) approach. We used the response times from the *mixed* condition as the outcome measure. (Recall the *lowercase* condition has no variations in complexity.) We included *complexity* and *device-type* as fixed effects and *participant-id* as a random effect. The model accounted for the possibility that participants' performance may have varied differentially with complexity of the links presented to them (see Barr et al., 2013): poor typists might struggle more with complex links. We included device type as a fixed effect because our modeling suggested Android keyboards might be quicker, particularly for links with higher complexities (see Fig. 2).

The results of our modeling (Table 4) yielded no evidence of an interaction between complexity and device type, so we focused on the main effects. There was strong support for an effect of complexity on response time and for an effect of device on response time. The statistical modeling indicated two things. The first was that link complexity (as determined by estimated tap counts) was an excellent predictor of entry time (see Fig. 7). Tap-based models are quicker and easier to derive than time-based models, so this finding is of practical significance. The second was that keyboard shortcuts might be largely unused. The Android keyboard offers a few tap-saving shortcuts. Our modeling suggested that the Android keyboard ought to be quicker to type on, particularly as links

**Table 4**
Linear mixed-effect modeling results. Parameters: T = Time, C=Complexity, P=Participant, D=Device. *Ds* are AIC-corrected log-likelihood ratios. $\chi 2$ and p are from the comparison of the null and alternative models and provided for reference.

| Effect | $H_0$Model | $H_a$Model | $D_{AIC}$ | $\chi^2$ | *p* |
|---|---|---|---|---|---|
| Complexity × Device | T~(1+C\|P)+C+D | T~(1+C\|P)+C+D+C*D | −2.9 | 0.01 | .94 |
| Complexity | T~(1+C\|P)+D | T~(1+C\|P)+C+D | 34.1 | 25.6 | <.001 |
| Device | T~(1+C\|P)+C | T~(1+C\|P)+C+D | 4.4 | 5.0 | .02 |

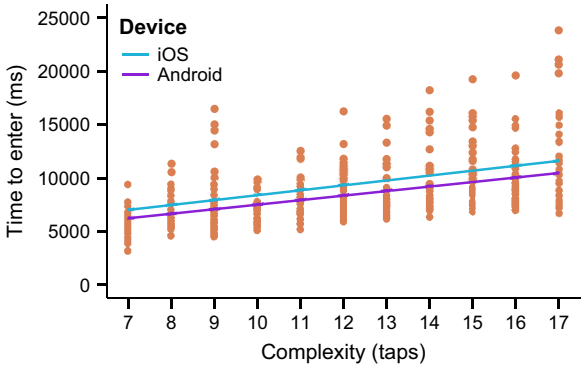


**Fig. 7.** Response time for data collected in the *mixed* condition. Trend is for the effect of complexity.

became more complex. In the experimental results there was a main effect of device on response speed but, critically, no interaction. This suggests that participants using Android were just naturally faster responders: the lack of an interaction implies that there was no particular benefit of using Android keyboards for more complex links. If participants *were* using shortcuts on the Android keyboard, we would expect their advantage over iOS users to grow with increasing link complexity. We did not observe such an interaction effect (see Fig. 7).

### 3.3.4. Effect of link schema on response speed

More complex *mixed* links took longer to enter. We knew from our modeling (Table 2) that most *mixed* links needed more taps to enter than longer *lowercase* links. On an iOS keyboard, a tiny minority (1%) of *mixed* links were less complex than *lowercase* links. But which was faster in our experiment? Our modeling suggested that nine-character *lowercase* links would be faster to enter.

We could not directly compare response times from the *mixed* condition with those from the *lowercase* condition because complexities in the *mixed* condition were sampled uniformly: a link of complexity seven was as likely to be generated as a link of complexity fourteen. The underlying distribution is, however, not uniform. As Fig. 2 shows, some complexities are far more likely than others. If we had instead sampled from the underlying distribution, our 828 trials would likely only have had one trial with a complexity of seven and no trials at all of complexity seventeen.

To make a fair comparison between the *mixed* and *lowercase* conditions, the data had to reflect the underlying distribution of complexities. There were two options. One was to use a subsampling (or resampling) approach, the other was to use interpolation to estimate data that fits the underlying distribution. Given the large differences between the sampling and underlying distributions, a vast amount of interpolated data would be required. We therefore took the conservative approach of using subsampling.

The first step of our analytic procedure was to compute mean response times for correct trials in the *lowercase* condition for each participant. Subsampling is only required on data from the *mixed* condition. To perform the subsampling we used the discrete probability distribution for iOS devices illustrated in Fig. 2. The most frequent complexity in the underlying (iOS) distribution is eleven: all 27 responses of complexity eleven were retained. For the other complexities, responses were randomly sampled in proportion to the underlying distribution. For instance, no samples were drawn for complexities 7, 16 or 17. Eight samples were drawn with complexities 9 and 14. These subsamples were then grouped by participant and mean response times were calculated.

The result of our resampling was two sets of data: mean response times for *lowercase* trials and subsampled mean response times for *mixed* trials. Due to random sampling, not all participants had data for *mixed* trials. Four participants were discarded for this reason, leaving 15 participants. Mean response time for *lowercase* links was 8600 ms ($SD$=1950 ms). Mean response time for *mixed* links was 9792 ms ($SD$=2700 ms). A paired $t$-test was used to determine whether average response times differed between the two conditions. It showed that participants were significantly faster entering *lowercase* links than they were *mixed* links, $t(14)=2.88$, $p=.012$, $d=0.75$ 95% CI [306, 2076]. This result indicates that given the likely output of *lowercase* and *mixed* shortening schemas, we would expect *lowercase* links to be faster to enter. This, despite *lowercase* links containing two characters more than the *mixed* links.

The results of the experiment differed considerably from the predications of the time-based model. Time to enter *lowercase* links was estimated by the model to be 4950 ms. Mean entry time for *lowercase* links in the empirical study was 8600 ms. Time to enter *mixed* links was estimated between 5727 ms and 6888 ms, but in practice it was 9792 ms. One possible reason for the discrepancy might be the model's failure to account for the time participants spent glancing between the television and their devices. Another possible factor in the differences might have been correction behavior: the model never made typing errors, but participants often reported making typos that were then corrected before submission. (Corrected errors were often salient to the experimenter by a particularly slow response from a participant.) These correction delays would have increased average response time (Banovic et al., 2013). Although the time-based model was inaccurate in predicting absolute time required to type links, both the tap- and time-based models were accurate in their predictions about the relative differences between conditions and complexities.

### 3.3.5. Effect of link schema on accuracy

The results show that longer lowercase links are faster to enter than shorter mixed-case links. Does this increase in response speed come at the cost of accuracy? No. A response was counted as an error when the URL a participant requested did not exactly (i.e., including case) match the one that they were supposed to enter. Errors were made on a total of 111 trials, for an error rate of 13%. Of the 412 *lowercase* trials, 54 (13%) were errors. Of the 416 *mixed* trials, 57 were errors (14%). Put simply, there was no effect of link type (*mixed* vs *lowercase)* on error rate.

### 3.3.6. Error types and magnitudes

Next, we considered the magnitude of errors. For incorrect trials, how far were participants from entering the correct link? We computed how many edits would be required to transform the entered –incorrect– string into the correct string (e.g., *hs4rfgS* is two edits from *hr4rffS*). A case-sensitive Damerau-Levenshtein (DL) distance was used to compute the number of edits between target links to entered links. The closer to the target input the entered link was, the smaller the DL distance.

Of the 111 errors made, 100 (90%) had a DL distance of one. This means that a single insertion, deletion, transposition, or substitution would have corrected the entry. The highest DL distance was six, and was caused by inadvertent activation of caps-lock on an iOS keyboard. The low DL distances are indicative of small slips in typing (e.g., hitting adjacent keys) and suggest a high degree of conformity with the demands of the experiment.

### 3.3.7. Subjective rating of difficulty

After each trial we asked participants to rate, using a slider, whether they found the link hard or easy to enter. Did participants' subjective experience of the different link types match their performance? Were less complex links perceived as being easier to

enter?

Participants' scores for a round were visible when they rated. Subjectively harder links may have taken longer to enter, or participants may have been influenced by their score and rated rounds where they scored highly as being easier. Linear mixed modeling suggested that after accounting for the random effects of *participant-id* and *round-score* (i.e., after accounting for participants' individual propensity to have rated higher-scoring responses as easier), the complexity of *mixed* links remains an excellent predictor of reported link difficulty ($D_{AIC}=106.5$, $\chi^2=75.8$, $p<.001$). Participants found it harder to enter shorter, but more complex links. Using *lowercase* links over *mixed* links enables faster input and is likely to be less frustrating for users too.

## 4. Visual appearance

So far we have considered the practical advantages of typing longer lowercase links over shorter mixed-case links. Despite having more characters, longer lowercase links are objectively

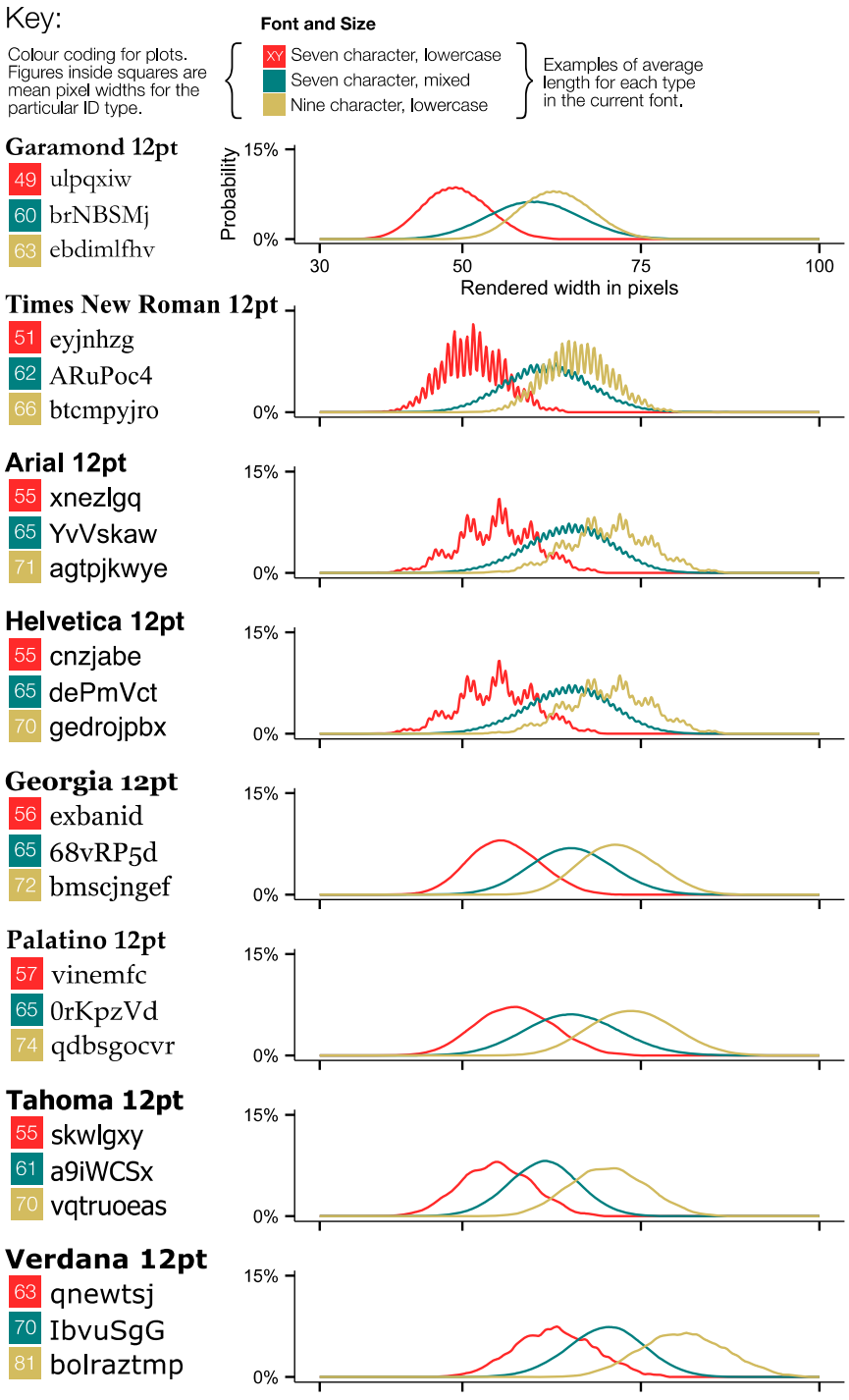


**Fig. 8.** Distribution of link ID widths in pixels for nine common fonts. With variable-width fonts, the average increase in width of nine-character lowercase ID is 11% over seven-character mixed-case IDs. Fonts are plotted from most space efficient (top) to least space efficient (bottom).

quicker and subjectively easier to enter. But the additional length of nine-character link IDs has implications for the rendering of links on screens. Nine-character lowercase link ID have two characters more than seven-character mixed-case IDs. This is a 29% increase in length.

One might imagine that links with fewer characters might be easier to fit into tight spaces on printed documents (see Fig. 1) or on small screens. This intuition is not entirely accurate. Even though the two additional characters in nine-character links represent a 29% increase over seven characters, far less physical space is required. This is because most commonly used fonts are 'variable width'. This means that each glyph (e.g., e, #, L, 9) has a different width. A capital letter is typically wider than a lowercase letter. Characters like 'l' take up less horizontal space than characters like 'm'. Most variable-width fonts also make use of kerning. This means the gap between two consecutive characters is not fixed and can vary.

How much of an effect does kerning and glyph-width have on the physical size of link IDs when they are rendered? We again turned to simulation to explore this question. We selected eight common variable-width fonts (see Fig. 8). For each of these fonts we looked at three types of link ID: seven- and nine-character lowercase link IDs and seven-character mixed-case link IDs. For each link ID type we generated $10^6$ link IDs for each font and rendered them in a browser. We then measured the width of the link ID in pixels as they were rendered by a browser.

Fig. 8 plots the distribution of widths for each of the three link ID types for each of the fonts. Two things should be clear from these plots. The first is that some fonts (e.g., Garamond) are more space efficient than others (e.g., Tahoma). The second is that the distributions are highly variable. Local maxima and minima appear for some fonts, but not for others. The idiosyncratic variation in the distributions is the result of differences in kerning and glyph-width for each font.

Our most salient finding was that across the eight variable-width fonts sampled, nine-character lowercase link IDs are, on average, only 11% wider than seven-character mixed-case links (see Table 5). This is much less than the 29% increase one might expect if only character counts were considered (i.e., seven characters versus nine characters).

Our analysis shows that nine-character lowercase links are only marginally wider than seven-character mixed-case links. Times New Roman is only 6% wider, while Arial and Helvetica are 9% wider. For most fonts, little extra space is needed for nine-character lowercase links compared to seven-character mixed-case links. These results show that one of the potential drawbacks of longer lowercase links –the fact they take up more space– is not as serious as it might seem at first glance. This lends further support to our conclusion that nine-character lowercase links may be better in practice then seven-character mixed-case links.

**Table 5**
Mean widths of link IDs in pixels for seven- and nine-character lowercase link IDs and seven-character mixed-case link IDs. The *Difference* column indicates the difference in rendered width between nine-character lowercase and seven-character mixed-case links. All measures are in pixels and taken at 12 pt.

| Font | Seven-character lowercase | Seven-character mixed case | Nine-character lowercase | Difference (%) |
|---|---|---|---|---|
| Garamond | 49 | 60 | 63 | 5 |
| Times New Roman | 51 | 62 | 66 | 6 |
| Arial | 55 | 65 | 71 | 9 |
| Helvetica | 55 | 65 | 71 | 9 |
| Georgia | 56 | 65 | 72 | 11 |
| Palatino | 57 | 65 | 74 | 14 |
| Tahoma | 55 | 61 | 70 | 15 |
| Verdana | 63 | 70 | 81 | 16 |

## 5. Conclusion: Link shortening

Using computational modeling and empirical investigation we have shown that slightly longer lowercase links afford a sufficient number of possible link IDs whilst being faster to enter than shorter links with mixed characters. We compared seven-character mixed links, like those generated from services like *bit.ly* with nine-character lowercase links. Our nine-character links were faster to enter and our participants' subjective reports of difficulty seem to suggest a preference for longer lowercase links. It is important to note, however, that our method of obtaining subjective ratings, with ratings collected after scores were allocated, means that our evidence for this preference is indicative not definitive.

On the basis of these findings, we recommend that link shortening services should switch to using lowercase links, or, at least, should offer a checkbox that would allow users to generate mobile-friendly links using the schema we have outlined in this paper. There are two arguments against using lowercase links. First, that moving from seven to nine characters represents a 29% increase in link ID length. Perhaps in character constrained settings this would be problem. Two characters is 1.5% of the total space in a tweet. However, in practice short links rarely appear verbatim. Twitter, for instance, makes all links take-up 23 characters of a tweet, even if the original link was shorter. Soon, Twitter will not count certain kinds of links as part of the character count at all[2]. Using as few characters as possible may have little practical benefit in digital scenarios. Second, moving from seven-character to nine-character link IDs also has a smaller effect on the visual display of links than one might expect. Using computational simulations, we have shown that the 29% increase in character count only translates to a 5-16% increase in link ID pixel width for common variable-width fonts. This marginal increase suggests that when printed or displayed on screens, nine-character lowercase IDs will take up little extra room.

## 6. Generalizability of approach

Up until now we have focused on computationally and experimentally exploring improvements to link shortening services. The claims we make in this work are primarily about this particular application domain. Our approach is flexible, though: it can also be used for evaluating the design of unique identifiers in a variety of scenarios. The only constraint is that the schema for identifiers is systematic and well defined (i.e., it has a known alphabet and length).

In this section we use the simulations described previously to model two other application domains where unique identifiers are likely to be typed into mobile devices with small keyboards. We outline the current schema for unique identifiers in each scenario, report modeling results and evaluate the suitability of current ID schemas given the results of the simulations. These additional application domains serve to illustrate that transcription of unique identifiers is a common use case and that their design can be approached systemically.

### 6.1. Bus-stop identifiers

Transport for London (TfL), the organization that runs buses in

[2] https://blog.twitter.com/2016/doing-more-with-140-characters.


**Fig. 9.** Unique identifiers on Transport for London bus-stops.

London, has five-digit numeric codes on many of their bus-stops (see Fig. 9). Assuming that all ten digits (0–9) are used in these codes, there are a total of $10^5$ unique identifiers available for bus stations. If the scheme omits the zero character (0 and O are easily confused), TfL have 59,049 possible unique identifiers for their bus-stops. In practice, TfL control 19,500 bus-stops. Of these, 2500 are labelled with these five-digit identifiers[3].

These IDs are intended to be entered on mobile devices (see Fig. 9, leftmost panel). Are five-digit numeric codes the best choice? Based on the models we developed for the analysis of short links, these codes take five long presses to enter on Android keyboards. They take six taps to enter on iOS and Windows keyboards because they require an initial mode switch that requires an additional tap.

Three lowercase letters ($26^3$) would provide a sufficient set size to provide unique identifiers to each of the 2500 bus-stops that currently have codes. Even dropping out potentially confused characters (o, i, l, s), four lowercase characters would yield almost a quarter of a million ($22^4$) unique identifiers for TfL to use. Presumably this would be a sufficient quantity to keep-up with TfL's bus-stop building for the foreseeable future. Entering four lowercase characters requires only four taps on all three keyboards. This is a 30% saving for iOS and Windows phones. On Android phones five long presses becomes four quick taps.

Could even fewer taps be required by using a combination of uppercase and lowercase letters together with numbers? We modeled the number of taps required to enter three-character IDs made up of mixed-case letters and numbers (excluding o, i, l, s; 0, 1, 5; O,I,L,S). The results of our modeling, illustrated in Fig. 10 and Table 6, show that reducing the length of the identifier but increasing the number of possible characters is not a good trade-off. On the iOS keyboard, for example, 65% of three character codes made up of numbers and mixed-case letters would require more than five taps to enter. Even using the Android keyboard, three character IDs with numbers and mixed-case letters will, at best, not outperform four lowercase letters. In conclusion, for optimal ease of entry, TfL bus-stops should be identified with four-character strings made up of lowercase letters.

### 6.2. *'Win codes'*

A more complex example of a unique identifier is 'win codes'. These are IDs used in 'buy and win' promotions. They typically appear on bottles of fizzy drinks or packets of snacks. The codes are typed into apps or web-pages and entrants are told whether they have won a prize. One of the key design constraints for these codes is maintaining a sufficiently large set of possible codes that guessing is not worthwhile. The codes in Fig. 11 are ten characters long and formed of uppercase letters and numbers. Assuming that easily confused characters are omitted (I,L,1,O,0,S,5), the total set size afforded by this scheme is $29^{10}$ (over 400 trillion).

Using the same computational models described previously, we can estimate how many taps these codes will take to enter on three mobile keyboards. The modeling shows that these ten-character codes take between 11 and 25 taps, depending on the keyboards (see Fig. 12 and Table 7).

To prevent guessing, a large set of possible permutations is required for this application. Including lowercase letters would increase the possible set size for a given number of characters. Adding lowercase letters (excluding l, i, o and s) gives a possible set size of $51^{10}$. This is a vastly larger set than just using capital letters and numbers. Indeed, we can maintain a set several orders of magnitude larger if one character is dropped ($51^9 \gg 29^{10}$). Introducing another case means more potential for mode switching, however. Does dropping a character while adding more complexity yield a greater or lesser tap requirement, on average? Introducing the lowercase letter reduced the number of taps required across all keyboards compared to a code containing only uppercase letters and numbers (see Fig. 13 and c.f., Tables 7 and 8).

An alternative strategy would be to just print the win codes entirely in lowercase letters. Omitting potentially confused characters (o,i,l,s), a string of eleven lowercase characters would exceed ($22^{11} > 29^{10}$) the possible set size of mixed numbers and capital letters (again, omitting easily confused glyphs). The expected number of taps to enter eleven lowercase letters is eleven on all keyboards. On average, a win code made up of eleven lowercase characters can be entered with fewer taps than one made up of nine mixed-case letters and numbers (see bottom two rows, Table 8) or one made up of uppercase letters and numbers (see bottom two rows, Table 7).

The findings of the two examples modeled in this section replicate the findings of our analysis of short links: even slightly longer lowercase IDs maintain the size of the set of possible inputs while requiring few taps (and therefore less time) to enter. On mobile devices mode switching is onerous. The design of IDs for entry on mobile devices should account for this and use lowercase-only IDs.

[3] https://tfl.gov.uk/corporate/about-tfl/what-we-do/buses.

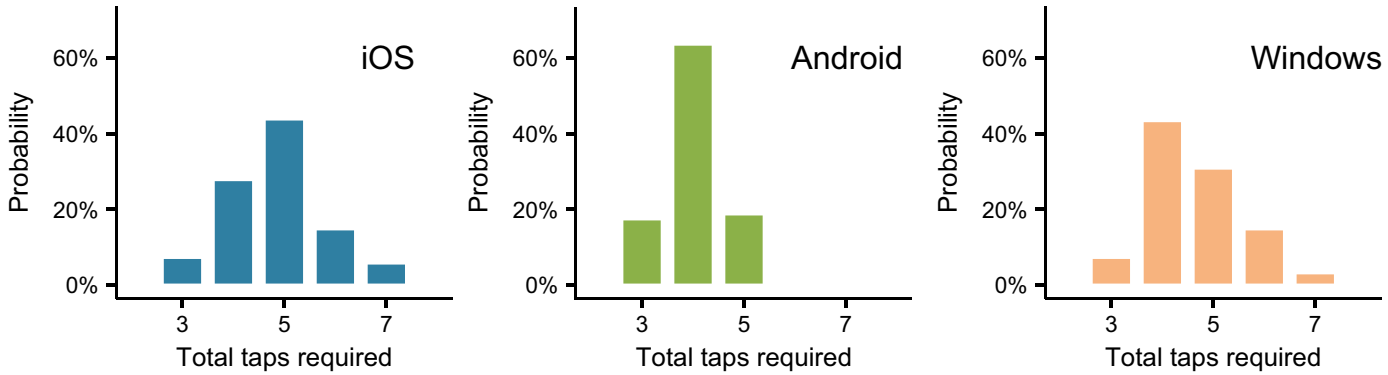


**Fig. 10.** Tap model results for three-character bus-stop identifiers using mixed-case letters and numbers.

**Table 6**
Mean, standard deviation, range and lowercase model comparisons for results of tap-optimizing model. Four-character lowercase stop identifiers require four taps to enter.

| | iOS | Android | Windows |
|---|---|---|---|
| $E(X)$ of taps | 5 | 4 | 5 |
| $\sigma$ of taps | 1 | 1 | 1 |
| Range of taps | 3-7 | 3-5 | 3-7 |
| $P > 4$ taps | 65% | 19% | 49% |
| $P < 4$ taps | 7% | 18% | 7% |

## 7. General discussion

We have developed a systematic, quantified understanding of a specific problem: the design of mobile-friendly unique identifiers. But our results also apply to the design of other text-based services. There has been a trend toward bespoke and adaptive keyboards (e.g., Dunlop and Levine, 2012; Karrenbauer and Oulasvirta, 2014; Leiva et al., 2015; Wiseman et al., 2013). More often than not, though, input devices are a fixed constraint in the design of a service. Most users are typing on the keyboard that came with their phone. Those keyboards have advantages, limitations and quirks. The mode-switching that most touchscreen keyboards require to reach numbers and capital letters is at the root of design improvements we propose in this paper. When designing services, it is vital to be aware of the fixed constraints of a system and to then focus on the aspects of a service's design that can be controlled. Making changes to input data in this way is a cheap, quick and easy way to improve user experience.

Our data suggest that tap-based models are entirely sufficient for making practically useful predictions about text-entry performance on touchscreen keyboards. For exploring the design space for text-entry tasks like ours, a tap-based model is far easier to derive than a time-based model: parameters can be obtained by any individual with a touchscreen phone. This makes them accessible to engineers. Our methods are of particular relevance to finite-state systems like shortening services, confirmation codes or bus-stop signs. Settings where likely values are known a priori, for example in medicine (Wiseman et al., 2013), would also benefit from systematic exploration in the manner we have described in this paper.

### 7.1. Limitations

Participants typed URLs into their preferred browser with the keyboard that they normally used. No software was installed. This meant certain measures were unavailable. For instance, without direct access to our participants' keyboards our study could not record inter-keystroke interval (IKI) data. This paper has focused on comparing two design solutions, so the omission of IKI data does not compromise the conclusions we draw from our results. It does, however, limit the degree to which we are able to explore transcription strategies including chunking, interleaving, checking and correcting. Each of these factors is important in determining how quickly people can type characters strings (Brumby et al., 2007; Janssen et al., 2010; Smith et al., 2008). Furthermore, our timing data may have been affected by factors outside our control, such as network latency. Caution should therefore be exercised when considering the timing data in absolute terms.

**Fig. 11.** Example win codes on bottle caps. Source: http://bit.ly/1WbY7LI.

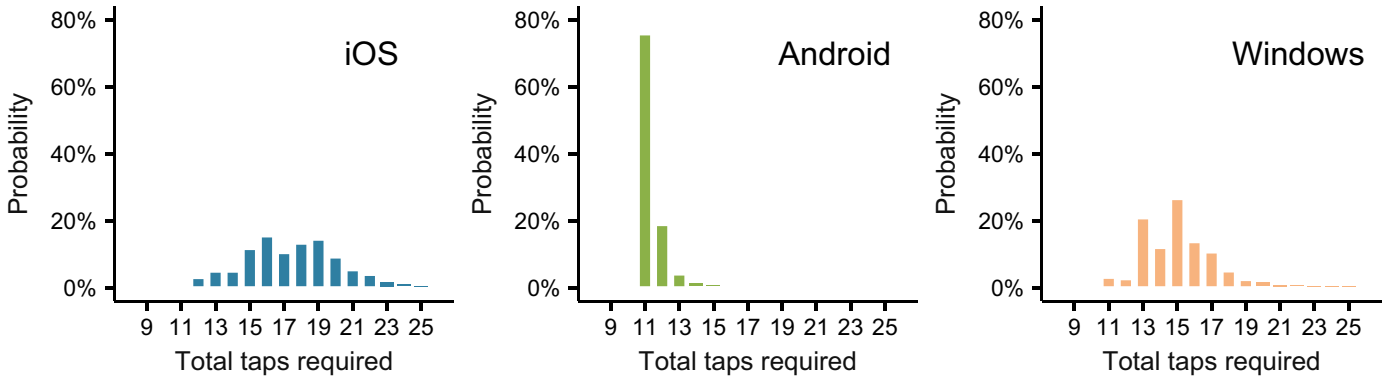


**Fig. 12.** Tap distribution for ten-character 'win codes'. Based on uppercase letters and numbers, excluding potentially confused characters (O,I,L,S,0,1,5).

**Table 7**
Mean, standard deviation, range and lowercase model comparisons for results of tap-optimizing model. Eleven-character lowercase win codes require eleven taps to enter.

| | iOS | Android | Windows |
|---|---|---|---|
| $E(X)$ of taps | 17 | 11 | 15 |
| $\sigma$ of taps | 3 | 1 | 2 |
| Range of taps | 12–25 | 11–15 | 11–25 |
| $P > 11$ taps | 100% | 24% | 97% |
| $P < 11$ taps | 0% | 0% | 0% |

**Table 8**
Mean, standard deviation, range and lowercase model comparisons for results of tap-optimizing model. Eleven-character lowercase win codes require eleven taps to enter.

| | iOS | Android | Windows |
|---|---|---|---|
| $E(X)$ of taps | 14 | 12 | 14 |
| $\sigma$ of taps | 3 | 1 | 2 |
| Range of taps | 9–22 | 9–15 | 9–22 |
| $P > 11$ taps | 87% | 71% | 96% |
| $P < 11$ taps | 8% | 6% | $< 1\%$ |

The sample for our experiment was not broadly representative of the general population. Our sample consisted of technically proficient staff and students at a HCI research laboratory. Although the biases inherent in our sample might limit broad generalizability, it is likely that our particular sample provided the most conservative test of our design recommendations. As proficient users of mobile technology, our participants are likely to have been more experienced with making keyboard mode switches on their devices. As efficient mode switching is a practiced behavior, among less experienced users mixed-case links would likely be even slower to input than our proposed nine-character lowercase design.

Our simulations of visual appearance only considered the size of the representations, not their legibility. How easy link IDs are to discern when printed or projected is an important consideration for our target use case. The fonts we used in our simulations are some of the most commonly used in operating systems and online, so we are confident that the results of our simulations are of practical relevance. Font legibility should be considered alongside chunking in any work looking in detail at the perceptual and cognitive components of link transcription.

We also elected not to explore the wider design space in this work. Other approaches to making links more digestible might prove to be quicker or more meaningful. Some services (e.g., *what3words*) have used strings of words (e.g., http://map.what3words.com/research.hard.work). Links on *what3words* are not meaningful, but these approaches might be adapted to trade set size and brevity for meaning and readability. This design problem offers ample space for exploration; conceptually, computationally and empirically.

Finally, in this paper we have focused solely on short links as a means of advertising resources to users. Other mechanisms, such as QR codes (a form of two dimensional barcodes) and NFC tags (which use radio induction to wirelessly communicate information at close range), also offer ways of advertising resources. Each technology has advantages and limitations. Short links can be entered on any phone with a browser. NFC tags yield information wirelessly without user input but the technology to read the tags is not yet ubiquitous on mobile devices. QR codes can store significant information and can be read by any device with a camera.

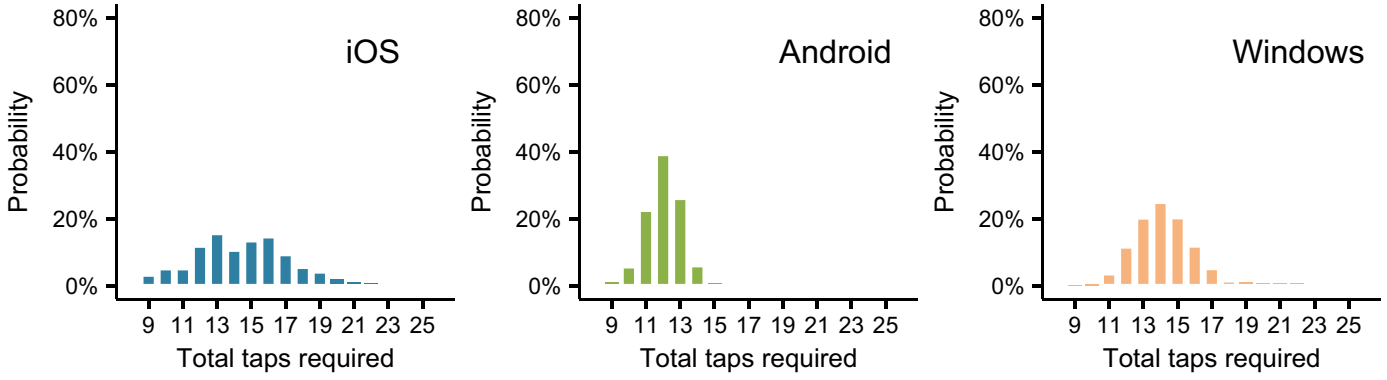


**Fig. 13.** Tap distribution for nine-character 'win codes' if lowercase letters are included along with numbers and capital letters.

However, standalone applications are often required to read QR codes and are limited by the quality of the camera being used to read them (e.g., scanning might be difficult in low light). Each technology has advantages and limitations. A complete empirical assessment of the relative merits of each technology could be made in future work.

## Acknowledgements

This work was supported by the UK Engineering and Physical Sciences Research Council grant EP/L504889/1. We would like thank Keith Vertanen for his efficient handling of this paper. We would like to also thank Mike Byrne and two anonymous reviewers for their insights and helpful suggestions. We would like to acknowledge Frederik Brudy for the generous loan of his *fbr.me* domain. Finally, we would like to thank our participants for agreeing to give up their time to help us.

## References

Banovic, N., Grossman, T., Fitzmaurice, G., 2013. The effect of time-based cost of error in target-directed pointing tasks. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. ACM, New York, NY, USA, pp. 1373–1382. ⟨http://doi.org/10.1145/2470654.2466181⟩.

Barr, D.J., Levy, R., Scheepers, C., Tily, H.J., 2013. Random effects structure for confirmatory hypothesis testing: Keep it maximal. J. Mem. Lang. 68 (3), 255–278. http://dx.doi.org/10.1016/j.jml.2012.11.001.

Berners-Lee, T., Fielding, R. T., Masinter, L., 2005. Uniform Resource Identifier (URI): Generic Syntax. ⟨http://doi.org/10.17487/RFC3986⟩.

Brumby, D. P., Howes, A., Salvucci, D. D., 2007. A cognitive constraint model of dual-task trade-offs in a highly dynamic driving task. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. ACM, New York, NY, USA. pp. 233–242. ⟨http://doi.org/10.1145/1240624.1240664⟩.

Charman, S.C., Howes, A., 2003. The adaptive user: an investigation into the cognitive and task constraints on the generation of new methods. J. Exp. Psychol.: Appl. 9 (4), 236–248. http://dx.doi.org/10.1037/1076-898X.9.4.236.

Cheng, L.-P., Liang, H.-S., Wu, C.-Y., Chen, M. Y., 2013. iGrasp: grasp-based adaptive keyboard for mobile devices. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. ACM. New York, NY, USA. pp. 3037–3046. ⟨http://doi.org/10.1145/2470654.2481422⟩.

Chhabra, S., Aggarwal, A., Benevenuto, F., Kumaraguru, P., 2011. Phi.Sh/$oCiaL: the phishing landscape through short URLs. In: Proceedings of the 8th Annual Collaboration, Electronic Messaging, Anti-Abuse and Spam Conference. ACM. New York, NY, USA. pp. 92–101. ⟨http://doi.org/10.1145/2030376.2030387⟩.

Dayan, P., 2014. Rationalizable Irrationalities of Choice. Top Cognit. Sci. 6 (2), 204–228. http://dx.doi.org/10.1111/tops.12082.

Dunlop, M., Levine, J. 2012. Multidimensional pareto optimization of touchscreen keyboards for speed, familiarity and improved spell checking. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. ACM, New York, NY, USA. pp. 2669–2678. ⟨http://doi.org/10.1145/2207676.2208659⟩.

El Batran, K., Dunlop, M. D., 2014. Enhancing KLM (Keystroke-level Model) to Fit Touch Screen Mobile Devices. In: Proceedings of the 16th International Conference on Human-computer Interaction with Mobile Devices & Services. ACM., New York, NY, USA. pp. 283–286⟨http://doi.org/10.1145/2628363.2628385⟩.

Gallagher, M.A., 2015. Modeling Password Entry on Mobile Devices: Please Check Your Password and Try Again (Thesis). Rice University, Houston, Texas, Retrieved from https://scholarship.rice.edu/handle/1911/87877.

Gallagher, M.A., Byrne, M.D., 2015. Modeling Password Entry on a Mobile Device. In: Proceedings of the International Conference on Cognitive Modeling. pp. 45–50. (Retrieved from) ⟨http://www.iccm2015.org/proceedings/papers/0009/paper0009.pdf⟩.

Gray, W.D., Sims, C.R., Fu, W.-T., Schoelles, M.J., 2006. The Soft Constraints Hypothesis: A Rational Analysis Approach to Resource Allocation for Interactive Behavior. Psychol. Rev. 113 (3), 461–482. http://dx.doi.org/10.1037/0033-295X.113.3.461.

Greene, K.K., Franklin, J., Kelsey, J., 2015. Tap on, Tap Off: Onscreen Keyboards and Mobile Password Entry. Proc. ShmooCon.

Greene, K.K., Gallagher, M.A., Stanton, B.C., Lee, P.Y., 2014. I can't type that! P@$ $w0rd entry on mobile devices. In: Tryfonas, T., Askoxylakis, I. (Eds.), Human Aspects of Information Security, Privacy, and Trust. Springer International Publishing, Cham, Switzerland, pp. 160–171. http://dx.doi.org/10.1007/978-3-319-07620-1_15.

Greene, K.K., Tamborello, F.P., Micheals, R.J., 2013. Computational cognitive modeling of touch and gesture on mobile multitouch devices: applications and challenges for existing theory. In: Kurosu, M. (Ed.), Human-Computer Interaction. Interaction Modalities and Techniques. Springer, Berlin Heidelberg, pp. 449–455, Retrieved from http://link.springer.com/chapter/10.1007/978-3-642-39330-3_47.

Healy, A.F., Kole, J.A., Buck-Gengler, C.J., Bourne, L.E., 2004. Effects of prolonged work on data entry speed and accuracy. J. Exp. Psychol.: Appl. 10 (3), 188–199. http://dx.doi.org/10.1037/1076-898X.10.3.188.

Howes, A., Duggan, G.B., Kalidindi, K., Tseng, Y.-C., Lewis, R.L., 2015. Predicting short-term remembering as boundedly optimal strategy choice. Cognit. Sci. . http://dx.doi.org/10.1111/cogs.12271

Howes, A., Lewis, R.L., Vera, A., 2009. Rational adaptation under task and processing constraints: Implications for testing theories of cognition and action. Psychol. Rev. 116 (4), 717–751. http://dx.doi.org/10.1037/a0017187.

Jakobsson, M., Akavipat, R., 2012. Rethinking passwords to adapt to constrained keyboards. Proc. IEEE MoST.

Janssen, C.P., Brumby, D.P., Garnett, R., 2010. Natural Break Points: Utilizing Motor Cues when Multitasking. Proc. Hum. Fact. Ergon. Soc. Annu. Meet. 54 (4), 482–486. http://dx.doi.org/10.1177/154193121005400444.

Janssen, C.P., Brumby, D.P., Garnett, R., 2012. Natural break points: the influence of priorities and cognitive and motor cues on dual-task interleaving. J. Cognit. Eng. Decis. Mak. 6 (1), 5–29. http://dx.doi.org/10.1177/1555343411432339.

Karrenbauer, A., Oulasvirta, A., 2014. Improvements to keyboard optimization with integer programming. In: Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology. ACM. New York, NY, USA. pp. 621–626. ⟨http://doi.org/10.1145/2642918.2647382⟩.

Kieras, D.E., Meyer, D.E., 2000. The role of cognitive task analysis in the application of predictive models of human performance. In: Schraagen, J.M., Chipman, S.F., Shalin, V.L. (Eds.), Cognitive Task Analysis. Lawrence Erlbaum Associates, Mawah, New Jersey, pp. 237–260.

Klien, F., Strohmaier, M., 2012. Short links under attack: geographical analysis of spam in a URL shortener network. In: Proceedings of the 23rd ACM Conference on Hypertext and Social Media. New York, NY, USA: ACM. pp. 83–88. ⟨http://doi.org/10.1145/2309996.2310010⟩.

Leiva, L.A., Sahami, A., Catala, A., Henze, N., Schmidt, A., 2015. Text Entry on Tiny QWERTY Soft Keyboards. In Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems. New York, NY, USA: ACM. pp. 669–678. ⟨http://doi.org/10.1145/2702123.2702388⟩.

Li, H., Liu, Y., Liu, J., Wang, X., Li, Y., Rau, P.-L. P., 2010. Extended KLM for mobile phone interaction: a user study result. In CHI '10 Extended Abstracts on Human Factors in Computing Systems. ACM. New York, NY, USA. pp. 3517–3522. ⟨http://doi.org/10.1145/1753846.1754011⟩.

MacKenzie, I.S., 2002. KSPC (Keystrokes per Character) as a characteristic of text entry techniques. In: Paternò, F. (Ed.), Human Computer Interaction with Mobile Devices. Springer, Berlin Heidelberg, pp. 195–210. http://dx.doi.org/10.1007/3-540-45756-9_16.

Oulasvirta, A., Reichel, A., Li, W., Zhang, Y., Bachynskyi, M., Vertanen, K., Kristensson, P.O., 2013. Improving two-thumb text entry on touchscreen devices. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems New York, NY, USA: ACM. ⟨http://doi.org/10.1145/2470654.2481383⟩.

Payne, S.J., Howes, A., 2013. Adaptive interaction: a utility maximization approach to understanding human interaction with technology. Synth. Lect. Hum. Cent. Inform. 6 (1), 1–111. http://dx.doi.org/10.2200/S00479ED1V01Y201302HCI016.

Rice, A. D., Lartigue, J.W., 2014. Touch-level Model (TLM): Evolving KLM-GOMS for touchscreen and mobile devices. In: Proceedings of the 2014 ACM Southeast Regional Conference. ACM, New York, NY, USA. pp. 53:1–53:6. ⟨http://doi.org/10.1145/2638404.2638532⟩.

Salthouse, T.A., 1986. Perceptual, cognitive, and motoric aspects of transcription typing. Psychol. Bull. 99 (3), 303–319. http://dx.doi.org/10.1037/0033-2909.99.3.303.

Shin, D.-H., Jung, J., Chang, B.-H., 2012. The psychology behind QR codes: User experience perspective. Comput. Hum. Behav. 28 (4), 1417–1426. http://dx.doi.org/10.1016/j.chb.2012.03.004.

Smith, M.R., Lewis, R.L., Howes, A., Chu, A., Green, C., Vera, A., 2008. More than 8,192 ways to skin a cat: Modeling behavior in multidimensional strategy spaces. In: Proceedings of the 30th annual conference of the Cognitive Science Society. pp. 1441–1446. Retrieved from ⟨https://msu.edu/course/lin/892/rick-stuff/smith-et-al-cogsci-2008-submitted.pdf⟩.

Trommershäuser, J., Gepshtein, S., Maloney, L.T., Landy, M.S., Banks, M.S., 2005. Optimal compensation for changes in task-relevant movement variability. J. Neurosci. 25 (31), 7169–7178. http://dx.doi.org/10.1523/JNEUROSCI.1906-05.2005.

Varcholik, P.D., LaViola Jr., J.J., Hughes, C.E., 2012. Establishing a baseline for text entry for a multi-touch virtual keyboard. Int. J. Hum. Comput. Stud. 70 (10), 657–672. http://dx.doi.org/10.1016/j.ijhcs.2012.05.007.

Vidas, T., Owusu, E., Wang, S., Zeng, C., Cranor, L.F., Christin, N., 2013. QRishing: The susceptibility of smartphone users to QR code phishing attacks. In: Adams, A.A., Brenner, M., Smith, M. (Eds.), Financial Cryptography and Data Security. Springer Berlin Heidelberg, pp. 52–69, Retrieved from ⟨http://link.springer.com/chapter/10.1007/978-3-642-41320-9_4⟩.

Wang, A.I., 2015. The wear out effect of a game-based student response system. Comput. Educ. 82, 217–227. http://dx.doi.org/10.1016/j.compedu.2014.11.004.

Wang, A.I., Øfsdahl, T., Mørch-Storstein, O.K., 2008. An evaluation of a mobile game concept for lectures. In: Proceedings of IEEE 21st Conference on Software Engineering Education and Training. CSEET '08. pp. 197–204. ⟨http://doi.org/10.1109/CSEET.2008.15⟩.

Wiseman, S., Brumby, D.P., Cox, A.L., Hennessy, O., 2013. Tailoring Number Entry Interfaces To The Task of Programming Medical Infusion Pumps. Proc. Hum.

Fact. Ergon. Soc. Annu. Meet. 57 (1), 683–687. http://dx.doi.org/10.1177/1541931213571148.

Wiseman, S., Cox, A.L., Brumby, D.P., 2013. Designing Devices With the Task in Mind: Which Numbers Are Really Used in Hospitals? Human Factors. J. Hum. Fact. Ergon. Soc. 55 (1), 61–74. http://dx.doi.org/10.1177/0018720812471988.

Wiseman, S., Soto Miño, G., Cox, A.L., Gould, S.J.J., Moore, J., Needham, C., 2016. Use your words: designing one-time pairing codes to improve user experience. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. ⟨http://doi.org/10.1145/2858036.2858377⟩.

Wobbrock, J.O., Cutrell, E., Harada, S., MacKenzie, I.S., 2008. An error model for pointing based on Fitts' law. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. New York, NY, USA: ACM. pp. 1613–1622. ⟨http://doi.org/10.1145/1357054.1357306⟩.

Zhang, Y., Hornof, A.J., 2014. Understanding multitasking through parallelized strategy exploration and individualized cognitive modeling. In: Proceedings of the 32nd Annual ACM Conference on Human Factors in Computing Systems. ACM, New York, NY, USA. pp. 3885–3894. ⟨http://doi.org/10.1145/2556288.2557351⟩.