Effects of Language Modeling and its Personalization on Touchscreen Typing Performance

Andrew Fowler*
Oregon Health & Science University
Portland, OR
fowlera@ohsu.edu

Kurt Partridge, Ciprian Chelba, Xiaojun Bi, Tom Ouyang, and Shumin Zhai
Google, Inc.
Mountain View, CA
{kep,ciprianchelba,bxj,ouyang,zhai}@google.com

ABSTRACT
Modern smartphones correct typing errors and learn user-specific words (such as proper names). Both techniques are useful, yet little has been published about their technical specifics and concrete benefits.

One reason is that typing *accuracy* is difficult to measure empirically on a large scale. We describe a closed-loop, smart touch keyboard (STK) evaluation system that we have implemented to solve this problem. It includes a principled typing simulator for generating human-like noisy touch input, a simple-yet-effective decoder for reconstructing typed words from such spatial data, a large web-scale background language model (LM), and a method for incorporating LM personalization. Using the Enron email corpus as a personalization test set, we show for the first time at this scale that a combined spatial/language model reduces word error rate from a pre-model baseline of 38.4% down to 5.7%, and that LM personalization can improve this further to 4.6%.

Author Keywords
Mobile text entry; keyboard error correction; language modeling

ACM Classification Keywords
H.5.2. User Interfaces: Input devices and strategies

INTRODUCTION
Text input methods have often been key features in shaping the paradigm shifts in mobile computing in the last decades. For example, handwriting with a modified character set (Graffiti) was central to the Palm Pilot era of PDAs. Miniature thumb keyboards dominated the design of BlackBerry and similar devices. Smart touch keyboards (STK) enabled the current generation of full touchscreen mobile devices.

STK products rely on language models, or least a dictionary, to correct touch errors, autocomplete partial letter strings to complete words, and predict what the user will type next. Surprisingly (and for methodological reasons discussed later) the quantitative effects of applying language modeling to touch-based STKs have never been formally reported in the research literature. In fact many recent studies of touch screen typing do not involve LMs at all (e.g. [10]). Our concern is that in many cases, evaluation without language-model based correction may lead to incorrect conclusions about the benefits of a particular technique. For example, [34] found that improvements in spatial error modeling had little effect in practice since LM-based correction compensated for such spatial errors. For this reason, quantitatively benchmarking the effects of a standard LM to STK is one of the goals of this paper.

Conversely, traditional metrics of language models in isolation may not accurately reflect how they perform in a deployed decoder. A model-intrinsic measure like per-word perplexity, though commonly used to compare LMs, can correlate poorly with downstream metrics, as is the case with word error rate in speech recognition [6]. *Extrinsic* evaluations like typing accuracy, which more closely reflect the experience of an actual user, can only be evaluated if the user's role in the decoding process is considered.

An especially relevant language modeling technique for mobile text entry, and one that is also difficult to measure, is *LM personalization*, in which a language model's lexicon and probabilities are made flexible and adaptive to the user. A natural way to accomplish this is to utilize a user's written history, since it best reflects that person's usage of language. However, scientific understanding of text input LM personalization is limited. The most detailed work dates from 2006, and studies previous-generation mobile devices with physical keyboards [30]. Modern soft keyboards, with more fuzzy input on touchscreens, may rely more heavily on LMs, and therefore are more likely to benefit when personalization works, or suffer when it does not.

In this paper, we contribute the first large-scale longitudinal simulation study of the effects of language modeling, with and without personalization, on finger-touch-based soft keyboards. As a methodological contribution, we created a novel integrated framework (dubbed Sketch Jr.) for testing input *accuracy*, which differs from intrinsic measures such as LM perplexity and keystroke savings, which are typically computed in isolation of the input problem. Our method allows for *extrinsic evaluation* of our models by incorporating large-scale simulation of human-like inputs. We evaluate our LMs in connection with sloppy user input based on empirical data, using a principled error correction and word completion model (decoder). We use the longitudinal real-world text-writing history of a large group of individuals (the Enron Corpus). Through this human simulation, we show that a background LM can reduce typing word error rate (WER)

*This work was done while Andrew Fowler was an intern at Google Inc.

CHI 2015, April 18–23, 2015, Seoul, Republic of Korea
ACM 978-1-4503-3145-6/15/04
http://dx.doi.org/10.1145/2702123.2702503

from 38.4% to 5.7%, and that personalized LMs further reduce this error to 4.6% (a 19.3% relative change compared to the non-adapted model). We also examine personalization methods based on the idea of an exponentially-decaying cache, and find that a decay time in months is optimal, but that the decay curve has little effect on the overall performance result. This work is also, as far as we know, the first to study and quantify the tradeoff between the overall error rate and *false corrections*, a subclass of errors highly visible to end-users.

RELATED WORK

Language Models for Text Entry

In the broad sense, language models have long been used in text input. The simplest LM is a lexicon, i.e. a list of permissible words. A lexicon can already provide quite powerful constraints in, for example, decoding ambiguous 12-button-based dialpad input found on traditional phones, which dates back to the early 1970's [28]. Since entering logographic text such as Chinese with a Roman alphabet is highly ambiguous, applying some form of model that maps Roman letter-based phonetic input to logographic characters has also been necessary and common in East Asian countries for a long time [30]. Innovations such as word-gesture typing, or shape writing [36, 37], also leverage language regularities in the form of a lexicon that could further be adapted to the individual user by extracting a vocabulary from the user's past documents [22], or by moving passive words into an in-use active vocabulary [23]. Goodman and colleagues [11] drew inspiration from speech recognition for stylus keyboard tapping and applied a character n-gram model to correct stylus taps[1], achieving a reduction in *character-level* error rate from 2.40% to 1.39%. Klarlund and Riley [17] applied N-gram language modeling to a cluster keyboard (i.e. a keyboard with reduced number of keys), and showed large improvements over unigram frequency models. Tanaka-Ishii [30] presented a very comprehensive and systematic study of four types of LMs, namely unigram, co-occurrence, MTF (move to front, which gives higher priority to recent words), and PPM (prediction by partial match), and studied their use in adaptation for text input. These were applied in English with reduced keyboards, in Japanese with kana, and in Thai also with ambiguity but not errors. It was found that PPM outperformed other techniques in ranking prediction candidates. The PPM model used was an interpolation between a unigram model trained on a 30M Wall Street Journal news corpus (for English) and a n-gram model trained on a small user corpus. The average rank improvement with adaptation depended on the user corpus size. For example when the size of the user corpus increased from 0 to 50,000 words, the PPM prediction average rank of the intended word improved from 1.3 to 1.09.

Language Model Adaptation

The essential problem that LM adaptation seeks to solve is the problem of *domain mismatch*. LMs can be trained on enormous text corpora, but those corpora rarely match the domain of the target text one wishes to model. In-domain text is usually small in quantity and not enough to build a robust model. LM adaptation algorithms attempt to combine the benefits of a large, out-of-domain model with a small in-domain model. *Personalization* can be seen as the ultimate adaptation problem, since one is adapting to a single individual (for which in-domain text is almost certainly limited). LM adaptation has been studied extensively. An excellent review of various algorithms can be found in [2] and [9].

Cache-based LM adaptation keeps a continuously-updated cache of words, and use this cache to build the adaptation/personalization model. In the context of model adaptation, the model generated from the cache is the in-domain model. A cache varies slightly from other adaptation techniques in two ways: First, the adaptation model must be updated on the fly because new words are typed continuously (the technique is commonly called dynamic cache-based adaptation to reflect this fact). Second, the user cache may begin in a completely empty state, or be primed with a small amount of in-domain text (in our case, we primed with 30 days of user emails). Cache-based LM adaptation was used by [24] to improve the performance of a speech recognition system.

Cache-based LMs are further explored in [8], which describes a method in which the relative weights of words in the cache are made to decay exponentially, with "older" words having lower weights. This reflects the common-sense notion of *recency*, which suggests that, in general, more recently-used words in a text will re-occur with higher probability than words used in the more distant past. The results of [8] indicate that using an exponentially-decaying cache results in improvements in perplexity, but as shown later, our extrinsic evaluation of text entry appears to indicate that these gains are negligible in our application.

RESEARCH METHODOLOGY

Two crucial questions in our experimental design were whether to primarily rely on modeling and simulation or human subjects directly, and whether to collect text from users or use an existing corpus. We decided to simulate using a large real world text corpus (Enron) for all experiments, for the following reasons:

- **Practicality and Scale:** Scale is important to language modeling evaluation, particularly with personalization. Variances and nuances in language use can only be revealed with longitudinal data. Further, as we later confirmed, large user-to-user variance is possible and many users are needed to give meaningful results. Collecting this amount of data from individual users is possible, but impractical. In addition, we required a large development set in order to empirically optimize various system parameters. Large-scale parameter optimization of this kind without simulation is infeasible.
- **Privacy and Reproducibility:** Another problem with collecting native text from individuals is that they would have a reasonable expectation of privacy. The Enron Corpus largely avoids this issue, and its widespread availability allows for experiments that are vastly more reproducible.
- **Availability of Foundational HCI Research:** Recent HCI research has measured crucial parameters of human motor control for text entry tasks. It has been shown that tapping on key targets may be modeled by a Gaussian distribution

[1]Since Goodman et al. involved *small-scale* observations with a *stylus* input using *character-level* evaluations/LM, it differs significantly from our work, in particular because their baseline tapping error rate is one fifth of ours.

model whose parameters depend on hand posture (finger, thumb, or two thumbs), the portion of the keyboard (left, right, top, or bottom), and the individual user [10, 1, 3].

We do not wish to discount the advantages of human tests, which would provide valuable information not available to simulation, such as how users respond to personalized text input and what strategies are used to correct errors. Human tests would also validate our simulation approach. We reserve these questions as important avenues for future work; for the present work, however, we assert that combining human input models established from users' natural typing data and longitudinal real world text writing records captures the essence of smart touch keyboard evaluation and reveals the effects of language modeling at a large scale.

DATA

Background Language Model

Our background model was a Katz-smoothed [16] bigram LM trained on 114 billion words scraped from the publicly-accessible web in English. We pruned the model size using entropy pruning [29], a technique for decreasing the size of a backoff LM. The basic method involves selecting n-grams for removal from the model by measuring the relative entropy of models with and without that n-gram. A threshold is set for excluding n-grams by this criterion. The choice of Katz over the more popular Kneser-Ney [19] for smoothing was dictated by the small memory footprint available for the LM on a mobile device, and the reduced difference between the two approaches after entropy pruning [5]. The final pruned model contained about 8.5 million n-grams (8.4M bigrams and 168K unigrams).

Enron Corpus as an Evaluation Set

The Enron Corpus [18] is a large set of emails that were collected by the Federal Regulatory Commission when the Enron Corporation was under investigation in late 2001. This corpus was made public in 2003, and consists of over 600,000 individual email messages from multiple years (1998 to 2002). It is one of the largest collections of email publicly available, and is a valuable and widely-used resource for natural language research.

In the field of text entry, Enron data has been used for the creation and comparison of phrase sets to be used in other text entry experiments [20, 21, 31] . In [26], Enron data was compared to data from Facebook and Twitter in an information-theoretic manner, with the goal of judging how "representative" various phrases are. Enron has also been used for domain adaptation in tasks other than language modeling [7, 27, 35].

The Enron Corpus is well suited for text entry research for three reasons: First, it consists of human communication, making it a better match to text input applications. Second, it consists of a relatively large body of real-world usage of text communication (as opposed to news corpora [4] or fiction writing [32]). Third, the corpus preserves a long history of communication by user with minimal privacy concerns, allowing for more in-depth research into personalization techniques.

However we found the Enron corpus to be inadequate in its raw form. Our preprocessing of the corpus was extensive, and included deduplication, signature text removal, name canonicalization, and attached text removal. Because our goal was LM personalization, the driving idea of these preprocessing steps was a desire to have clean text for each user that was strongly linked to that user, i.e. not generated automatically or typed by somebody else. This is particularly important when modeling for text entry, since URLs and attached messages are very unlikely to have been typed by the person doing the text entry. We found that extracting clean, human-generated text from the Enron Corpus is surprisingly nontrivial, so we have made the details of our preprocessing available as an auxiliary document to this paper.

Development and Test Sets

The next step was to divide the Enron emails into test sets. We selected 90 Enron users, each with more than 1500 total words in their collected sent emails. We defined a minimum word count per user for two reasons: First, we wanted a large enough volume of text to perform adaptation experiments that involved learning user words and adapting a model to them. Second, we wanted to avoid sparsity issues. We divided these 90 users into development and test sets of 45 users apiece, with a similar distribution of per-user word counts in each set. The development set contained 38,114 messages and 1,355,266 words, and the test set contained 31,740 messages and 1,214,403 words. Since our models are based on caching, there is no Enron training set per se; user models are trained dynamically as one proceeds through each user's portion of the test set.

SIMULATION

The basic workflow of our system is to generate simulated keyboard taps for a given text, use a decoder to reconstruct the text from those noisy taps in the presence of various LMs, then evaluate the output.

Simulated Smart Touch Keyboard Typing

We have discussed the benefits of human simulation as a general research method in LM personalization. Here we provide critical details needed to making such simulation meaningful to text input. We also point out aspects of typing not simulated in the current work.

In order to study the impact of LM personalization in STK applications, we simulated each Enron user typing his or her test set in chronological order using a simple STK decoder, to be described shortly. We used the key layout from the specifications of the Nexus 5 QWERTY keyboard, which has a single-key width of 6.16 mm and a key height of 9.42 mm. We assumed that the user was perfect in *intent*; in other words, the user's target word is always the actual target word[2]. Additionally, if the decoder fails to produce the target word, it is simply counted towards the error rate but we do not simulate any attempt on the part of the user to correct such errors with backspace or other methods.

To evaluate the power of LMs and personalized LMs in error correction, we introduced human errors in the simulated input process that reflected the main source of error in touch

[2]This does not exclude the possibility of spelling errors in the Enron Corpus; in this case, we simply assume that the typo is the correct target, and judge it accordingly.

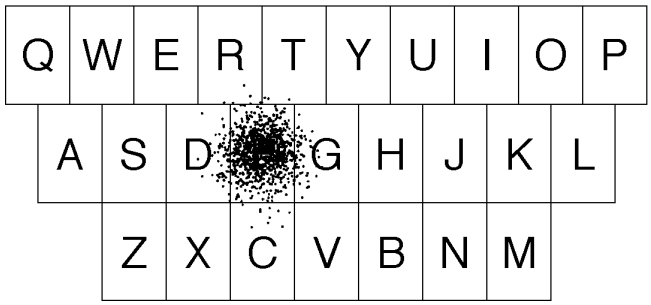

Figure 1. Our keyboard layout with 1000 randomly sampled taps on the "F" key

screen keyboards. We incorporated spatial noise in the tapping/typing signal by sampling randomly from a 2D Gaussian distribution. The mean of the distribution for each tap was the center of the intended key, and the standard deviations were set to be 1.97 mm in the x direction and 1.88 mm in the y direction, as illustrated in Figure 1. These values were based on [1], which determined the 2D Gaussian distribution to be representative of actual human tapping, and described typical variances. Taps landing outside of the border of the intended key and inside another key were given the identity of the new key in the literal string output (a typing error). Taps outside the border of *every* key were discarded and resampled. We found that these settings result in a per-tap error rate of 12.8%, which translates to 38.4% of words in the test set containing at least one key error. In practice, the actual amount of spatial errors follows a speed-accuracy trade off model. The faster one types, the more errors one would make. See "FFitts law" [3] for an in depth exploration of the speed-accuracy model of touch screen target acquisition containing both relative and absolute precision errors.

We deliberately chose the 2D Gaussian for modeling spatial touch, for multiple reasons: First, we desired a simple, principled model (in the spirit of "Occam's razor") that still captured the spatial variance needed to test LM contribution to decoding noisy touchscreen input. Second, 2D Gaussian is at the core of spatial models used in commercial level keyboards such as the Android Open Source Project (AOSP) keyboard. Third, while more complex models of touch have been explored in the literature (Holz et al. [13], Azenkot & Zhai [1], and others highlight additional phenomena involved with finger touches, in particular spatial offsets that depend on finger position, visual cues, and hand posture), it is quite hard to put these models in practice because of issues such as sensor requirements. Fourth, Yin et al. [34] showed that if a more complex hierarchical backoff model (with hand posture and key location dependencies) is used, the gain is large when considering individual letters only but small (diminishes) when connected with an LM. We believe further gains can still be made in the future with improved spatial modeling, but a simple Gaussian model reflects the state of the art and is sufficient in testing LM contribution.

We made three further simplifying assumptions about the test data and the simulated user. First, since the simulated STK contains only the symbols a–z, the typing simulation only considers those letters. We stripped out other non-space symbols before testing. Second, the simulated user always taps the space bar correctly, i.e. always taps it when intended and never taps it when not intended. Third, the input process is case-insensitive.

We chose to focus on 2D spatial noise exclusively as a source of error in the tapping signal. There are other potential error sources, many of which fall under the category of *cognitive error*. An example of a cognitive error would be the user misspelling a word because they do not know the correct spelling. In such a scenario, an incorrect string is created, but it is not due to an error of typing; the user intended the string, whether or not it was correct. For instance, typing `forword` instead of `forward` would constitute a cognitive error, while `forqard` would likely be a spatial error. Other cognitive errors include selecting the wrong word from the suggestion list, or failing to tap the correct word when it appears in the selection list. Because modeling cognitive error is subtle and because we wish to be able to interpret our results more straightforwardly, we exclude it from our experiments. Similarly, we exclude errors involving inserted and deleted characters. Overall, our goal was to introduce a predominant type of error in sufficient and representative quantity to test the LM personalization effect in correcting that error type. Other types of errors may follow the same pattern, but should be further researched in the future.

A Simple Model Decoder

Once our simulator had generated a sequence of taps, we next applied a simple model decoder to represent the core function of a STK. The general idea of combining a spatial model of touchscreen behavior with language modeling for text entry has previously appeared in the literature. For example, Weir et al. [33] applied user-customized spatial models and used touch pressure as a signal to increase the spatial model weight relative their LM weight. Our work differs from Weir et al. in our use of longitudinal user data, the large scale of our simulations, and the fact that we personalized our LMs rather than our spatial model. For the purpose of systematically evaluating the impact of LM adaptation in mobile text input applications, our decoder was not designed to be optimal, complete, or computationally efficient. Instead it was meant to be as simple as possible but still able to take advantage of a language model and effectively correct imprecise spatial input.

Suppose we have a tap sequence $T = t_1, t_2, ..., t_n$, where the t_i values are the coordinates of each tap, and a potential word w=$l_1, l_2, ..., l_m$, where the l_i values are the letters that make up that word. Note that any predicted word must have at least n letters, so we require that $m \geq n$. We define the spatial score $S(w|T)$ as

$$S(w|T) = \sum_{i=1}^{n}((t_{ix} - l_{ix})^2 + (t_{iy} - l_{iy})^2), \qquad (1)$$

where t_{ix} and t_{iy} are the x and y coordinates of the tap t_i, and l_{ix} and l_{iy} represent the x and y coordinates of the key center of the letter l_i.

We further let $L_{LM}(w|c)$ be the negative logarithm of the LM probability of the word w given context c. The combined score $B(w|c, T)$ is equal to

$$B(w|c, T) = L_{LM}(w|c) - \gamma S(w|T), \qquad (2)$$

where γ is the spatial model weight. The best-scoring word w_{best} under the combined model is therefore

$$w_{best} = \arg\max_{w} B(w|c,T). \quad (3)$$

We found through experiments on our development set that the optimal spatial model weight γ for our task is 0.00022[3].

Algorithm 1 Procedure for generating possible words matching a tap sequence

```
1:  procedure GetPotentialWords(T, d)
2:      input: list T of n taps; distance threshold d
3:      wordlist ← []
4:      for t_i in T do
5:          generate set S_i of all letters on keyboard with centers
            within distance d of t_i
6:      for w_p in lexicon L with ≥n letters do
7:          for letter l_i in w_p, 1 ≤ i ≤ n do
8:              if l_i ∈ S_i then
9:                  continue
10:             else
11:                 break (word contains too-distant letter)
12:         append w_p to wordlist
13:     return wordlist
```

An important parameter in our spatial decoding technique is a pruning criterion, used to decrease the size of the search space by limiting potential keys for each tap. Any key with a center within a specific radius from the tap may be a potential letter. Given the geometry of our simulated keyboard, this allows for between one and five potential keys for each tap. The search algorithm then recursively traverses the tree of possible words and word prefixes given these potential keys, and filters out all prefixes that do not match a known word in our LM lexicon. This is demonstrated in the procedure GetPotentialWords in Algorithm 1. We found using the development set that the pruning radius of 7.99 mm resulted in fast simulation without degrading model performance.

Figure 2 and Table 1 illustrate a simple example of the combined decoder at work. The spatial decoder has no notion of word probabilities in language therefore alone can only function as a naive keyboard. It favors the spatially-closest words. The LM has no spatial data (other than what set of keys is nearby each tap), so it simply gives the most probable words possible given the pruning radius. The combined model synthesizes these two sources of information to generate a better word list. Note that all three models, *even the spatial model*, are constrained by the lexicon; the word '`bs`' is absent from Table 1 because it is not in the lexicon, despite the fact that it is the typed literal string. The top word in the spatial model, '`ba`,' is the word in the lexicon (albeit rare) with the highest spatial score. (As mentioned elsewhere, our decoding procedure has a method for including the actually-typed sequence in the suggested words list, but this is a separate process.)

[3]The small value of γ is a consequence of our spatial model outputting much larger absolute values than our LM, ultimately because the spatial model uses pixel widths as a distance unit.

Algorithm 2 Procedures for typing simulation and evaluation

```
1:  procedure CalculateAccuracy(w, n, c, d)
2:      input: target word w of length n, previous context c
3:      input: pruning threshold d, autocorrect threshold φ
4:      T ← []
5:      for letter l_i in w do
6:          t_i ← sample from 2D Gaussian centered at l_i
7:          add t_i to T
8:      wordlist ← GetPotentialWords(T, d)
9:      lit ← string of actual keys tapped by taps in T
10:     if wordlist is not empty then
11:         for w_p in wordlist do
12:             calculate score B(w_p|c, T) given T and c
13:         if highest score > φ then
14:             w_best ← w_p with highest model score
15:         else (best word not good enough; use lit. string)
16:             w_best ← lit
17:     else (no suggestions; use literal string)
18:         w_best ← lit
19:     if w_best = w then
20:         return success
21:     else
22:         return failure
23: procedure CalculateEfficiency(w, n, c)
24:     input: target word w of length n, previous context c
25:     T ← []
26:     for i in 0 ... n do (iterate over number of taps)
27:         if i > 0 then
28:             l_i ← i^th letter of w
29:             t_i ← key center of l_i
30:             add t_i to T
31:         wordlist ← GetPotentialWords(T, 0)
32:         lit ← string of actual keys tapped by taps in T
33:         if wordlist is not empty then
34:             for w_p in wordlist do
35:                 calculate score B(w_p|c, T) given T and c
36:             w_bestset ← three w_p with highest model score
37:         else (no suggestions; use literal string)
38:             w_bestset ← lit
39:         if lit ∉ w_bestset then
40:             if w_bestset contains less than three words then
41:                 append lit to end of w_bestset
42:             else
43:                 replace 3^rd word in w_bestset with lit
44:         if w ∈ w_bestset then
45:             return i + 1 (count of taps needed to type w)
```

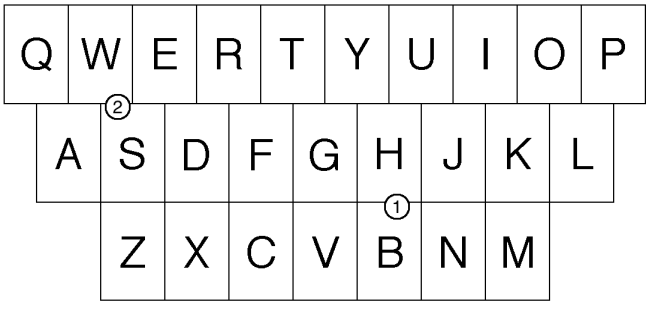

Figure 2. Two example taps on the simulated keyboard.

	Decoding Method		
Rank	Spatial Model	Language Model	Combined
1	ba	be	be
2	ha	he	best
3	be	had	bad
4	he	best	he
5	bad	bad	had

Table 1. Top potential words given the taps in Figure 2, under the spatial model only, language model only, and combined decoder.

EVALUATION METRICS

Language models are often evaluated using the *per-word perplexity* metric across a test set [15]. It is an information-theoretic measure of how well a statistical model predicts a sample. In theory, a language model better suited to the test data will result in a lower perplexity value. Many LM adaptation techniques are evaluated in this way, and report a decrease in observed perplexity as evidence of an improved model. In practice, however, even significant improvements in perplexity do not necessarily correlate with commensurate improvements in extrinsic objectives, particularly in speech recognition research, where WER is the dominant evaluation metric [6, 12]. Further, perplexity is often difficult to compare across tests, because it is lexicon-dependent, i.e. the models involved must use the same word list. This is not easy to accomplish in an adaptation/personalization setting, where adding new words to a model is of central importance.

One response to the shortcomings of perplexity and other LM-intrinsic metrics is to use LM-extrinsic evaluation, a research philosophy outlined in [14]. In our case, the extrinsic evaluation metrics are keystroke savings and WER. Keystroke savings (see Equation 4) is the de-facto evaluation standard for text entry efficiency measurement [25]. WER is a standard way of measuring the accuracy of a word-based decoding task, and is widely used in speech recognition.

Importantly, an LM's extrinsic power in error correction (i.e. word-level accuracy) can *only* be measured in the presence of noise and a complete closed-loop system that includes a decoder and erroneous input. The development of such a system, and the ability it affords to measure error correction, are key contributions of our research. In ASR it was only when a reference recognizer was applied that language modeling studies could move from intrinsic measures such as perplexity to extrinsic measures such as WER [6]. Since "fat" finger imprecision is the main source of noise in finger-operated touch keyboard [1], the simple model decoder as described earlier together with the spatial noise model and the real world personal history data set of Enron emails enabled us to measure the extrinsic power of the LM and its adaptation.

Simulating Word Correction

When simulating word *correction*, we proceed one word at a time. The simulator samples one tap (with spatial noise) for *every letter* in the target word, then returns the single most probable word from the decoder. If this word has a probability above a certain *autocorrect threshold*, the word is typed, followed by a space. If not, the uncorrected *literal string* is typed (the literal string is the sequence of visual keys tapped; in the case of Figure 2, the literal string is 'bs'). A threshold value 0.7 was used for all of our accuracy experiments, for reasons explained in the Results section below. This is demonstrated in the procedure CALCULATEACCURACY in Algorithm 2. The evaluation metric for word correction is word error rate. WER is calculated by dividing the number of incorrectly-decoded words by the total number of words.

Simulating Word Prediction

When simulating word *prediction*, we also proceed one word at a time, but there are two important differences. First, we provide the simulated user with a list of three word suggestions after each tap, potentially saving keystrokes. Second, in order to isolate the prediction effect, we exclude spatial noise.

The following is demonstrated in the procedure CALCULATEEFFICIENCY in Algorithm 2. For each word, we incrementally simulate perfect input for each letter, starting with zero taps. At each point, the LM (no spatial model or decoding) generates the three most probable words given the letters so far (including the case where there are not yet letters typed). If the literal string is not among these three, the third-most-probable word is removed and replaced with the literal string. Otherwise, the list remains unchanged. We assume that the simulated user will notice immediately if the target word is among these top three. If so, we count one keystroke for selecting the word. If not, we make another simulated tap and generate a new suggestion list. (Note that, in word-initial position, no taps have occurred, and all words in the LM are possible.) In the case where all n taps are needed to type an n-letter word, the simulated user makes one last check for the target word in the top-three word prediction list. If the target is there, it is selected. If not, a space key is typed. When the space key is tapped, the single most probable word in the prediction list is entered. This is the reason the literal string is always included in the list. Without the literal string the user has no way of entering words that are not in the LM lexicon.

The evaluation metric for word prediction is keystroke savings, as defined in Equation 4. Because we assume no spatial noise when measuring word prediction, accuracy is always 100%.

$$ks = 1 - \frac{total\ taps\ required}{total\ characters\ in\ test\ set} \quad (4)$$

Note that, when simulating either correction or prediction, each word requires a minimum of one keystroke, since either space or select-suggestion must be tapped to move to the next word.

Since our LM is a bigram model, its probabilities are conditioned on the previous word in each context. At the beginning of each line (every line in the test set is a sentence), the model is primed with the start-sentence symbol <S>. After each word is entered, that word becomes the previous word, *even if the typed word is incorrect and does not match the true string*. The latter scenario matches real-world use, since the typed word is all any typing system has access to, even if it is not correct. In the case of words not in the LM lexicon, the previous word is set to the unknown symbol <UNK>.

LANGUAGE MODEL PERSONALIZATION

We implemented two basic types of cache-based language models: Uniform and exponentially-decaying. The uniform

cache considered a sliding window of words stretching back as far as available in the user's test data, and consisted of an unsmoothed unigram LM from the words in that window. It was continuously updated with each individual user's writing history as simulation progressed. We used linear interpolation to combine the background model with this in-domain cache model[4]. Linear interpolation is a simple technique for LM adaptation. Equation 5 (adapted from [2]) illustrates the basic interpolation method, where $P(w|c)$ is the conditional probability of word w given context c, P_A is the adaptation model, P_B is the background model, and λ represents the relative weight of the background model.

$$P(w|c) = (1 - \lambda)P_A(w|c) + \lambda P_B(w|c) \quad (5)$$

We found through experiments on held-out data that the optimal window size is infinite, i.e. all words in the user's history should be kept in the cache. Further, we found the optimal λ value to be 0.8.

The exponentially-decaying cache was derived from [8], which first described the technique. It used an unsmoothed unigram model like the uniform cache, but instead of counting words directly we modified the weights according to how far back they were in the cache. This captured the recency effect. Equation 6 (adapted from [8]) illustrates the basic method, where P_{cache} is the conditional probability of word w_i given the cache w_1 through w_{i-1}, and I is a binary function such that $I(A)$ = 1 if A is true, and 0 otherwise. The decay rate α describes how quickly word weights decay in the cache, and β is a normalizing constant:

$$P_{cache}(w_i|w_1, w_2, ..., w_{i-1}) = \beta \sum_{j=1}^{i-1} I(w_i = w_j)e^{-\alpha(i-j)} \quad (6)$$

We used a grid search on held-out data to find that the optimal decay rate α is 0.0003, which is a slow rate of decay but consistent with the values determined in [8]. An α value of 0.0003 means that it takes roughly 2300 words of history before the adjusted weight decreases to half of its initial value. Since most users in the test set generate between 1000 and 2000 words per month, this decay rate can be said to take months to take effect. Note, however, that this formulation is based on word positions rather than elapsed time. A word located at position $i - 100$ may have been typed at any point in the past, depending on the user. As with the uniform cache method, we combined the cache model with the background model using linear interpolation, as in Equation 5. We again found the optimal λ value to be 0.8.

For both cache methods, we primed the cache with the first 30 days of email from each of the 45 individuals in the test set. Messages sent during this period were used only to fill the personalization cache, and not simulated for evaluation purposes. The choice of 30 days was based on an empirical study on the development set, which showed that 30 days generally provides a reasonable initial cache size. The mean size of this priming cache in the test set was 1429 words. Both cache models were updated from the 31st day onward till the end of each individual's text available in Enron.

[4]Note that since a LM requires large quantities of text, using the cache model alone would not be effective; even users with large text histories do not have enough text to build a useful LM without sparsity issues.

Model	WER %	OOV %
No Language Model	38.4	n/a
Background LM Only	5.7	1.6
Uniform Cache Adaptation	4.6	0.9
Decaying Cache Adaptation	4.6	0.9

Table 2. Word error rates for various model setups, averaged across all simulated Enron users.

We define the model lexicon at any given time to be all words in *either* the background model unigram list or the cache. When a word was absent from one of these two models, it received a probability of zero in that model before the interpolation defined by Equation 5 took place. Though the background model did contain an <UNK> symbol, we normalized it out when calculating word probabilities[5]. We also make no attempt to estimate probabilities of words not in either model, i.e. entirely absent from the model lexicon (though we do preserve the literal string, which can sometimes be absent from both models).

RESULTS

By replicating and simulating users' natural finger touch input behavior, multi-user and longitudinal writing records, and the primary functions keyboard decoding, we are able to reveal what simple human subjects lab experiments could not do—quantitatively benchmarking the effects of language modeling with and without personalization at scale.

Word Correction

Table 2 shows WER results for the accuracy experiments. The first row represents the value of a naive keyboard without any language modeling and takes the literal string only, which is equivalent to decoding touch input according to its closest key centers. The word error rate of such a naive keyboard is 38.4%.

With a background LM combined with the simple decoder, both as described earlier, the error rate is drastically reduced to 5.7%. This is at the 0.7 autocorrect threshold operating point, as explained below. LM personalization further improves the error rate to 4.6%, a relative decrease of 19.3%. The exponentially-decaying cache method did not outperform the uniform cache method.

The OOV rate is calculated on a per-word basis as the human simulation progresses through the test set. In the case of cache-based models, a given word is considered OOV if it did not exist in the background model or the cache (though since all OOVs are added to the cache immediately, no word can be an OOV twice for the same user). The cache is emptied before moving to a new user in the test set.

False Corrections and the Autocorrect Threshold

In our word correction experiments, we considered one type of failure to be worse than the others. If the literal string is correct, but the autocorrect mechanism changes it to a word that is incorrect, we call this a *false correction*. The rationale for setting apart these failures is this: Suppose a user carefully types a word such that all taps in that word fall inside their respective target key boundaries. The user likely has a strong

[5]We still use <UNK> for n-gram contexts; see earlier paragraph on LM conditioning.

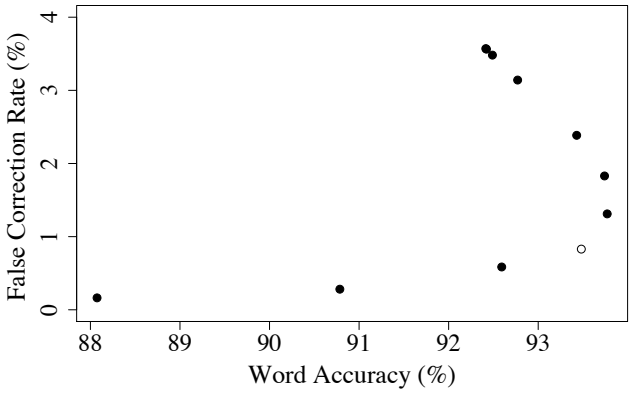

Figure 3. Tradeoff between overall word accuracy and false correction rate, managed by varying the autocorrect threshold from 0.0 (upper right) to 0.95 (lower left). The rightmost dot is optimal from an overall word accuracy standpoint, but we chose the white dot as an operating point to bring the false correction rate below 1%.

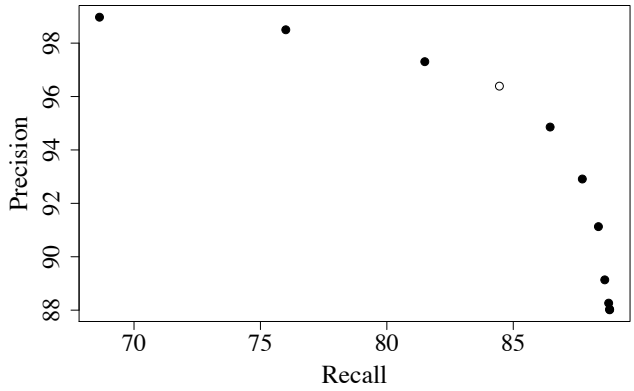

Figure 4. Precision vs. recall in word correction, generated by varying autocorrect threshold from 0.0 (lower right) to 0.95 (upper left). Operating point is white dot, equivalent to Figure 3.

expectation that the STK will output the literal string, and autocorrecting to another string is particularly undesirable. We therefore may wish to decrease the rate of false corrections, even at the cost of a slight increase in overall word error rate.

We found that this tradeoff can be managed by introducing an *autocorrect threshold* parameter to our model. It operates as follows: If the top-scoring word is not equal to the literal string, *and* its probability[6] is below the autocorrect threshold, we prevent the decoder from performing a correction and simply *use the literal string*.

Figure 3 illustrates the tradeoff, calculated on a subset of the development set. It turns out that allowing the decoder to *always* perform correction, i.e. an autocorrect threshold of 0.0, results in a high rate of false corrections, more than 3.5%. This amounts to nearly half of all failures for this test. The optimal autocorrect threshold value for maximizing *overall* accuracy is roughly 0.6, which corresponds to the typical accuracy of any given literal string[7]. We selected 0.7 as the operating point for the autocorrect threshold in all of our accuracy experiments. This choice caused a slight decrease in overall accuracy, but it brought the false correction rate below 1%. On the test data, the false correction rate was 0.70% for the baseline model and 0.39% for the uniform cache mixture model, representing an even larger relative WER improvement than we observed for overall failures.

Precision and Recall

Another way to think about the performance of the word completion model is as a tradeoff between precision and recall. In our case, precision is defined as the proportion of autocorrected words that were changed to the true word. Recall is defined as the proportion of words needing correction (i.e. with incorrect literal strings) that were autocorrected to the true word. Figure 4 shows the precision vs. recall curve. Note that, under the conditions of our model, 100% recall is not possible. This is due to the fact that sometimes a literal string (with an error) is the only word in the predicted word list. This can happen when the decoder fails to generate any candidates because the tap sequence is too far from any known word. In this situation, correction cannot occur in our model (not even a false correction), because no candidates exist.

[6]We converted the log-space model scores to probabilities and normalized to make this calculation.

[7]This is just the accuracy of a word in the no-model scenario, or $1.0 - 0.384$.

Model	Keystroke Savings %
Background LM Only	42.1
Uniform Cache Adaptation	45.7
Exponential Cache Adaptation	45.8

Table 3. Keystroke savings for various model setups, averaged across all simulated Enron users. See Equation 4 for definition of keystroke savings.

Word Prediction

Table 3 shows keystroke savings results for our efficiency experiments. Without any language modeling (not even a lexicon), keystroke savings would not be possible. The background LM increased (potential) keystroke savings from 0% to 42.1%. When using a cache model for LM personalization, this number was further increased to 45.7%, a relative gain of 8.6%. As before, the uniform cache method performed comparably to the exponentially-decaying cache method.

Results by User

The results in Tables 2 and 3 represent averages over all users in the test set, but LM personalization had different effects on different users. Of the 45 users in the test set, 42 exhibited improved typing accuracy under the uniform cache adaptation method. This relative improvement in WER ranged widely, from 2.9% to 30.9%. Of the three users whose accuracy got worse, one had accuracy decrease from 91.4% to 89.3%, one decreased from 91.5% to 91.1%, and one decreased from 92.9% to 92.7%. All three with decreased performance had fewer than 1500 words in their priming emails, which suggests that it may be prudent to delay the application of cache-based adaptation until a sufficient amount of user text has been collected.

CONCLUSION AND DISCUSSION

Through research methodology innovation and evaluation system development, we have investigated the effects of language models, with and without personalization, for improved accuracy and efficiency of contemporary touch screen

keyboards operated with sloppy finger touches. In our benchmark tests, we demonstrated that it was possible to decrease word error rate due to imprecise finger touch tapping from 38.4% to 5.7% using a simple decoder and a background language model. LM personalization further improved this result to 4.6%. Language modeling also improved prediction efficiency to 42.1% (without personalization) and 45.7% (with personalization). The OOV rate of the test set improved from 1.6% to 0.9% using a personalized model. We also showed how raising the autocorrect threshold could lower the false correction rate from 3.5% to 0.7% without significantly affecting overall model performance.

These results were only observable and tuneable because of our novel evaluation methodology. First, we leveraged web-scale data training and entropy pruning to produce a state-of-the-art background LM. Second, the adoption of the Enron Corpus with long-term individual history enabled us to study personalization effects with real-world language use fidelity. Third, we developed a simple yet effective model decoder, which enabled us to measure extrinsic power in an integrated closed loop evaluation system. Fourth, we introduced human-like noisy spatial input in the evaluation based on empirical findings previously reported in the literature. Taken together, these methodological innovations allowed us to efficiently conduct computational experiments over two million words of text, which otherwise would have taken months or years of calendar time to perform. Crucially, this combined closed-loop system also allowed us to report *accuracy* on a large scale (whereas efficiency has been a typical metric in text entry research). We additionally found, somewhat counter-intuitively (but consistent with [30]), that the exponential decay cache model did not outperform the simple uniform cache.

All of these methodological innovations have limitations, as is true of any research methodology. Although we based both the spatial input and the language content on human generated data from [1] and Enron respectively, they were nonetheless simplified to a degree that efficient experiments could be run. Future work needs to replicate and expand these results.

Another opportunity for future work is to study how cognitive errors, particularly spelling errors, affect decoding performance. This would clarify whether the gains we have observed via LM personalization with spatial noise can be observed in a more general error framework. Along this same lines, although we found that adding OOV words to the cache is always desirable in our system, cognitive error would likely introduce the problem of learning misspellings, which may require separate techniques to manage. Other ideas, such as varying the amount of spatial noise, expanding to input methods other than QWERTY, and applying our techniques to gesture typing, are all interesting avenues for future work.

ACKNOWLEDGMENTS

This paper is based on work supported by the National of Institutes of Health under grant R01 DC009834. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect those of the National Institutes of Health or the authors affiliations.

REFERENCES

1. Azenkot, S., and Zhai, S. Touch behavior with different postures on soft smartphone keyboards. In *Proceedings of the International Conference on Human Computer Interaction with Mobile Devices and Services*, MobileHCI '12, ACM (2012), 251–260.
2. Bellegarda, J. R. Statistical language model adaptation: review and perspectives. *Speech Communication 42* (2004), 93–108.
3. Bi, X., Ouyang, T., and Zhai, S. Both complete and correct? multi-objective optimization of touchscreen keyboard. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '14, ACM (2014).
4. Charniak, E., Blaheta, D., Ge, N., Hall, K., Hale, J., and Johnson, M. BLLIP 1987-89 Wall Street Journal Corpus Release 1.
5. Chelba, C., Brants, T., Neveitt, W., and Xu, P. Study on Interaction between Entropy Pruning and Kneser-Ney Smoothing. In *Proc. Interspeech*, ISCA (Makuhari, Japan, 2010), 2242–2245.
6. Chen, S. F., Beeferman, D., and Rosenfield, R. Evaluation metrics for language models. Tech. rep., Carnegie Mellon University, 1998.
7. Chiticariu, L., Krishnamurthy, R., Li, Y., Reiss, F., and Vaithyanathan, S. Domain adaptation of rule-based annotators for named-entity recognition tasks. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '10, Association for Computational Linguistics (2010), 1002–1012.
8. Clarkson, P. R., and Robinson, A. J. Language model adaptation using mixtures and an exponentially decaying cache. In *Proceedings of the International Conference on Acoustics, Speech and Signal Processing*, vol. 2 of *ICASSP '97*, IEEE (1997), 799–802.
9. DeMori, R., and Federico, M. Language model adaptation. In *Computational models of speech pattern processing*. Springer, 1999, 280–303.
10. Findlater, L., Wobbrock, J. O., and Wigdor, D. Typing on flat glass: examining ten-finger expert typing patterns on touch surfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '11, ACM (2011), 2453–2462.
11. Goodman, J., Venolia, G., Steury, K., and Parker, C. Language modeling for soft keyboards. In *Proceedings of the International Conference on Intelligent User Interfaces*, IUI '02, ACM (New York, NY, USA, 2002), 194–195.
12. Goodman, J. T. A bit of progress in language modeling. *Computer Speech & Language 15*, 4 (2001), 403–434.
13. Holz, C., and Baudisch, P. Understanding touch. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '11, ACM (2011), 2501–2510.

14. Jones, K. S. Towards better NLP system evaluation. In *Proceedings of the Workshop on Human Language Technology*, Association for Computational Linguistics (1994), 102–107.

15. Jurafsky, D., and Martin, J. H. *Speech & Language Processing*, second ed. Prentice Hall, 2008.

16. Katz, S. Estimation of probabilities from sparse data for the language model component of a speech recognizer. In *Proceedings of the International Conference on Acoustics, Speech and Signal Processing*, vol. 35 number 3 of *ICASSP '87*, IEEE (1987), 400–01.

17. Klarlund, N., and Riley, M. Word n-grams for cluster keyboards. In *Proceedings of the EACL Workshop on Language Modeling for Text Entry Methods*, TextEntry '03, Association for Computational Linguistics (Stroudsburg, PA, USA, 2003), 51–58.

18. Klimt, B., and Yang, Y. Introducing the Enron corpus. In *Proceedings of the Conference on Email and Anti-Spam*, CEAS '04 (2004).

19. Kneser, R., and Ney, H. Improved backing-off for m-gram language modeling. In *Proceedings of the International Conference on Acoustics, Speech and Signal Processing*, vol. 1 of *ICASSP '95*, IEEE (1995), 181–184.

20. Kristensson, P. O., and Vertanen, K. Asynchronous multimodal text entry using speech and gesture keyboards. In *Proc. Interspeech* (2011), 581–584.

21. Kristensson, P. O., and Vertanen, K. Performance comparisons of phrase sets and presentation styles for text entry evaluations. In *Proceedings of the International Conference on Intelligent User Interfaces*, IUI '12, ACM (2012), 29–32.

22. Kristensson, P. O., and Zhai, S. SHARK2: A large vocabulary shorthand writing system for pen-based computers. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, UIST '04, ACM (New York, NY, USA, 2004), 43–52.

23. Kristensson, P. O., and Zhai, S. Improving word-recognizers using an interactive lexicon with active and passive words. In *Proceedings of the International Conference on Intelligent User Interfaces*, IUI '08, ACM (2008), 353–356.

24. Kuhn, R., and De Mori, R. A cache-based natural language model for speech recognition. *Pattern Analysis and Machine Intelligence, IEEE Transactions on 12*, 6 (1990), 570–583.

25. MacKenzie, I. S., and Soukoreff, R. W. Text entry for mobile computing: Models and methods, theory and practice. *Human–Computer Interaction 17*, 2-3 (2002), 147–198.

26. Paek, T., and Hsu, B.-J. P. Sampling representative phrase sets for text entry experiments: a procedure and public resource. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '11, ACM (2011), 2477–2480.

27. Sandu, O., Carenini, G., Murray, G., and Ng, R. Domain adaptation to summarize human conversations. In *Proceedings of the Workshop on Domain Adaptation for Natural Language Processing*, Association for Computational Linguistics (2010), 16–22.

28. Smith, S. L., and Goodwin, N. C. Alphabetic data entry via the Touch-Tone pad: A comment. *Human Factors: The Journal of the Human Factors and Ergonomics Society 13*, 2 (1971), 189–190.

29. Stolcke, A. Entropy-based pruning of back-off language models. In *Proceedings of News Transcription and Understanding Workshop*, DARPA (Lansdowne, VA, 1998), 270–274.

30. Tanaka-Ishii, K. Word-based predictive text entry using adaptive language models. *Natural Language Engineering 13* (2 2007), 51–74.

31. Vertanen, K., and Kristensson, P. O. A versatile dataset for text entry evaluations based on genuine mobile emails. In *Proceedings of the International Conference on Human-Computer Interaction with Mobile Devices and Services*, MobileHCI '11 (August 2011).

32. Ward, D. J., Blackwell, A. F., and MacKay, D. J. Dashera data entry interface using continuous gestures and language models. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, UIST '00, ACM (2000), 129–137.

33. Weir, D., Pohl, H., Rogers, S., Vertanen, K., and Kristensson, P. O. Uncertain text entry on mobile devices. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '14 (April 2014), 2307–2316.

34. Yin, Y., Ouyang, T. Y., Partridge, K., and Zhai, S. Making touchscreen keyboards adaptive to keys, hand postures, and individuals: a hierarchical spatial backoff model approach. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '13, ACM (2013), 2775–2784.

35. Zajic, D. M., Dorr, B. J., and Lin, J. Single-document and multi-document summarization techniques for email threads using sentence compression. *Information Processing & Management 44*, 4 (2008), 1600–1610.

36. Zhai, S., and Kristensson, P. O. Shorthand writing on stylus keyboard. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '03, ACM (New York, NY, USA, 2003), 97–104.

37. Zhai, S., and Kristensson, P. O. The word-gesture keyboard: reimagining keyboard interaction. *Commun. ACM 55*, 9 (Sept. 2012), 91–101.