# FastInput: Improving Input Efficiency on Mobile Devices

Jingyuan Zhang[1], Xin Wang[1,2], Yue Feng[1,2], Mingming Sun[1,2] and Ping Li[1]
1 Big Data Lab (BDL-US), Baidu Research
2 National Engineering Laboratory of Deep Learning Technology and Application, China
{zhangjingyuan03, wangxin60, fengyue04, sunmingming01, liping11}@baidu.com

## ABSTRACT

Mobile devices (e.g., smartphones) play a crucial role in our daily lives nowadays. People rely heavily on mobile devices for searching online, sending emails, chatting with friends, etc. As a result, input efficiency becomes increasingly important for real-time communication on mobile devices. Due to the small size of the screen on mobile devices, however, it is oftentimes frustrating for users to correct or update the input sequences on an even smaller input area on the screen. This often causes poor user experience.

In this paper, we focus on improving the input efficiency on mobile devices to offer better user experience. In order to achieve efficient input, there are multiple challenges: 1) how to employ a single, unified representation of the keyboard layouts for different input languages; 2) how to build a framework to correct a mistouch immediately and predict the coming input texts (words or phrases) effectively; 3) how to deploy and evaluate the model on mobile devices with limited computational power.

To address these challenges, we introduce ***FastInput*** to improve the user input efficiency on mobile devices. Three key techniques are developed in FastInput – layout modeling, instant mistouch correction and user input text prediction. We also design solutions for efficient deployment and evaluation of FastInput on mobile devices. The proposed FastInput achieves higher efficiency compared to the traditional input system over millions of user input sequences in different languages.

## KEYWORDS

Input Efficiency; Mobile Devices; FastInput

**ACM Reference Format:**

Jingyuan Zhang, Xin Wang, Yue Feng, Mingming Sun, and Ping Li. 2018. FastInput: Improving Input Efficiency on Mobile Devices. In *The 27th ACM International Conference on Information and Knowledge Management (CIKM'18), October 22–26, 2018, Torino, Italy. ACM, New York, NY, USA*, 9 pages.
https://doi.org/10.1145/3269206.3272006

## 1 INTRODUCTION

The continuing growth of mobile devices consistently provides convenient online information access and real-time communication. Mobile devices have become indispensable to their users for online search, emails and social activities, etc. However, to efficiently conduct these activities on mobile devices, an efficient input method is required. In addition, it is preferred to equip modern input methods with the ability to productively predict the coming text according to user's input habit. It is especially favorable for instant communication with others.


(a) QWERTY for English (b) 10 Key for Chinese Pinyin

(c) AZERTY for Chinese Pinyin (d) 10 Key for Chinese Stroke

**Figure 1: Four layouts for input methods on mobile devices.**

Due to the small size of the screen on mobile devices, it is often frustrating for users to correct or update the input sequences on an even smaller input area on the screen. This usually causes poor user experience. Nowadays, lots of companies, including Apple, Baidu, Google, etc., are making substantial efforts to improve the input efficiency on mobile devices. Here, we summarize the relevant technical challenges as follows:

- There are many different keyboard layouts on mobile devices. Figure 1 shows four different layouts for input methods in English and Chinese on mobile phones. How to model different layouts into one unified representation is not trivial.
- Mistouch happens regularly on mobile devices due to the extensive usage of smart touch screens. It is essential to correct a mistouch directly and predict the coming texts (words or phrases) instantly. This is particularly difficult due to the small layout screen and complicated text sequences.
- Since it is difficult to collect and simulate user's touch behaviors, there is no public method to evaluate the input efficiency in an offline manner. Designing an elegant measurement model for the input efficiency is demanding.
- With limited memory access and instant response requirement, it is much more challenging to deploy the designed model on mobile devices for input efficiency improvement.


*CIKM '18, October 22–26, 2018, Torino, Italy*

ACM ISBN 978-1-4503-6014-2/18/10…$15.00
https://doi.org/10.1145/3269206.3272006

In this paper, we share our experiences in tackling the above issues and introduce a series of practical techniques which could be fairly easily deployed. The core framework is ***FastInput***, which consists of three key components: 1) layout modeling; 2) instant mistouch correction; and 3) user input text prediction.

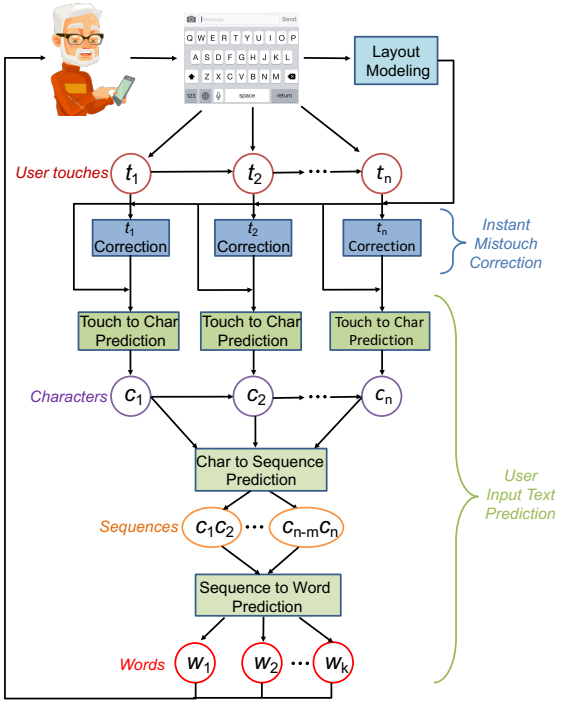



**Figure 2: The work flow of FastInput.**

Figure 2 illustrates the work flow of FastInput. Firstly, we extract a common representation from various layouts for the layout modeling. Secondly, to instantly correct mistouches, we take user's previous input information into consideration and design an effective classification model to learn the true characters a user want to type. Finally, we present an end-to-end prediction system that remarkably boosts input efficiency by transferring user's misspelled sequences of touches into the correct words/phrases users may be interested to input. In addition, we provide solutions for input efficiency evaluation and mobile device compatibility.

In summary, our key contributions include:

- Designing a distance-based strategy for a unified representation of keyboards on mobile devices, regardless of keyboard layouts and languages.
- Developing machine learning models to correct mistouch instantly and predict user input texts effectively.
- Proposing an innovative model to simulate user's touch behaviors for efficiency evaluation offline by replaying user's historical input actions.
- Adapting FastInput to mobile devices with detailed analysis of information reduction and model compression.

## 2 THE FASTINPUT FRAMEWORK

As shown in Figure 2, FastInput consists of three key components for efficiency improvement on mobile devices. We will introduce the detail of each component in the following subsections.

### 2.1 Layout Modeling

People tend to choose different keyboard layouts for their preferred input method editor on mobile devices. For example, people prefer to type English with the QWERTY layout (Figure 1 (a)) while they are more likely to use the 10 key layout (Figure 1 (b)) for Chinese characters. It is time-consuming and a waste of efforts to design different models for each type of the layout to improve the input efficiency. To circumvent this issue, we design a distance-based strategy to extract a unified representation of keyboards on mobile devices, regardless of layouts and languages.

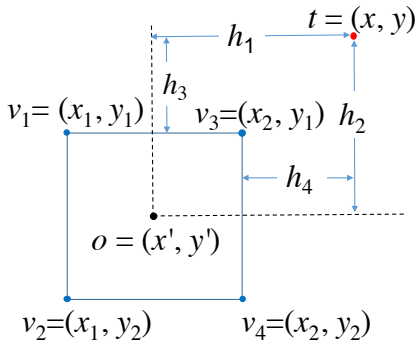



**Figure 3: The distance measure between a touch at location $t$ and a key at location $o$. In this example, $t$ is not in $k$.**

The layout modeling extracts keyboard representations according to the distance from the center of a key to the touch point of a user. Given a coordinate system with the horizontal axis and vertical axis, we denote a touch point as $t = (x, y)$. As illustrated in Figure 3, a key can be represented by its four vertices as $k = \{v_1, v_2, v_3, v_4\}$, where $v_1 = (x_1, y_1)$, $v_2 = (x_1, y_2)$, $v_3 = (x_2, y_1)$ and $v_4 = (x_2, y_2)$ are the left top, left bottom, right top and right bottom vertices, respectively. We denote the center of $k$ as $o = (x', y')$, where $x' = (x_1 + x_2)/2$ and $y' = (y_1 + y_2)/2$.

We further denote the representation for a touch point $t$ and a given key $k$ as a quad $h(t, k) = \{h_1, h_2, h_3, h_4\}$, where

$$h_1 = x - x', \qquad h_2 = y - y',$$

$$h_3 = \begin{cases} \min(|x - x_1|, |x - x_2|) & \text{if } t \text{ not in } k \\ 0 & \text{else,} \end{cases}$$

$$h_4 = \begin{cases} \min(|y - y_1|, |y - y_2|) & \text{if } t \text{ not in } k \\ 0 & \text{else.} \end{cases}$$

Figure 3 provides an example of how to calculate $h(t, k)$. Given any keyboard, we mainly focus on the 26 characters from A to Z and denote the representation of a touch $t$ in the layout as a vector $\mathbf{h}_t \in \mathbb{R}^{104}$ by concatenating all the distances of $h(t, k)$ for $k$ starting from A to Z. The length of $\mathbf{h}_t$ is $104 = 26 \times 4$. Based on the layout modeling, we build machine learning models to correct mistouch instantly and predict user input texts effectively.

## 2.2 Instant Mistouch Correction

In order to correct user mistouches on mobile devices, we first study the input behavior of users to figure out how mistouches are generated. Then we apply a classification technique to learn what character a user would actually mean to type when s/he touches the keyboard.

*2.2.1 Understanding User Behaviors.* As is shown in Figure 1, there are gaps between two characters. With a small virtual keyboard, people oftentimes mistouch the gaps or adjacent characters when s/he types on mobile devices. We study the falling positions of touches based on billions of input sequences and Figure 4 visualizes the touching area of each character on a QWERTY layout. For simplicity, function keys including "shift", "control", "enter", etc, are represented with numbers in Figures 4 and 5. It is found that about 20% of the touches are falling out of the boundaries of characters. Such mistouches would significantly reduce the input efficiency.

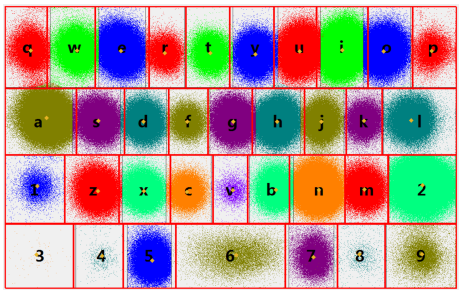


**Figure 4: The touching areas on mobile devices. The red lines are the boundaries of characters on the keyboard.**

We consider a basic strategy for correcting mistouches, by virtually expanding the key boundaries according to user's touch areas in Figure 4. Each key is extended to a larger rectangle that includes all the touching points for this character. Figure 5 visualizes the enlarged keyboard for the QWERTY layout. In this way, each touch can be easily assigned to a key.

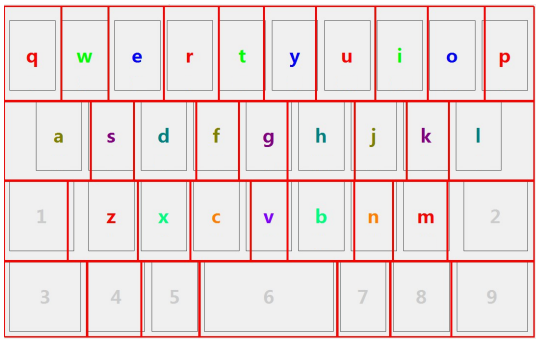
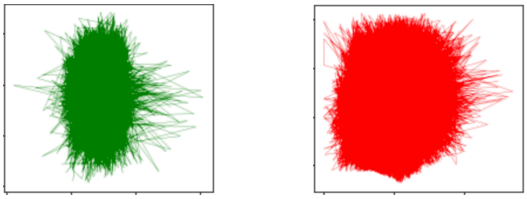



**Figure 5: The enlarged keyboard for the QWERTY layout on mobile devices. The red lines are the extended boundaries of characters on the keyboard.**

Obviously, this simple basic correction method overlooks a lot of mistouches due to different touching behaviors caused by different character orders in sequences. Figure 6 demonstrates how the touching area of a character $c$ differs from each other when the character is in different positions of a word sequence. It can be observed that the touching area of $c$ is more centralized when it is the first character in sequences (in Figure 6 (a)) while the area is more dispersed if $c$ is after a character in sequences (in Figure 6 (b)). That suggests the falling point of a character is not only related to the layout of the keyboard, but also relevant with the order of the character in the input sequence. In order to correct a touch effectively, we have to take both the layout and the character orders into consideration.



(a) $c$ is the first character (b) $c$ is not the first one

**Figure 6: The influence of the order of a character $c$ on the touching area.**

*2.2.2 Learning User Touches.* The instant mistouch correction task can be modeled as a classification problem by learning from user's previous input behaviors to predict which character a user would like to input when s/he makes a touch on the keyboard. Formally, let $\mathcal{X} = \{\mathbf{X}_1, \cdots, \mathbf{X}_N\}$ denote the input data of $N$ touches. $\mathbf{X}_i = \{X_{i1}, \cdots, X_{iM}\}$ is the $i$-th set of features in $\mathcal{X}$, which represents the information related to the $i$-th touch. $\mathbf{z} = \{z_1, \cdots, z_N\}$ denotes the multi-class labels, where $z_i \in \{A, \cdots, Z, \sqcup\}$ is the label of $\mathbf{X}_i$. $\sqcup$ denotes the non-letter character, including the numbers and punctuations during the input process. For simplicity, in this paper, we focus on the classification of these 27 characters only.

Recall that both layouts and character orders are important for predicting user touches accurately. As is discussed in Section 2.1, the touch information in a layout can be represented as a vector $\mathbf{h} \in \mathbb{R}^{104}$. For the character orders, only the previous characters are available and they are the most relevant information. Hence, we can build a character transition dictionary $\mathcal{D} = \{D_p, D_{pp}, D_{ppp}, \cdots\}$ from large amounts of existing input sequences to model the order influence of characters. $D_p$ records the transition information from a previous character to a current one, $D_{pp}$ is about the transition information from the previous two characters, and so on. Specifically, each element $D_p(c_i, c_j) \in D_p \in \mathbb{R}^{27\times 27}$ calculates the probability of the character $c_j$ occurring after $c_i$ as follows,

$$D_p(c_i, c_j) = Prob(c_j|c_i) = \frac{count(c_i c_j)}{count(c_i)}.$$

For those characters at the beginning or the end of a text sequence, we pad "$\sqcup$" before or after for the calculation. To avoid zero probabilities, we simply add one to the count of two consecutive characters $c_i c_j$ and adjust the denominator of $count(c_i)$ accordingly as the

Laplace Smoothing [5] in language modeling. Therefore, $D_p$ is a matrix with $27 \times 27$ probabilities. In a similar way, we can calculate other transition probabilities $D_{pp} \in \mathbb{R}^{27^2 \times 27}$, $D_{ppp}$, and so on.

Given the pre-built knowledge $\mathcal{D}$ and the last $q$ touches $t_q t_{q-1} \cdots t_2 t_1 t$, we can assume that the previous touches $t_q t_{q-1} \cdots t_2 t_1$ have already been matched with the character sequence $c_q c_{q-1} \cdots c_2 c_1$. We represent the character order influence on the current touch $t$ as a concatenated vector $\mathbf{s}_t = \{D_p(c_1, :), D_{pp}(c_2 c_1, :), \cdots, D_{p \cdots p}(c_q c_{q-1} \cdots c_2 c_1, :)\}$. Here $D_p(c_1, :) \in \mathbb{R}^{27}$, $D_{pp}(c_2 c_1, :)$ and $D_{p \cdots p}(c_q c_{q-1} \cdots c_2 c_1, :) \in \mathbb{R}^{27}$ correspond to the row vector of $D_p$, $D_{pp}$ and $D_{p \cdots p}$, respectively. With more previous touches, it requires more time and storage space to calculate the transition probability. But the improvement of helping correcting instant mistouches will not be that significant. In the experiments in Section 5.1, we will show that the result is satisfying enough with only $D_p$ and $D_{pp}$.

By concatenating the layout modeling and the character order influence into the feature representation $\mathbf{X} = \{\mathbf{h}_t, \mathbf{s}_t\}$ for a touch $t$, we feed it into a classification method to learn the corresponding character of a touch $t$. Due to the data privacy, user's input touches are encrypted. We can only randomly sample from the streaming input touches and study the anonymized touch sequences in an incremental way. To efficiently learn from user input streams on mobile devices, we apply an incremental linear classification algorithm [31] to solve this task. We name it as the IMC method (**I**nstant **M**istouch **C**orrection). After training IMC, we will obtain the probability values of each character for a touch. We select the character with the highest probability as the prediction result. IMC is deployed in an online manner. If a user mistakenly touches a gap or an adjacent key on the screen, the proposed IMC model can accurately learn the character the user would like to input and instantly correct the mistouch.

## 2.3 Input Text Prediction

After the instant mistouch correction task, we would like to further learn the words/phrases (sequences of characters) a user intends to input. This component can boost the input efficiency by displaying the possible texts to users even if s/he inputs misspelled sequences of touches. In this section, we propose an end-to-end system ITP (Input Text Prediction) to discover the potential words at each time a user touches the keyboard. The input efficiency can be improved if the desired word is always corrected by the system although it is mistakenly typed by the user.

Currently, deep learning technologies are widely used for the end-to-end learning task [7, 25, 27, 30] (See Section 6 for more details). It is, however, typically difficult to fit the complex deep learning models on mobile devices with limited memory access. Therefore, as is shown in Figure 2, we propose the ITP system with three modules – Touch2Char (the touch to character prediction), Char2Seq (the character to sequence prediction) and Seq2Word (the sequence to word prediction). Each module is effective and efficient. The three modules together help the ITP system enhance the input efficiency on mobile devices.

*2.3.1 Touch to Character.* In order to predict user's input texts precisely, the first step is to correct user's touch sequences to the corresponding character sequences. The previous instant mistouch correction model IMC can be directly applied here. However, IMC model only considers the previous characters. For the text prediction task, the nearest characters before and after a touch can be used to increase the possibility of a touch sequence becoming a meaningful word or phrase. Hence, we design Touch2Char with the same incremental linear classification algorithm [31] in IMC to find the most possible character of each touch in the sequence. Differently, we add more features by constructing more transition information to capture character relations in meaningful words/phrases.

In addition to $D_p$, $D_{pp}$ and other character transition statistics for the previous characters from IMC, we add $D_n$, $D_{nn}$, $D_{pn}$, $D_{ppn}$, $D_{ppnn}$ and other transition probabilities in Touch2Char. The final character transition dictionary for Touch2Char is denoted as $\mathcal{D}'$. Similar to $D_p$, $D_n \in \mathbb{R}^{27 \times 27}$ records the probability information of transiting to the current character given the one next to it. Similar to $D_{pp}$, $D_{nn}$ (or $D_{pn}$)$\in \mathbb{R}^{27^2 \times 27}$ are about the transition probabilities given the next two characters (or the previous and the next characters). $D_{ppn}, D_{pnn} \in \mathbb{R}^{27^3 \times 27}$, $D_{ppnn} \in \mathbb{R}^{27^4 \times 27}$ and other transition probabilities can be calculated correspondingly. With more and more transition statistics, the performance improvement of Touch2Char becomes less and less significant. In the experiment, we collect the transition information within five characters (e.g., $D_n$ with two characters, $D_{pn}$ with three and $D_{ppnn}$ with five characters) and achieve satisfying results as is shown in Table 2.

Compared to IMC focusing on detecting a current mistouch, Touch2Char makes use of more information to generate more meaningful character sequences. For example, assume we have already had a touch sequence "ha". When we input "s", "p" and "y" one by one, we may not correct anything for each touch since "as", "sp", "py", "has", "asp" and "spy" have much higher probabilities to appear together according to the transition information $D_p$ and $D_{pp}$ in IMC model. However, with the transition dictionary $\mathcal{D}'$, the Touch2Char module may discover that "p" should be the most possible character to be fit in the touch sequence "ha?py". In this way, Touch2Char will consider "happy" as a desirable character sequence the user would like to type instead of "haspy".

*2.3.2 Character to Sequence.* Given the touches a user has already input until now, the Touch2Char module will produce a predicted probability value of every character for a touch in the sequence. If we simply select the character with the highest probability value for each touch, we may miss the real sequence of characters a user would like to input. For instance, a user mistakenly touches "jam" for "ham" since "j" is the adjacent key of "h" in the QWERTY layout. We cannot predict "ham" correctly because "jam" may be the most probable character sequence if we type "j", "a" and "m" one by one. However, with a touch sequence of length $L$, there are $27^L$ combinations of character sequences. It would be too time-consuming or even impossible for larger $L$ to calculate and rank the possibility scores for each combination.

Our Character to Sequence module Char2Seq aims to select the most probable sequences of characters as meaningful texts that the user plans to input in an effective and efficient way. As a widely-used algorithm for the most possible sequence selection task, the beam search algorithm [7, 33, 35] is employed to optimize the generation of sequences of characters. Beam search is a greedy algorithm that explores the solution space by expanding the most

promising partial solutions in a limited set. It is a heuristic method that optimizes the breadth-first search to reduce the computation cost and memory requirements. In addition, beam search can be easily implemented in an incremental manner to deal with the streaming touch sequences.

In our Char2Seq module, the beam search algorithm works as follows: at each time of a touch, we keep track of the best $k$ hypotheses, i.e., the most possible $k$ sequences of characters after the current touch. We refer to $k$ as the beam size. When a user makes a new touch at a following time step, we will have $V = 27$ new possible characters for the new touch. It makes a total of $V \times k$ new hypotheses (sequences). We will score these sequences by their overall probabilities and only keep the best $k$ ones. In this way we always keep a beam of sequences and the procedure repeats until we reach the end of the sequence. The best $k$ sequences with the highest probability scores are the output of the Char2Seq module, representing as the text candidates a user expects to input.

*2.3.3 Sequence to Word.* With the $k$ candidate texts from the Char2Seq module, it is still challenging to discover the most meaningful one that a user would like to type. For example, if a user types "holf", the Char2Seq module may find that "hold", "hole" and "half" are the top three words with equal possibilities. In this case, the Seq2Word step aims to incorporate more information to further distinguish the $k$ texts and return the best one to the user.

Motivated by the learning to rank techniques widely used in search engines [22, 28, 39], we utilize the LambdaRank algorithm [3] to come up with an optimal ordering of the $k$ texts and present the top one as the final text that satisfies the user's expectation most. LambdaRank transforms ranking into a pairwise regression problem by ordering a pair of items at a time and using it to get the final ranking results. It can also be easily implemented in an incremental manner to deal with the streaming touch sequences.

Besides the overall probability value of each candidate text, we take other valuable information into account and feed them together to the LambdaRank model to learn a ranking score for the text. The valuable information includes the popularity of the text, the number of matched words/phrases in a prior-known corpus, and the frequencies of those matched texts, etc. Here the prior-known corpus records all the mappings from an input sequence to the selected word/phrase. For instance, if a user types "ok", s/he may choose "ok" or "okay" as the displayed word. The corpus will record the map from "ok" to "okay". However, it would be too large if the corpus saves each possible mapping since an input sequence may match to tens of or even hundreds of different words/phrases, especially for the Chinese language. In this paper, we mainly focus on the five most frequent texts for a sequence and use their frequencies for the ranking task. The LambdaRank algorithm produces a ranking result for the $k$ candidate texts. The top one ranked text is considered as the most possible one the user would like to type.

In summary, with the three modules, the Input Text Prediction system ITP helps improve the input efficiency by displaying the correct words/phrases to users even if they misspelled the text.

## 3 MOBILE DEVICE COMPATIBILITY

The proposed Touch2Char module in the ITP (Input Text Prediction) component needs to store a large amount of pre-calculated character transition probabilities. For example, the transition dictionary $D_{ppnn}$ records the probabilities of $27^4 \times 27$ combinations of five characters, which takes around 55 megabytes in memory (using 4 bytes to store each float value). In mobile devices, the memory access for the input method are limited to only several hundreds of kilobytes. It is impossible to fit all the pre-calculated probabilities into mobile devices.

In order to adapt Touch2Char to the environment of mobile devices, we propose to select the most important transition probabilities during the classification process. We first analyze the effectiveness of different transition probabilities in achieving reasonable classification results. It is observed that the classification accuracy of using $D_{ppn}$ and $D_{pnn}$ is very close to the performance of using all probabilities within five characters in Table 2. Therefore, we only keep $D_{ppn}$ and $D_{pnn}$ as our main transition knowledge on mobile devices for the input text prediction task.

However, both $D_{ppn}$ (or $D_{pnn}$) have $27^3 \times 27$ probability values. Each still takes large memory space of around 2 megabytes. What's worse, the probability value in the range of $[0.0, 1.0]$ saved in the double (or float) type could reduce the processing and running speed on mobile devices. Hence, we propose to divide the range of $[0, 1]$ into $n$ equal intervals and use bit values to represent the probability falling in each interval. For example, we can use the binary (one bit) representation by setting $n = 2$. A probability value is represented as "0" if it is in the range of $[0, 0.5)$. Otherwise, it is "1". Such bit representations help reduce the memory space to a large extent. With a smaller $n$, less space is needed while more transition information is missed, which may cause more significant drop of the performance for Touch2Char.

We evaluate the influence of the one-bit representation on the performance of Touch2Char. It is discovered that the accuracy drops by 0.357% compared with the original float representation. We further split $[0, 1]$ into $n = 4$ intervals and represent the probability as a two-bit value. Specifically, "00" presents the probability falling in the range of $[0, 0.25)$, "01" is about the value in $[0.25, 0.50)$, "10" is about the value in $[0.50, 0.75)$ and "11" is about the value in $[0.75, 1.0]$. The accuracy of the two-bit representation drops only 0.025% compared with the float representation. Hence, we set $n = 4$ to compress the transition probabilities $D_{ppn}$ and $D_{pnn}$ to a more efficient representation on mobile devices. In this way, any three characters in $D_{ppn}$ needs $27 \times 2 = 54$ bits for the transition probabilities over the 27 characters, i.e., 7 bytes for the storage. Each 7 bytes for any three characters are considered as a unit and are stored in a continuous space so that it can be conveniently located and read out for the classification method in the Touch2Char module. The total memory of $D_{ppn}$ or $D_{pnn}$ now becomes $7 \times 27^3 = 137,781 = 135$ kilobytes. Compared to the space required by float representations, this is a huge saving with little drop of classification performance.

## 4 INPUT EFFICIENCY EVALUATION

With the proposed FastInput framework, our next step is to evaluate the benefit of deploying FastInput into the mobile devices. In the following, we first introduce the evaluation metric for the instant mistouch correction (IMC). Then we show how to evaluate the input text prediction (ITP).

## 4.1 Evaluation of IMC

For each touch sequences, if a backspace appears, we believe a mistouch happens. The mistakenly touched one before the backspace and the correct one after the backspace are located for the training and evaluation of IMC. In the experiment, we use error rate, macro average and micro average false positive rates [32] as the performance measures of instant mistouch correction. The macro average false positive rates (macro-FPR) computes the FPR independently for each class and then takes the average as the result. The micro average false positive rates (micro-FPR) will aggregate the contributions of all classes to compute the average FPR. We use the false positive rate FPR in the experiment because it is important to help analyze the cost of the unnecessary actions of correcting a touch.

## 4.2 Evaluation of ITP

With anonymized touch sequences in an online streaming manner, it is much more challenging to verify the quality of the proposed ITP model. Currently, there is no public method to evaluate the input efficiency in an offline manner.

In this paper, we propose an innovative model OIS (Offline Input Simulation) to mimic user's touch behaviors in an offline manner. The intuition is to study how user reacts to a mistouch on the keyboard. When a mistouch happens, a user will first search from the list of candidate texts. If s/he cannot find the desired text, he will press the backspace key to correct the mistouches. However, if our framework FastInput could successfully predict the expected text, backspaces are no longer needed and the input efficiency can be improved. The OIS model replays a user's original input touches by simulating when s/he presses the backspace key to correct a mistouch under the FastInput framework. The simulation process is as follows: Assume a user has already input a sequence of touches as $t_0 t_1 \cdots t_n$ and s/he wants to select his/her expected text $w$. If $w$ is not in the predicted ranking list obtained from FastInput, a backspace is pressed to mimic the user removing the last touch $t_n$. The user may type a new character $t'_n$ or do nothing. With a new sequence of touches $t_0 t_1 \cdots t_{n-1}$ or $t_0 t_1 \cdots t_{n-1} t'_n$, FastInput will generate a new ranking list of candidate texts. If $w$ is not in the list again, another backspace is needed. This procedure repeats until we find $w$ in the predicted list of FastInput or we enter a backspace to remove the first touch $t_0$ of the sequence. Once $w$ is found in the predicted list or $t_0$ is removed, the OIS model will enter the characters in $w$ to finish correcting $t_0 t_1 \cdots t_n$. Then OIS will move to the next touch sequence until all the sequences are replayed out.

The simulated touches generated by OIS are considered as the output under our proposed ITP model. By comparing the original touches with the new ones, we can easily evaluate the performance of ITP. In this paper, we compute the backspace rate between the original and simulated touches as the performance measure. To evaluate the input efficiency from the character level, we run the ITP model after a user input a touch and compute the backspace rate for each touch. We denote this measure as BR@T. In addition, we measure the input efficiency from the sequence level by running the ITP model after a backspace is typed by the user. We assume that the user has finished typing a sequence of characters if s/he inputs a backspace. In this way, we can calculate the backspace rate for each sequence. We denote it as BR@S.

Furthermore, for a more intuitive evaluation of the input efficiency, we provide two more measurements: IE@T and IE@S. IE@T is about the input efficiency from the character level and IE@S is from the sequence level. After each running of ITP, IE@T (or IE@S) calculates the ratio between the length of the correct character sequence and the actual sequence (each backspace is included and counted once) input by the user. For example, a user mistakenly typed "haspy" for "happy". From the character level, the user will type "has<ppy" if s/he realized the wrong touch "s" immediately. Here "<" means the backspace inserted by the user. So the user touches 7 times for "has<ppy" to correct "happy" with 5 touches and the IE@T will be $5/7 = 71.43\%$. Similarly, IE@S for "happy" would be $5/11 = 45.45\%$ because "haspy<<<ppy" is needed to correct "happy" after the whole sequence "haspy" is typed. These four types of metrics measure the input efficiency of ITP from different perspectives.

# 5 EXPERIMENT

In this section, we conduct extensive experiments to evaluate the proposed FastInput framework. In order to show the effectiveness of FastInput on different languages, we train and test on Chinese and English input touches. For each language, 1,000,000 touch sequences are selected for training and another 1,000,000 ones are for testing. Each touch sequence records the input behavior of a user when s/he is typing a piece of text (word/phrase). The backspaces are included if a user tries to correct his/her input touches. Since we mainly focus on improving the input efficiency of words/phrases on mobile devices, we remove the numbers and punctuations appeared in each touch sequence.

User's input touches are encrypted to protect data privacy and cannot be downloaded and saved for any purpose. To train and test FastInput, we can only fetch anonymized touch sequences one by one from the online streaming input touches and feed each touch sequence into the IMC and ITP systems for training and evaluation. Each module in ITP can be employed in an incremental way effectively and efficiently.

## 5.1 Performance of IMC

In order to show the effectiveness of the instant mistouch correction method (IMC), we compare with the following methods:

- Layout: It is a traditional mistouch correction method. As is shown in Figure 5, a larger touch boundary of a key is first decided and fixed. Then each touch would be assigned to a key during the prediction.
- IMC: The proposed method in this paper. We combine the layout information, the influence of the previous characters together for the mistouch correction.

In the experiment, we use the incremental LIBLINEAR method [31] for classification. Specifically, we select the L2-regularized L1-loss support vector classification (dual) and other default parameters for the classification method. In order to show the influence of previous characters, we derive two variations of IMC. IMC ($D_p$) only considers one previous character during the correction and IMC ($D_p + D_{pp}$) takes both the previous one and two characters into

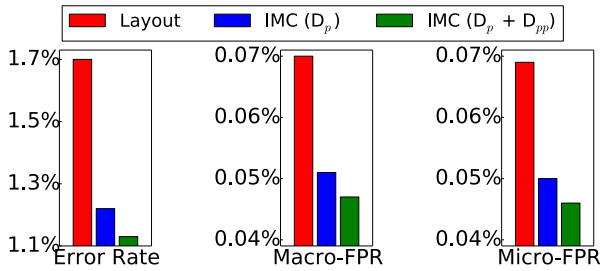



**Figure 7: The IMC result on the Chinese touches. Lower values mean better performance for all the three metrics.**

account. Figure 7 presents the performance on the Chinese input data. It can be observed that our proposed IMC has a significant improvement compared with the traditional Layout method. Though a series of complicated rules are designed to set the real boundaries of keys by analyzing huge amounts of user's historical input behaviors, the influence of character orders in word/phrase sequences are ignored. By incorporating character order information into the touch prediction task, the proposed IMC model outperforms the Layout method. In addition, IMC ($D_p + D_{pp}$) performs slightly better than IMC ($D_p$), showing that more previous characters can help improve the instant mistouch correction task.

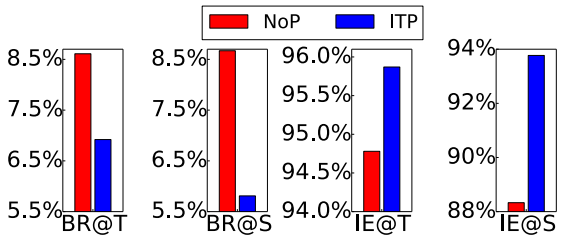
## 5.2 Performance of ITP

Besides the instant mistouch correction, we also study the input text prediction (ITP) to auto-correct user's input sequences of characters. In order to evaluate the performance of ITP, we compare with the following methods:

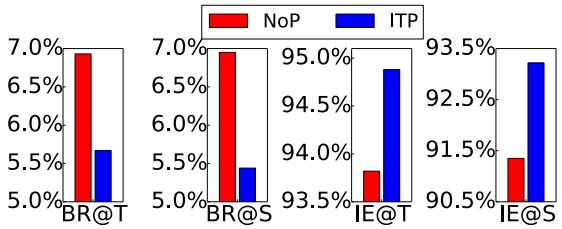
- NoP: It is a classical method without any text prediction technique. Given a touch, we simply locate it to the key according to the enlarged key boundary in Figure 5. The interval area between two keys are unequally divided according to the frequency of the adjacent keys. No sequence information is taken into account and it involves no text predictions.
- ITP: The proposed method in this paper. We combine the Touch2Char, Char2Seq and Seq2Word together to boost the performance of the input text prediction.

We use the same parameter settings for the classification method in Touch2Char as in the IMC method. For the beam search in the Char2Seq module, we set the beam size $k$ to be 5. For the LambdaRank algorithm in the Seq2Word module, we use the standard tree implementation of (robust) logitboost [11, 21].

Figure 8 presents the performance of ITP on both the Chinese and English data. Here the transition probabilities within five characters are all applied in the Touch2Char module of ITP with the float representation. It can be observed that our proposed ITP performs the best for all the four types of metrics on both data. Compared to the NoP baseline, ITP achieves a significant improvement over all the metrics, showing the effectiveness of applying text prediction techniques. By incorporating the classification method, the beam search algorithm and the learning to rank model into the ITP system, users' input efficiency is greatly enhanced.




(a) The ITP performance on the Chinese sequences.




(b) The ITP performance on the English sequences.

**Figure 8: The ITP performance on the Chinese and English sequences. For the backspace rate BR@T and BR@S, the lower the value the better the performance. For the input efficiency IE@T and IE@S, the larger the value the better the performance.**

We further present some case studies to show the effectiveness of the proposed ITP system. For the tested input sequences, we select 5 touch sequences in Chinese and another 5 in English as shown in Table 1. Our FastInput framework predicts each sequence correctly without inserting any backspace. We further show the simulation results of the baseline method NoP under our proposed Offline Input Simulation (OIS) model. "OIS for NoP@S" means the simulation procedure of NoP at the end of a sequence while "OIS for NoP@T" is about the simulation after each touch in a sequence. For example, in Table 1 (b), if a user wrongly typed "bitthdat" for "birthday", the OIS for NoP@S will have 6 backspaces inserted to get the correct word. But the OIS model for NoP@T will insert only 2 backspaces. It can be observed that the OIS model captures user's touch behaviors in an offline manner to help assess the input efficiency accurately. Under the OIS method, our proposed FastInput framework can predict each sequence correctly without any backspace insertion, no matter at the touch level or at the sequence level.

## 5.3 Compatibility analysis on mobile devices

Section 5.2 shows the result of ITP with the transition probabilities (in float representation) within five characters in the Touch2Char module. In this section, we study the compatibility of the FastInput framework on mobile devices as described in Section 3. We first test the performance of Touch2Char with different transition probabilities in the float or bit representations. Table 2 shows the classification accuracy of the Touch2Char model on the Chinese data. "ppn+pnn (b)" and "ppn+pnn (f)" are about the models with $D_{ppn}$ and $D_{pnn}$ in two-bit and float representations, respectively. "ppnn$_{no}$ (b)" and 'ppnn$_{no}$ (f)" are about the model without $D_{ppnn}$ in two-bit and float representations, respectively. "all" is the model

**Table 1: Case studies on the Chinese and English input sequences.**

(a) Examples from the Chinese input sequences.

| Input Sequence | quandoiyao | wohauiyi | tingyi | xinfqi | sgiguang |
|---|---|---|---|---|---|
| English Translation | All to be | I doubt | agree | week | time |
| Correct Text (Predicted by ITP) | quandouyao | wohuaiyi | tongyi | xingqi | shiguang |
| OIS for NoP@S | quandoiyao<<<<uyao | wohaui<<<uaiyi | tingyi<<<<<ongyi | xinfqi<<<gqi | sgiguang<<<<<<<higuang |
| OIS for NoP@T | quandoi<uyao | woha<uu<aiyi | ti<ongyi | xinf<gqi | sg<higuang |

(b) Examples from the English input sequences.

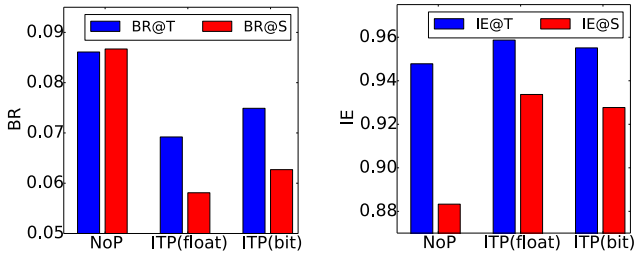
| Input Sequence | bitthdat | anttime | faicly | costomer | thiught |
|---|---|---|---|---|---|
| Correct Text (Predicted by ITP) | birthday | anytime | fairly | customer | thought |
| OIS for NoP@S | bitthdat<<<<<<rthday | anttime<<<<<ytime | faicly<<<rly | costomer<<<<<<<ustomer | thiught<<<<<ought |
| OIS for NoP@T | bit<rthdat<y | ant<ytime | faic<rly | co<ustomer | thi<ought |

**Table 2: Classification accuracy of Touch2Char.**

| Touch2Char | ppn+pnn (b) | ppn+pnn (f) | $\text{ppnn}_{no}$ (b) | $\text{ppnn}_{no}$ (f) | all |
|---|---|---|---|---|---|
| Accuracy | 95.14% | 95.19% | 95.22% | 95.29% | 95.31% |

with all probabilities within five characters in the float representation. Here we test the performance of $\text{ppnn}_{no}$ to see the influence of $D_{ppnn}$. It can be observed that the performance does not change much if we exclude $D_{ppnn}$ from the Touch2Char model. The reason is that the transition information of $D_{ppnn}$ may be indirectly inferred from $D_{ppn}$ and $D_{pnn}$. We can also observe that "ppn+pnn (b)" does not drop too much compared with others. Hence, we can safely use the two-bit representation of $D_{ppn}$ and $D_{pnn}$ to help achieve good performance and save large memory space on mobile devices simultaneously.

In addition, we analyze the performance of the Input Text Prediction (ITP) with the bit representations. Figure 9 presents the result comparisons on the four measurements BR@T, BR@S, IE@T and IE@S. "ITP (bit)" is the ITP method using the bit representation of $D_{ppn}$ and $D_{pnn}$ only in the Touch2Char module. "ITP (float)" is the proposed method in Section 2.3. ITP (bit) drops a little bit compared to ITP (float) since it loses some information of the character transitions. However, ITP (bit) still outperforms other baseline methods on the four types of measurements.




(a) BR results (the lower the better) (b) IE results (the higher the better)

**Figure 9: Performance of different text prediction models.**

With the better performance of ITP (bit), we deploy it to the mobile devices in C and record the average running time of each module over 100,000 touch sequences. Table 3 shows the running time of ITP (bit) in seconds. Both the loading time of the pre-calculated knowledge and the average prediction time for each input sequence are recorded in Table 3. It can be observed that the proposed FastInput framework is truly efficient on mobile devices.

**Table 3: Running time (sec) after adapting to mobile devices.**

| Modules | Layout Modeling | Touch2Char | | Char2Seq + Seq2Word |
|---|---|---|---|---|
| | | Transition Dictionary | Classification Model | |
| Loading Time | 0.000423 | 0.001489 | 0.003750 | 0.021224 |
| Modules | Sequence Parsing | Touch2Char | | Char2Seq + Seq2Word |
| Prediction Time | 0.000017 | 0.000358 | | 0.020923 |

## 6 RELATED WORK

A language can be represented in different writing systems. For example, the Chinese language can be represented in Pinyin (or Romanized Chinese), Zhuyin and Hanzi. Pinyin and Zhuyin are phonetic systems for representing Hanzi characters. A lot of patents have been filed to identify the input sequence of characters under different writing systems [26, 36, 40]. There are some other research and patents [8, 18, 23, 34, 38] of predicting texts or conversions of input. To the best of our knowledge, however, no instant mistouch correction and offline efficiency evaluation are addressed.

Due to its scalability, linear classification methods [6, 9, 10, 20, 31] are commonly used in many applications such as text classification [15, 24]. In order to efficiently learn from user input streams on mobile devices, we apply an incremental linear classification algorithm of LIBLINEAR [31] for the instant mistouch correction task and the input text prediction task. LIBLINEAR is an easy-to-use tool that supports L2-regularized logistic regression [19], L2-loss and L1-loss linear support vector machines [1]. It is very efficient and achieves competitive performance compared with other linear classifiers such as Pegasos [29] and SVMperf [17].

In recent years, Recurrent Neural Networks (RNN) [13], together with its variants such as Long-Short Term Memory (LSTM) [12, 14] and Gated Recurrent Unit (GRU) [7] have shown great success in

modeling sequential data. Several RNN-based neural network architectures have been proposed and successfully applied to sequence to sequence learning tasks such as machine translation [7, 30] and text summarization [25, 27]. The neural networks often consist of an encoder and a decoder [7]. The encoder extracts a fixed-length representation from a variable-length input sequence, and the decoder generates a correct sequence (e.g., translation) from this representation. The beam search algorithm [33, 35] is widely used to decode RNNs to generate possible candidates of the output sequence. Due to the complexity of the neural network models, it is difficult to fit the mobile devices. In this paper, we design our own end-to-end learning techniques to improve the input efficiency.

Learning to rank techniques are widely used in commercial search engines and recommender systems [22, 28, 37, 39]. A popular scheme is to transform ranking into a pairwise classification, multiclass classification, or regression problem. For example, the basic idea of pairwise model is to look at pairs of items at a time, come up with the optimal ordering for that pair of items, and then use it to determine the final ranking for all the items. Many methods, including RankNet [2], LambdaRank [3], RankSVM [16], GBRank [41], McRank [22] etc., are proposed and have proven to be very successful algorithms for solving real world ranking problems [4]. In this paper, we employ the classic LambdaRank method with standard tree implementation of (robust) logitboost [11, 21], to further enhance the performance of our FastInput system.

## 7 CONCLUSION

In this paper, we introduce the comprehensive solution of improving the input efficiency on mobile devices. The proposed FastInput framework is effective and practical on mobile devices, as validated by an extensive empirical study. It is clear that, the proposed system is not limited to the Chinese and English input, but also can be leveraged for other input languages. We hope this work presented in this paper will be able to provide useful insights to the related research community in industry as well as academia.

## REFERENCES


[1] Bernhard E Boser, Isabelle M Guyon, and Vladimir N Vapnik. 1992. A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*. ACM, 144–152.

[2] Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. 2005. Learning to rank using gradient descent. In *ICML*. ACM, 89–96.

[3] Christopher J Burges, Robert Ragno, and Quoc V Le. 2006. Learning to rank with nonsmooth cost functions. In *NIPS*. 193–200.

[4] Olivier Chapelle and Yi Chang. 2011. Yahoo! learning to rank challenge overview. In *Proceedings of the Learning to Rank Challenge*. 1–24.

[5] Stanley F Chen and Joshua Goodman. 1999. An empirical study of smoothing techniques for language modeling. *Computer Speech & Language* 13, 4 (1999), 359–394.

[6] Wei-Lin Chiang, Mu-Chu Lee, and Chih-Jen Lin. 2016. Parallel dual coordinate descent method for large-scale linear classification in multi-core environments. In *SIGKDD*. ACM, 1485–1494.

[7] Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. 2014. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259* (2014).

[8] Jannes Dolfing, Brent Ramerth, Douglas Davidson, Jerome Bellegarda, Jennifer Moore, Andreas Eminidis, and Joshua Shaffer. 2017. Predictive text input. (Sept. 12 2017). US Patent 9,760,559.

[9] Mark Dredze, Koby Crammer, and Fernando Pereira. 2008. Confidence-weighted linear classification. In *ICML*. ACM, 264–271.

[10] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. 2008. LIBLINEAR: A library for large linear classification. *Journal of machine learning research* 9, Aug (2008), 1871–1874.

[11] Jerome H. Friedman, Trevor J. Hastie, and Robert Tibshirani. 2000. Additive logistic regression: a statistical view of boosting. *The Annals of Statistics* 28, 2 (2000), 337–407.

[12] Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. 1999. Learning to forget: Continual prediction with LSTM. (1999).

[13] Christoph Goller and Andreas Kuchler. 1996. Learning task-dependent distributed representations by backpropagation through structure. In *IEEE International Conference on Neural Networks*, Vol. 1. IEEE, 347–352.

[14] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.

[15] Thorsten Joachims. 1999. Transductive inference for text classification using support vector machines. In *ICML*, Vol. 99. 200–209.

[16] Thorsten Joachims. 2002. Optimizing search engines using clickthrough data. In *SIGKDD*. ACM, 133–142.

[17] Thorsten Joachims. 2006. Training linear SVMs in linear time. In *SIGKDD*. ACM, 217–226.

[18] Kenneth Kocienda, Greg Christie, Bas Ording, Scott Forstall, Richard Williamson, and Jerome René Bellegarda. 2012. Method, device, and graphical user interface providing word recommendations for text input. (July 31 2012). US Patent 8,232,973.

[19] Kwangmoo Koh, Seung-Jean Kim, and Stephen Boyd. 2007. An interior-point method for large-scale l1-regularized logistic regression. *Journal of Machine learning research* 8, Jul (2007), 1519–1555.

[20] Mu-Chu Lee, Wei-Lin Chiang, and Chih-Jen Lin. 2015. Fast matrix-vector multiplications for large-scale logistic regression on shared-memory systems. In *ICDM*. IEEE, 835–840.

[21] Ping Li. 2010. Robust LogitBoost and Adaptive Base Class (ABC) LogitBoost. In *UAI*. Catalina Island, CA.

[22] Ping Li, Christopher J Burges, and Qiang Wu. 2007. Mcrank: Learning to rank using multiple classification and gradient boosting. In *NIPS*. 897–904.

[23] Christian Liensberger. 2015. Context sensitive auto-correction. (Dec. 22 2015). US Patent 9,218,333.

[24] Kamal Nigam, Andrew Kachites McCallum, Sebastian Thrun, and Tom Mitchell. 2000. Text classification from labeled and unlabeled documents using EM. *Machine learning* 39, 2-3 (2000), 103–134.

[25] Romain Paulus, Caiming Xiong, and Richard Socher. 2017. A deep reinforced model for abstractive summarization. *arXiv preprint arXiv:1705.04304* (2017).

[26] Matthew Robert Scott, Huihua Hou, Weipeng Liu, and Weijiang Xu. 2015. Input Method Editor. (April 30 2015). US Patent App. 13/635,306.

[27] Abigail See, Peter J Liu, and Christopher D Manning. 2017. Get To The Point: Summarization with Pointer-Generator Networks. In *ACL*, Vol. 1. 1073–1083.

[28] Aliaksei Severyn and Alessandro Moschitti. 2015. Learning to rank short text pairs with convolutional deep neural networks. In *SIGIR*. ACM, 373–382.

[29] Shai Shalev-Shwartz, Yoram Singer, Nathan Srebro, and Andrew Cotter. 2011. Pegasos: Primal estimated sub-gradient solver for svm. *Mathematical programming* 127, 1 (2011), 3–30.

[30] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *NIPS*. 3104–3112.

[31] Cheng-Hao Tsai, Chieh-Yen Lin, and Chih-Jen Lin. 2014. Incremental and decremental training for linear classification. In *SIGKDD*. ACM, 343–352.

[32] Vincent Van Asch. 2013. Macro-and micro-averaged evaluation measures [[basic draft]]. (2013).

[33] Ashwin K Vijayakumar, Michael Cogswell, Ramprasath R Selvaraju, Qing Sun, Stefan Lee, David Crandall, and Dhruv Batra. 2016. Diverse beam search: Decoding diverse solutions from neural sequence models. *arXiv preprint arXiv:1610.02424* (2016).

[34] Xin Wang and Brent D Ramerth. 2017. Predictive conversion of language input. (Dec. 12 2017). US Patent 9,842,101.

[35] Sam Wiseman and Alexander M Rush. 2016. Sequence-to-sequence learning as beam-search optimization. In *EMNLP*. 1296–1306.

[36] Genqing Wu, Xiaotao Duan, and Tai-Yi Huang. 2015. Input method editor. (May 5 2015). US Patent 9,026,426.

[37] Murat Yagci, Tevfik Aytekin, and Fikret Gurgen. 2017. On Parallelizing SGD for Pairwise Learning to Rank in Collaborative Filtering Recommender Systems. In *Proceedings of the Eleventh ACM Conference on Recommender Systems*. ACM, 37–41.

[38] Ying Yin, Tom Yu Ouyang, Kurt Partridge, and Shumin Zhai. 2013. Making touchscreen keyboards adaptive to keys, hand postures, and individuals: a hierarchical spatial backoff model approach. In *CHI*. ACM, 2775–2784.

[39] Jun Yu, Dacheng Tao, Meng Wang, and Yong Rui. 2015. Learning to rank using user clicks and visual features for image retrieval. *IEEE transactions on cybernetics* 45, 4 (2015), 767–779.

[40] Jian Zeng, Liangyi Ou, Wei Sun, Xiangye Xiao, Yinfei Zhang, Yonggang Wang, and Yuanbo Zhang. 2017. Input method editor. (Jan. 24 2017). US Patent 9,552,125.

[41] Zhaohui Zheng, Keke Chen, Gordon Sun, and Hongyuan Zha. 2007. A regression framework for learning ranking functions using relative relevance judgments. In *SIGIR*. ACM, 287–294.