INTERSPEECH 2023
20-24 August 2023, Dublin, Ireland

Language Model Personalization for Improved Touchscreen Typing

Jiban Adhikary, Keith Vertanen

Michigan Technological University, USA
jiban@mtu.edu, vertanen@mtu.edu

Abstract

Touchscreen keyboards rely on language modeling to auto-correct noisy typing and to offer word predictions. While language models can be pre-trained on huge amounts of text, they may fail to capture a user's unique writing style. Using a recently released email personalization dataset, we show improved performance compared to a unigram cache by adapting to a user's text via language models based on prediction by partial match (PPM) and recurrent neural networks. On simulated noisy touchscreen typing of 44 users, our best model increased keystroke savings by 9.9% relative and reduced word error rate by 36% relative compared to a static background language model.

Index Terms: language model personalization, human-computer interaction, touchscreen typing

1. Introduction

When people type on a touchscreen mobile device, a language model (LM) facilitates the input process by providing word predictions and auto-corrections. Typically, behind a text entry application there is a generalized LM which works for every user. These models are static and do not change with time. However, every person has a unique style of writing. Moreover, while writing, people tend to repeat the same word or even the same phrase. As such, it is desirable to have a model that can change and adapt to a user's writing behaviour.

In this work, we investigate improving text entry performance by using LMs that adapt to a person's writing history. We conduct simulations on a publicly available and time-ordered dataset. We assume an ideal user of a touchscreen keyboard with word predictions and auto-correct. We test with and without introducing noise to the user's touch locations. We adapt various LMs based on a user's past text and show performance gains using word error rate (WER) and keystroke savings.

2. Related work

Past work [1, 2, 3] used adaptive LMs based on a unigram cache [4, 5] for speech recognition. The prediction by partial match (PPM) algorithm [6] has been used to provide text entry with adaptive language modeling [1, 7]. Another approach is to train an n-gram model on a user's past text [8, 9]. Neural models have been also used for personalization via caching [10, 11, 12], or by fine-tuning the model's parameters [13, 9].

Fowler et al. [3] explored LM adaptation on touchscreen keyboard performance. They simulated a group of Enron employees typing emails in chronological order. The simulated users provided sloppy touch input, but could make use of auto-correct and word predictions. They showed a background LM reduced typing WER from 38.4% to 5.7%. A personalized LM using a unigram cache further reduced WER to 4.6%.

We used the recently released dataset[1] that was used in Fowler et al. [3] as a basis for our experiments. We improve and extend this prior work in the following ways:

1. In addition to a unigram cache, we investigate adaptation via an adaptive n-gram model based on the Prediction by Partial Match (PPM) algorithm. We also investigate adaptation via a Recurrent Neural Network Language Model (RNNLM). To our knowledge, no work has shown the effectiveness of PPM for personalization on a touchscreen keyboard.
2. We use character n-gram LMs instead of the word bigram LM used by Fowler et al. [3]. This allows easier adaptation in the face of novel words. We tested a (roughly) similar 6-gram character LM, and also a longer-span 12-gram character LM. We find conditioning on more previous context substantially improves performance, but adaptation still provides further gains despite our stronger background baseline.
3. Fowler et al. [3] primed each user's model on their previous 30 days of email. We instead used 1520 words from each user for priming and 1520 words for evaluation (see section 3). In the released dataset, users' test set sizes varied from 3040 to 140K words (a factor of 3600). We opted for an experimental design comparing algorithms on different users but with the same amount of priming data. This allows for a tighter comparison between different adaptation algorithms. We believe this makes our results easier to interpret since our results represent performance across a large set of users under a consistent amount of priming and evaluation data. Further, we tested models with no priming text, simulating expected performance on, for example, a new device.
4. While Fowler et al. [3] showed word error rate results for noisy input, we also provide keystroke savings for noisy input. This helps us understand the degree adaptation might impact the effectiveness of a keyboard's word predictions.

3. Dataset and Background Models

Initially we hoped to compare more closely with Fowler et al. [3], but this was not possible for a number of reasons. The original dataset used in Fowler et al. [3] contained a development and test set with 45 users in each, but one user was dropped in the released dataset for privacy reasons. The language models, word vocabulary, and the scripts used to preprocess the raw text were also not released.

We created our own processed version of the raw text for the 44 users. We have released our processed text to facilitate

[1]https://github.com/google-research-datasets/EnronPersonalizationValidation

future research[2]. As in [3], we stripped non-alphabetical characters except apostrophe and converted to lowercase. We did not remove apostrophes as in [3] since contractions are common in emails and their removal could be problematic for future work using, for example, large pre-trained language models.

For each of the 45 test users, we limited each user's text to 3040 words (the amount of the user with the least text). We split this into two equal halves of 1520 words. The first half we denote as the *seed set* and the second half the *test set*. In our experiments, we simulated writing of the test set using either a *flat start* model that had no knowledge of the seed set, or a *primed* model that first updated its state based on the seed set.

Our background language model was the large version of the character 12-gram LM released by Vertanen and Kristensson [14]. This model was trained on 504 M words from blog posts, forum, and twitter, and has 40.9 M character n-grams. For comparison, Fowler et al. [3] used a word bigram having 8.5 M word n-grams. We also tested a 6-gram character LM which is roughly equivalent to a bigram word LM (since a word in English averages 5 characters). The 6-gram character LM was trained on the same data and had 3.6 M n-grams.

4. Experimental Setup and Models

We simulated touch input on a smartphone keyboard with word predictions that appear above the keyboard. An ideal user provided the touch input for each sentence character-by-character, observing the suggestion slots and selecting their desired word if it appears. The keyboard offered three suggestions, two word completion predictions based on the user's typing thus far, and the literal text typed (i.e. the letter sequence nearest to the simulated touch locations). Before any input for a word, the keyboard offered the three most probable words given the prior text.

Evaluation metrics. Our simulated user did not use the backspace key. Thus, in the case of noisy input, the keyboard might not offer a user's desired word. In such cases, we assume the user selected the most probable word. We measured how often this happened by reporting *word error rate (WER)* of the user's final text versus the reference text. Word predictions can speed writing by allowing a user to not type every character in a word. We measured this via *keystroke savings (KS)*. KS is the percentage reduction in keys pressed compared to letter-by-letter text entry. We assumed selecting a prediction required one keystroke and added any following space.

Keyboard. We used a QWERTY keyboard similar to [3] except we added an apostrophe key to the right of the `L` key. Keys had a width of 6.2 mm and a height of 9.4 mm. We assumed the user tapped the center of each desired key with optional Gaussian noise. We set the standard deviation of the noise to 2.0 mm and 1.9 mm in the x- and y-dimensions respectively. This was similar to the noise in [3] which was based on [15].

Decoder. To provide auto-correct and word predictions, we used the VelociTap decoder [16]. For the current touch sequence of a word, the decoder generates an n-best list of character sequence hypotheses according to a touch model and a character n-gram LM. The touch model is based on 2D Gaussians centered at each key. The decoder's free parameters were tuned on thousands of sentences of smartphone touch data as described in [17]. For complete details of the decoder, see [17].

We chose a character LM instead of a word LM to provide more granularity and efficient pruning during the decoder's search. Perhaps more importantly, in our problem of adapting to a user's writing, a character LM allows novel out-of-vocabulary words to be simply added to the word list used to produce word completion predictions. The background and the adaptive LMs are then immediately able to make sensible probability estimates for these newly added words based on any similar character patterns previously seen by the models. A word LM would have a harder time as it would start with no n-grams containing the novel word, so predictions (especially conditioned on prior text) would suffer from sparsity more than a character LM.

Word predictions. The decoder's best word prefix hypotheses are used to search for the most probable full words in a 168 K word list (similar in size to [3]). We used the most frequent words in billions of words of web text that also appeared in a large human-edited dictionary. Our word list is provided in our released dataset. During a user's simulation, any out-of-vocabulary word typed was added to the vocabulary for the remainder of that user's simulation. This allowed the new word to be suggested by the keyboard from that point on.

The decoder generated a list of the 100 most probable word predictions based on a user's previously written text. We optionally rescored this list with one or more adaptive models. We opted for this rescoring approach since using an LM such as an RNNLM directly in the decoder's search can be expensive [18]. The relative contribution of each adaptive model and the background model was controlled by a set of mixture weights.

We tested three adaptive models:

1. **Unigram cache.** This method uses the frequency of words encountered thus far to estimate a unigram LM [4]. Given the small size of our test set, we did not limit the size of the cache. We also did not use an exponentially decaying cache as it performed similar to a unigram cache in [3].
2. **Prediction by partial matching (PPM).** PPM was originally designed as a text compression algorithm [19]. It is similar to a smoothed n-gram model [20] but is more amendable to updating during use due to the model's storage of explicit counts. We adapted the PPM implementation from Google[3] which was implemented following the algorithm used in Dasher [7]. We tested PPM orders of 6 and 12. Note that a PPM model of order n looks at the n previous symbols, unlike an n-gram that looks at the previous $n - 1$ symbols.
3. **Recurrent neural network language model.** We incorporated an RNNLM to test whether the previous adaptive models provided gains over a strong baseline; ensembling an n-gram LM and an RNNLM has been shown to perform better than either in isolation [21]. We also wanted to see if the hidden state of the RNNLM could help adapt to previous text (i.e. without updating parameters via backpropagation).

We trained a character RNNLM [22] with LSTM units on the same text as the background LM. We optimized hyperparameters via a grid search optimizing for perplexity. We searched over: character embedding size [32, 64], LSTM units [128, 256, 384, 512], hidden layers [1, 2], learning rate [0.1, 0.01, 0.001], and dropout rate [0.5, 0.55, 0.6]. The final RNNLM had an embedding size of 64, 512 LSTMs, one hidden layer, and a dropout rate of 0.55. We used the Adam optimizer.

Simulation mixture weights. We conducted two experiments. In the first experiment, we simulated typing without any spatial noise. In the second experiment, we added spatial noise to the simulated touches. For both experiments, we optimized different mixture weights to combine the LM probabilities from

[2]`https://osf.io/45p3j/`

[3]`https://github.com/google-research/google-research/tree/master/jslm`

Table 1: *Mixture weights of different model combinations. BG denotes the background 12-gram character LM. Weights appear in the same order as the components in each ensemble's name.*

Model	Weights
BG + Unigram	0.50, 0.50
BG + RNNLM	0.45, 0.55
BG + PPM-12	0.75, 0.25
BG + Unigram + RNNLM	0.30, 0.30, 0.40
BG + Unigram + PPM-12	0.75, 0.05, 0.20
BG + PPM-12 + RNNLM	0.40, 0.20, 0.40
BG + Unigram + PPM-12 + RNNLM	0.30, 0.25, 0.15, 0.30

Table 2: *Model performance (± 95% CI) in Experiment 1 on simulated touch data without added spatial noise.*

	Flat start	Primed
Model	KS (%)	KS (%)
Background (6-gram)	39.3 ± 0.3	
Background (12-gram)	45.6 ± 0.3	
+ Unigram	46.3 ± 0.3	46.7 ± 0.3
+ RNNLM	46.8 ± 0.3	46.9 ± 0.3
+ PPM-6	47.4 ± 0.3	48.2 ± 0.3
+ PPM-12	47.7 ± 0.3	48.6 ± 0.3
+ Unigram + RNNLM	47.3 ± 0.3	47.6 ± 0.3
+ Unigram + PPM-12	47.7 ± 0.3	48.6 ± 0.3
+ PPM-12 + RNNLM	48.7 ± 0.3	49.5 ± 0.3
+ Unigram + PPM-12 + RNNLM	48.7 ± 0.3	49.4 ± 0.3

the adaptive models and the background 12-gram character LM. In this case, we optimized with respect to keystroke savings on six users' data from the development set in Fowler. Table 1 shows the tuned values we found for each model ensemble.

Optimizing the mixture weights using the text from all the 45 users in the development set was too computationally expensive especially for the models with multiple mixture weights. While it is possible slightly better mixture weights might have been found tuning on all 45 users, in our tests with one configuration (background + PPM-12), we found optimizing on 12 users resulted in very similar mixture weights and keystroke savings compared to optimizing on only six users.

5. Results

Experiment 1. We first simulated typing without any spatial noise. Table 2 shows results of our two background LMs and all combinations of rescoring with the adaptive models. Keystroke savings using just the 6-gram LM was 39.3% but was substantially higher at 45.6% using the 12-gram LM. Comparing the left and right columns, we see that all models benefited from priming on a user's data prior to evaluation. However, priming only afforded the RNNLM a small improvement of 0.1%.

In isolation, all adaptive models improved keystroke savings with PPM-12 providing the biggest gain of 3.0% in the primed condition. Our unigram cache did not provide as big an improvement as in [3]. We suspect this is due to the 12-gram LM conditioning on more previous text. Combining models provided additive gains with the exception of combining the unigram cache with PPM-12. We think the ability to condition on previous words and better smoothing of PPM-12 allows it to subsume the functionality of the unigram cache. Our best model PPM-12 + RNNLM provided a keystroke savings increase of

Table 3: *Model performance (± 95% CI) in Experiment 2. Models were primed and spatial noise was added.*

Model	KS (%)	WER (%)
Background (6-gram)	37.3 ± 0.3	2.02 ± 0.08
Background (12-gram)	43.6 ± 0.3	2.17 ± 0.09
+ Unigram	45.3 ± 0.5	1.66 ± 0.06
+ RNNLM	45.2 ± 0.3	1.91 ± 0.08
+ PPM-6	46.5 ± 0.3	1.37 ± 0.06
+ PPM-12	46.9 ± 0.3	1.36 ± 0.06
+ Unigram + RNNLM	46.4 ± 0.3	1.66 ± 0.07
+ Unigram + PPM-12	47.0 ± 0.3	1.41 ± 0.05
+ PPM-12 + RNNLM	47.9 ± 0.3	1.39 ± 0.06
+ Unigram + PPM-12 + RNNLM	47.8 ± 0.3	1.40 ± 0.06

3.9% on the primed data compared to the background 12-gram.

Experiment 2. Next, we added spatial noise to the simulated touches. Table 3 shows the results for the primed models. Comparing Table 3 to Table 2, we see keystroke savings dropped around 2% due to the noise. But similar to noiseless input, keystroke savings improved when one or more adaptive models was used. As without noise, PPM-12 provided the biggest gain. Combining PPM-12 with the RNNLM improved performance, but as with no noise the unigram cache provided no gain in conjunction with PPM-12.

Our best combination of PPM-12 + RNNLM improved keystroke savings by 4.3% and reduced the WER of final text by 0.78% compared to the background 12-gram. Without an LM, the WER on the noisy touch data was 39.3% (±0.27% 95% CI). Thus, it appears that the gains of the adaptive models are robust to the typical noise inherent in touchscreen keyboard input.

We compared the per-user performance of the background model and the ensemble of all three models with and without priming (Figure 1). For all users, the primed ensemble had a higher keystroke savings. For 37 out of the 44 users, the ensemble without priming outperformed the background LM.

Adding in PPM-12 did increase computation and memory requirements. For the 12-gram baseline model, per word decode time was 180 ms versus 270 ms for the 12-gram + PPM-12 model. The 12-gram baseline consumed 8.8 GB of memory versus 9.5 GB for the 12-gram + PPM-12 model. The maximum memory consumption of the other models and the combinations ranged from 8.8 GB to 10.6 GB with the 12-gram background model being responsible for most of the memory footprint.

6. Discussion and Limitations

In this work, we showed how three adaptive models – namely, a unigram cache, PPM, and an RNNLM – can be used to provide better word predictions than a static background LM. While previous work has investigated a unigram cache for LM adaptation with or without introduced spatial noise, models like PPM and RNNLM have not been tested on such input. We found combining PPM and an RNNLM outperformed a unigram cache and should be considered for use in touchscreen keyboards.

A PPM model can be queried and its counts can be updated computationally efficiently. Our results suggest that even devices with modest hardware resources can use PPM to improve word predictions. We found even a fairly low order PPM model can be used without losing significant performance. For example, with spatial noise and priming, adaptation using PPM-6 reduced keystroke savings by only 0.4% compared to PPM-12. From our observation, the adaptation process shined with out-

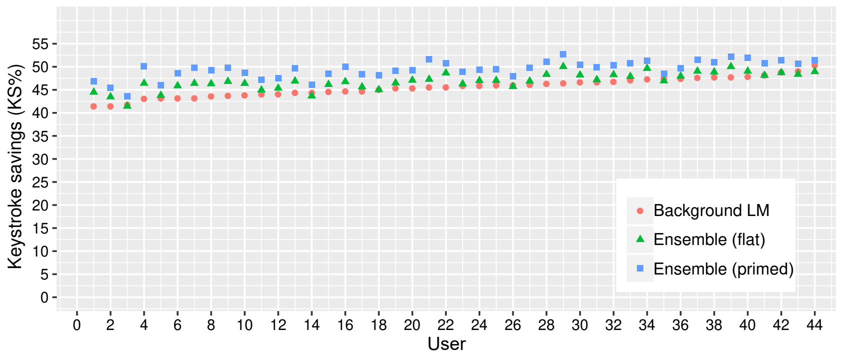

Figure 1: *The keystroke savings on noisy touch data for each user in three different settings. Ensemble (flat) represents the combination of background LM, unigram cache, RNNLM, and PPM-12 with a flat start. Ensemble (primed) represents when the models were primed with the seed text. Users are sorted according to the keystroke savings using the 12-gram background LM.*

of-vocabulary words. Although the unigram cache kept track of new unseen words, PPM-12 worked better than the unigram cache which only considers how often a word has been seen but not in the contexts in which the word commonly occurs (e.g. predicting a user's name after "Best regards,").

We tested our models on text written by Enron employees that were most likely typed on desktop keyboards. The style of text written on touchscreen keyboards may differ to some degree. The text may also contain typos or spelling errors that may not reflect a user's true intent. It would be interesting to validate our finding on text written on actual mobile devices. However, sourcing such data would be difficult for privacy reasons, and to our knowledge no such dataset exists. We believe the Enron personalization dataset by Fowler et al. [3] is currently the best available choice.

Frequently, a touchscreen keyboard user may touch neighboring keys instead of their intended key. We simulated this aspect by adding Gaussian noise to the target key's center. A higher fidelity simulation might sample touch locations from observations from actual touchscreen typing. A more advanced simulation might also insert extra erroneous taps or delete taps for some letters to simulate these events which do sometimes occur in real-world touchscreen input.

We tested our models by simulating the writing of around 3000 words. The behavior of the models for substantially more text needs further investigation. For example, models such as PPM may require pruning as more and more text are fed to it. This might be necessary to maintain predictive performance if a user's writing style changes over time. Pruning might also be necessary to control the memory required by the model.

We tested an RNNLM using persistence in its hidden states to model a user's previous writing. Judging by the difference between our primed and flat start variants, this did not appear to provide strong adaptation ability. It might be more advantageous to periodically update the network's parameters via backpropagating based on a user's recently written text or by fine-tuning a large pre-trained language model (e.g. GPT-2 [23]). However on mobile devices, such large language models may be too expensive to use on-device (as might be required for privacy). Additionally, such models can suffer from latency issues during inference [8]. Our work here has shown that PPM can offer a simple, compute- and memory-efficient approach to provide effective language model personalization.

It would be interesting to compare our simulated touchscreen keyboard against an off-the-shelf keyboard such as the iPhone keyboard or Android Gboard. But this would be challenging for a number of reasons. First, the language models these keyboards use are not public. We do not know, and probably cannot control, their vocab size, language model type, or data they were trained on. Second, to use these keyboards in experiments would require programmatic access to the underlying engine, including the ability to reset any adaptation algorithm.

Our results are based on observing the improvements in keystroke savings and word error rate of an ideal user who makes perfect use of a keyboard's predictions. Further, the simulated user made no use of other correction features (e.g. backspace). Whether gains would translate to real-world text entry will require a longitudinal user study in which each participant writes a substantial amount of text.

7. Conclusion

In this paper, we made use of a newly released data set to simulate the entry of a large number of email messages by a set of 44 users on a touchscreen keyboard. We simulated noise indicative of typing on a mobile device. We showed the effects of various adaptive language models on correcting an ideal user's noisy input and personalizing the models based on the user's past writing pattern. To facilitate future research, we have made our text data, word list, and language models available.

Our results show personalization can improve touchscreen keyboard performance. With noiseless touch input, the best personalized model achieved a 8.6% relative increase in keystroke savings compared to using only a static 12-gram character LM. With noisy touch input, we achieved a slightly higher 9.9% relative increase in keystroke savings. Further, we found personalization reduced how often a typical keyboard with three prediction slots will fail to provide a user's intended text. The final word error rate of the simulated user was reduced by 36% relative on noisy touchscreen input. In a real keyboard interface, this increase in keystroke savings and lower prediction failure rate should help users write faster without requiring that a user resort to backspacing or other correction features.

8. Acknowledgements

This material is based upon work supported by the NSF under Grant No. IIS-1750193. We thank Brian Roark and Shumin Zhai from Google for their work on releasing the Enron personalization dataset.

9. References

[1] K. Tanaka-ishii, "Word-Based Predictive Text Entry Using Adaptive Language Models," *Natural Language Engineering*, vol. 13, no. 1, p. 51–74, Mar. 2007. [Online]. Available: https://doi.org/10.1017/S1351324905004080

[2] P. R. Clarkson and A. J. Robinson, "Language Model Adaptation Using Mixtures and an Exponentially Decaying Cache," in *IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 2, 1997, pp. 799–802.

[3] A. Fowler, K. Partridge, C. Chelba, X. Bi, T. Ouyang, and S. Zhai, "Effects of language modeling and its personalization on touchscreen typing performance," in *Proceedings of the SIGCHI Conference on Human factors in Computing Systems*, ser. CHI '15. New York, NY, USA: ACM, 2015, pp. 649–658. [Online]. Available: http://doi.acm.org/10.1145/2702123.2702503

[4] R. Kuhn and R. De Mori, "Cache-based natural language model for speech recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 12, pp. 570–583, 07 1990.

[5] R. Kuhn, "Speech Recognition and the Frequency of Recently Used Words: A Modified Markov Model for Natural Language," in *Proceedings of the 12th Conference on Computational Linguistics - Volume 1*, ser. COLING '88. USA: Association for Computational Linguistics, 1988, p. 348–350. [Online]. Available: https://doi.org/10.3115/991635.991706

[6] T. C. Bell, I. H. Witten, and J. G. Cleary, *Text Compression*. Englewood Cliffs, NJ, USA: Prentice Hall, 1990. [Online]. Available: https://nla.gov.au/nla.cat-vn1413295

[7] D. J. Ward, A. F. Blackwell, and D. J. C. MacKay, "Dasher – a Data Entry Interface Using Continuous Gestures and Language Models," in *Proceedings of the 13th Annual ACM Symposium on User Interface Software and Technology*, ser. UIST '00. New York, NY, USA: Association for Computing Machinery, 2000, p. 129–137. [Online]. Available: https://doi.org/10.1145/354401.354427

[8] M. X. Chen, B. N. Lee, G. Bansal, Y. Cao, S. Zhang, J. Lu, J. Tsay, Y. Wang, A. M. Dai, Z. Chen, T. Sohn, and Y. Wu, *Gmail Smart Compose: Real-Time Assisted Writing*. New York, NY, USA: Association for Computing Machinery, 2019, p. 2287–2295. [Online]. Available: https://doi.org/10.1145/3292500.3330723

[9] M. King and P. Cook, "Evaluating Approaches to Personalizing Language Models," in *Proceedings of the 12th Language Resources and Evaluation Conference*. Marseille, France: European Language Resources Association, May 2020, pp. 2461–2469. [Online]. Available: https://aclanthology.org/2020.lrec-1.299

[10] E. Grave, A. Joulin, and N. Usunier, "Improving Neural Language Models with a Continuous Cache," in *5th International Conference on Learning Representations, ICLR 2017, April 24-26, 2017*. OpenReview.net, 2017. [Online]. Available: https://openreview.net/forum?id=B184E5qee

[11] K. Li, H. Xu, Y. Wang, D. Povey, and S. Khudanpur, "Recurrent Neural Network Language Model Adaptation for Conversational Speech Recognition," in *Proceedings of Interspeech 2018*, 2018, pp. 3373–3377. [Online]. Available: http://dx.doi.org/10.21437/Interspeech.2018-1413

[12] E. Grave, M. M. Cisse, and A. Joulin, "Unbounded cache model for online language modeling with open vocabulary," *Advances in Neural Information Processing Systems*, 2017. [Online]. Available: https://arxiv.org/abs/1711.02604

[13] K. Li, Z. Liu, T. He, H. Huang, F. Peng, D. Povey, and S. Khudanpur, "An Empirical Study of Transformer-Based Neural Language Model Adaptation," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2020)*, 2020, pp. 7934–7938.

[14] K. Vertanen and P. O. Kristensson, "Mining, Analyzing, and Modeling Text Written on Mobile Devices," *Natural Language Engineering*, vol. 27, pp. 1–33, 2021.

[15] S. Azenkot and S. Zhai, "Touch behavior with different postures on soft smartphone keyboards," in *Proceedings of the 14th International Conference on Human-Computer Interaction with Mobile Devices and Services*, ser. MobileHCI '12. New York, NY, USA: Association for Computing Machinery, 2012, p. 251–260. [Online]. Available: https://doi.org/10.1145/2371574.2371612

[16] K. Vertanen, H. Memmi, J. Emge, S. Reyal, and P. O. Kristensson, "VelociTap: Investigating Fast Mobile Text Entry Using Sentence-Based Decoding of Touchscreen Keyboard Input," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '15. New York, NY, USA: ACM, 2015, pp. 659–668. [Online]. Available: http://doi.acm.org/10.1145/2702123.2702135

[17] K. Vertanen, *Probabilistic Text Entry-Case Study 3*, 1st ed. New York, NY, USA: Association for Computing Machinery, 2021, pp. 277–320. [Online]. Available: https://doi.org/10.1145/3447404.3447420

[18] J. Adhikary, J. Berger, and K. Vertanen, "Accelerating Text Communication via Abbreviated Sentence Input," in *Proceedings of the Joint Conference of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing*, 2021, pp. 6574–6588.

[19] J. Cleary and I. Witten, "Data Compression Using Adaptive Coding and Partial String Matching," *IEEE Transactions on Communications*, vol. 32, no. 4, pp. 396–402, 1984.

[20] P. J. Cowans, "Probabilistic document modelling," Ph.D. dissertation, University of Cambridge, 2006.

[21] T. Mikolov, A. Deoras, S. Kombrink, L. Burget, and J. Cernockỳ, "Empirical evaluation and combination of advanced language modeling techniques," in *Proceedings of the International Conference on Spoken Language Processing*, 2011, pp. 605–608.

[22] T. Mikolov, M. Karafiát, L. Burget, J. Cernockỳ, and S. Khudanpur, "Recurrent Neural Network based Language Model," in *Proceedings of the International Conference on Spoken Language Processing*, vol. 2, 2010, pp. 1045–1048.

[23] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," 2018. [Online]. Available: https://d4mucfpksywv.cloudfront.net/better-language-models/language-models.pdf