2022

Intelligent Techniques to Accelerate Everyday Text Communication

Jiban Krishna Adhikary
Michigan Technological University, jiban@mtu.edu

INTELLIGENT TECHNIQUES TO ACCELERATE EVERYDAY TEXT COMMUNICATION

By
Jiban Krishna Adhikary

a dissertation

Submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

In Computer Science

MICHIGAN TECHNOLOGICAL UNIVERSITY
2022

This dissertation has been approved in partial fulfillment of the requirements for the Degree of Doctor of Philosophy in Computer Science.

Department of Computer Science

Dissertation Advisor:	*Dr. Keith Vertanen*
Committee Member:	*Dr. Scott Kuhl*
Committee Member:	*Dr. Laura Brown*
Committee Member:	*Dr. Elizabeth Veinott*
Department Chair:	*Dr. Linda Ott*

DEDICATED TO MY PARENTS

Contents

Listing of figures

Listing of tables

Acknowledgments

At first I want to express my sincere gratitude to my advisor, Keith, for his constant support, inspiration, sharp insight, and meticulous attention to details. He has been always been an excellent support and source of new ideas throughout this work.

I want to express my profound appreciation to my committee members, Dr. Scott Kuhl, Dr. Laura Brown, and Dr. Elizabeth Veinott for their time and energy in improving this work.

I want to acknowledge the Computer Science Department at Michigan Tech, the National Science Foundation, and the Michigan Tech graduate school for providing financial support during my stay at Michigan Tech.

Lastly, I am grateful to my family members and to my wife Sristy. Sristy has been my strength and source of inspiration during the difficult days of COVID-19 pandemic. This work would not have been possible if she was not there for me.

Abstract

People with some form of speech- or motor-impairments usually use a high-tech augmentative and alternative communication (AAC) device to communicate with other people in writing or in face-to-face conversations. Their text entry rate on these devices is slow due to their motor abilities. Making good letter or word predictions can help accelerate the communication of such users. In this dissertation, we investigated several approaches to accelerate input for AAC users. First, considering that an AAC user is participating in a face-to-face conversation, we investigated whether performing speech recognition on the speaking-side can improve next word predictions. We compared the accuracy of three plausible microphone deployment options and the accuracy of two commercial speech recognition engines. We found that despite recognition word error rates of 7–16%, our ensemble of n-gram and recurrent neural network language models made predictions nearly as good as when they used the reference transcripts. In a user study with 160 participants, we also found that increasing number of prediction slots in a keyboard interface does not necessarily correlate to improved performance.

Second, typing every character in a text message may require an AAC user more time or effort than strictly necessary. Skipping spaces or other characters may be able to speed input and reduce an AAC user's physical input effort. We designed a recognizer optimized for expanding noisy abbreviated input where users often omitted spaces and mid-word vowels. We showed using neural language models for selecting conversational-style training text and for rescoring the recognizer's n-best sentences improved accuracy. We found accurate abbreviated input was possible even if a third of characters was omitted. In a study where users had to dwell for a second on each key, we found sentence abbreviated input was competitive with a conventional keyboard with word predictions.

Finally, AAC keyboards rely on language modeling to auto-correct noisy typing and to offer word predictions. While today language models can be trained on huge amounts of text, pre-trained models may fail to capture the unique writing style and vocabulary of individual users. We demonstrated 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. Our best model ensemble increased keystroke savings by 9.6%.

1

Introduction

1.1 Motivation

The characteristic that separates human beings from other species is the ability to communicate in explicit ways, for example, by speaking or in writing. Other species can also communicate via sounds or making noise, but their communication is not as advanced as human beings. One of the most important ways we communicate is via face-to-face con-

versations. It is also one of the most effective ways of acquiring knowledge. But sometimes this means of communication is hampered when a person cannot talk due to a motor- or speech-impairment. People having a speaking disorder may use some form of Augmentative and Alternative Communication (AAC) [1]. For example, an AAC user may use a low-tech aid such as a paper notebook to communicate or they may use a high-tech tablet computer that speaks for them.

High-tech AAC devices let users communicate by entering text using an AAC interface. However, the rate at which they enter text on an interface is usually very low often less than 10 words-per-minute [45, 60, 64, 96, 107, 114]. These AAC devices typically make use of a language model (to be discussed in Chapter 2) to suggest word predictions. They also use only the AAC user's side of a conversation for predicting words. However, real life conversations are two-sided spoken dialogues where one side affects the responses of the other side. Intuitively, for better suggestions it makes sense to model both sides of a conversation. We believe further gain on an AAC user's response can be achieved by using the speech of the communication partner as input context. In order to use partner speech as context, we can perform automatic speech recognition on the partner speech to convert it to text. Then we can use this result as a context to a language model which is trained on two-sided conversations.

Due to an AAC user's motor ability, it might be difficult for them to enter every character of every word in a sentence during writing. For example, their input might be slow or it might be physically tiring for them to provide the input. One way to reduce their workload could be to reduce the number of characters they enter. In other words, they could enter some characters of the words but not all the characters in the words.

AAC users can also benefit from the text they have already written using their device. AAC devices normally use a static language model to suggest word predictions to the user. The model predicts the next word for a given context but the model does not get updated as new phrases are written on the device. For example, the probability of a next word for a given context does not change even if the same word appears after the context again and again in the future. Normally people tend to repeat words and phrases during writing. One way to provide better suggestions to an AAC interface would be to make use of a user's written history and adapt the language model with time.

In this dissertation, we will consider the above solutions and their advantages and disadvantages to design text entry interfaces for AAC users. We will examine if these interfaces help able-bodied users accelerate their input in a simulated rate-limited text entry setting. While evaluating a text entry interface in the upcoming chapters, we did not involve actual AAC users. Since we did not know if our proposed methods would work or not, we did not want AAC users to directly test an unproven interface. Instead, we performed simulation experiments and conducted user studies with able-bodied people with simulated rate-limited text input.

1.2 Overview and Contributions

Chapter 2 introduces some text entry interfaces for AAC users and what techniques we use to provide auto-correction and word suggestions. We discuss how text entry interfaces make use of statistical language models. Besides statistical language models, we describe more advanced language models such as Recurrent Neural Network Language Models (RNNLMs) and transformer based models. Then we discuss different metrics that are used

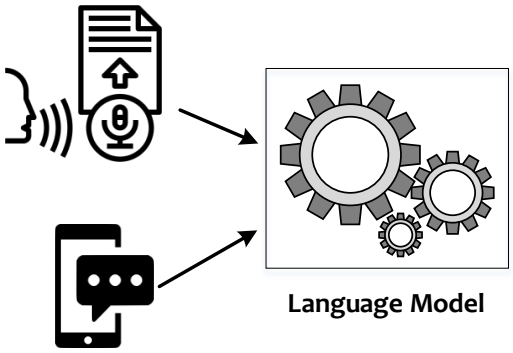

Figure 1.1: Using partner speech recognition results as context to improve next word prediction.

to evaluate a text entry system.

Below is a short summary and a list of the highlights of the remaining chapters:

Chapter 3: Investigating Speech Recognition for Improving Predictive AAC

In this chapter, we collect speech data and compare recognition accuracy of different plausible microphone setups on an AAC device. We also compare recognition accuracy of two commercial automatic speech recognition engines. Then we discuss training language models and report how these models perform with different test sets. Finally, we conduct offline experiments to show how speech context can be used to enhance face-to-face communication. In this case, we assume a user enters text via an AAC device and the communication partner speaks and we perform speech recognition on the partner's speech (Figure 1.1). The main takeaways from this chapter are:

- Speech can be recorded and accurately recognized using various microphones in different locations.
- A speaking partner's recognized text can be successfully used as context to improve AAC

user's next word predictions.

- Predictions made with partner speech recognition results as context are nearly as good as using reference transcripts.

Chapter 4: Dwell-based Text Entry and Partner Speech Context

In this chapter, we describe a crowdsourced user study simulating AAC-user like interaction by able-bodied people on a dwell keyboard. On a dwell keyboard a user dwells over a key for a certain period of time to activate it. Dwelling on a key can be done using a mouse pointer or eye-gaze. In this chapter, first we try to find what number of suggestion slots is appropriate for a dwell keyboard. We use a dwell time of 1000 ms and have users dwell on a key using a mouse pointer. From empirical interaction data, we also find the average time a user spends to actuate a key. Then we investigate via offline experiments if using partner speech recognition results as context to a language model and using different number of slots enhance user performance by reducing the number of keystrokes. Based on the estimated time to actuate a tap from the first study, we also calculate estimated entry rates with different number of slots. The key takeaways from this chapter are:

- Using a 1000 ms dwell time and 2–9 slots, users can enter text at 9–11 words-per-minute.
- On an average users spent 2298 ms per key tap during dwell typing.
- With partner speech context and five suggestion slots, a simulated user's keystroke savings improved from 56.9% to 58.1%.

Chapter 5: Accelerating Text Communication via Abbreviated Sentence Input

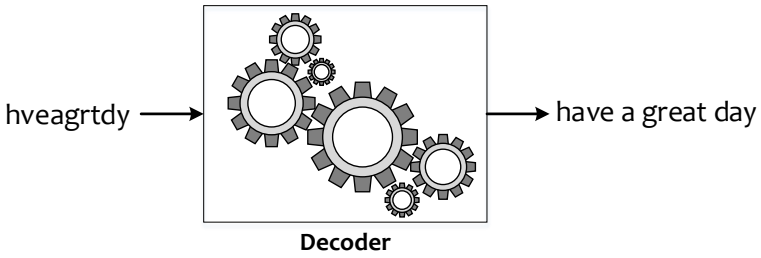

Figure 1.2: A system taking an abbreviated sentence as input and expanding it to the intended text.

In this chapter, we investigate input of conversational text by removing some letters of a sentence. To expand such input we need good language models and for good language models we require large amount of text data representing everyday conversations. Therefore, we propose an intelligent data selection technique to select conversational text using a neural model called BERT [29]. By conducting offline experiments and by conducting a crowdsourced user-study, we show that simulated rate-limited users can effectively input text by removing mid-word vowels and spaces between words. The key findings from this chapter are:

- A neural model e. g. BERT can be used to select target data from an out-of-domain data source and the quality of sampled data is slightly better than data sampled with existing approach.
- Theoretically, 38.2% characters can be saved during a text input process by not entering spaces between words and mid-word vowels.
- After practice, users entered text at 9.6 words-per-minute when they skipped mid-word vowels and spaces between words. When users entered text word-by-word and used word predictions, the entry rate was 9.9 words-per-minute.

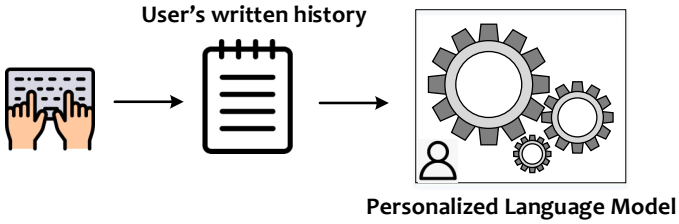

Figure 1.3: Personalizing a language model with a user's written history may provide good word predictions.

Chapter 6: Language Model Personalization

We examine different personalization techniques to enhance the quality of a user's next word predictions. We use a publicly available dataset containing a set of users' chronological written text. We conduct experiments with these users' text data and examine the performance of three different adaptive language models. The key results from this chapter are:

- An ensemble of a n-gram language model, a unigram cache, prediction by partial matching (PPM), and a recurrent neural network language model provides improved performance compared to using only an n-gram language model.
- With noiseless touch input the best personalized model achieved a 8.3% increase in keystrokes savings compared to a n-gram language model.
- With noisy input the best model achieved a 9.6% increase in keystroke savings compared to using only a n-gram character language model.

1.3 Relationship to Previous Publications

Chapter 3 and Chapter 5 have been previously published and some passages have been quoted verbatim from the following publications:

(i) **Investigating Speech Recognition for Improving Predictive AAC**. Jiban Adhikary, Robbie Watling, Crystal Fletcher, Alex Stanage, and Keith Vertanen. In Proceedings of the Workshop on Speech and Language Processing for Assistive Technologies (SLPAT 2019).

(ii) **Accelerating Text Communication via Abbreviated Sentence Input**. Jiban Adhikary, Jamie Berger, and Keith Vertanen. In the Joint Conference of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (ACL-IJCNLP 2021).

2

Overview of Text Entry Interfaces and Language Modeling

Entering text is a ubiquitous task. We use different text entry interfaces in our day-to-day life. In this chapter, first we briefly talk about different text entry interfaces. Then we will unravel different mechanisms behind a typical text entry system. This will not only provide

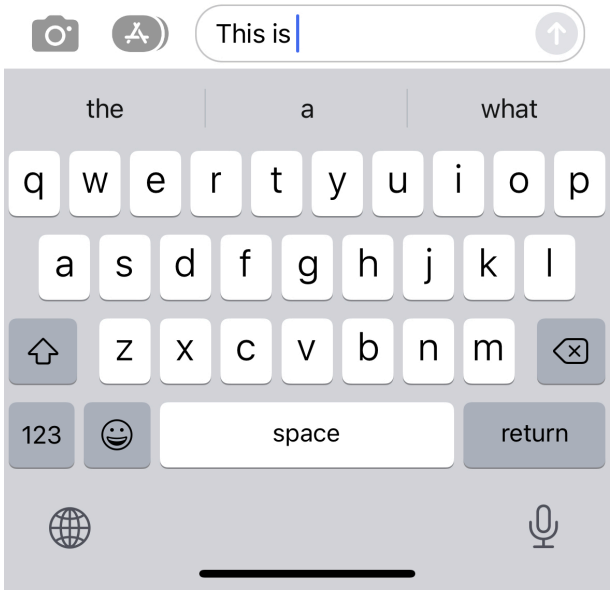

Figure 2.1: A smartphone keyboard. There are three prediction slots above the keyboard. The user has already entered `This is`. Considering the entered phrase as a context the keyboard has generated the most probable three next words and offered them in the suggestion slots.

the reader a general understanding about text entry research but also provide useful insight while reading the upcoming chapters.

2.1 Text Entry Interfaces

We can broadly categorize the text entry interfaces into two types based on the type of user:

2.1.1 Mainstream Text Entry Interfaces

This type of interfaces include different physical and virtual keyboards that normally people use to enter text. For example, desktop keyboards, smartphone keyboards (Figure 2.1),

HELLO |

A	B	C	D	E	F
G	H	I	J	K	L
M	N	O	P	Q	R
S	T	U	V	W	X
Y	Z	ENTER	SPACE	,	.

Figure 2.2: A row-column scanning keyboard. If a user wants to select a letter from A–F, the user will press a switch when the first row is highlighted.

and smartwatch keyboards. Touchscreen keyboards on smartphones or smartwatches normally provide auto-corrections and word predictions above the keyboard.

2.1.2 Text Entry Interfaces for AAC Users

Due to an AAC user's motor abilities, they normally do not use a mainstream keyboard interface. Literate AAC users use a variety of techniques to provide input on a keyboard. Input can be via a touchscreen [31, 77, 102], eye-tracker [30, 130], physical switches [7, 72, 112], or brain activity [55, 89].

A very common AAC user interface is a row-column scanning keyboard with a switch [27, 69, 73, 100]. In this type of keyboard, there are a number of rows and each row contains a set of letters. Each row is highlighted cyclically and a user uses a switch to select a row containing the intended letter. Then the keyboard cyclically highlights every column in the selected row and the user can further pinpoint the desired letter in that row using the switch. Figure 2.2 shows a row-column scanning keyboard.

Figure 2.3 shows the Dasher [130] interface for AAC users. In Dasher, a stream of letters

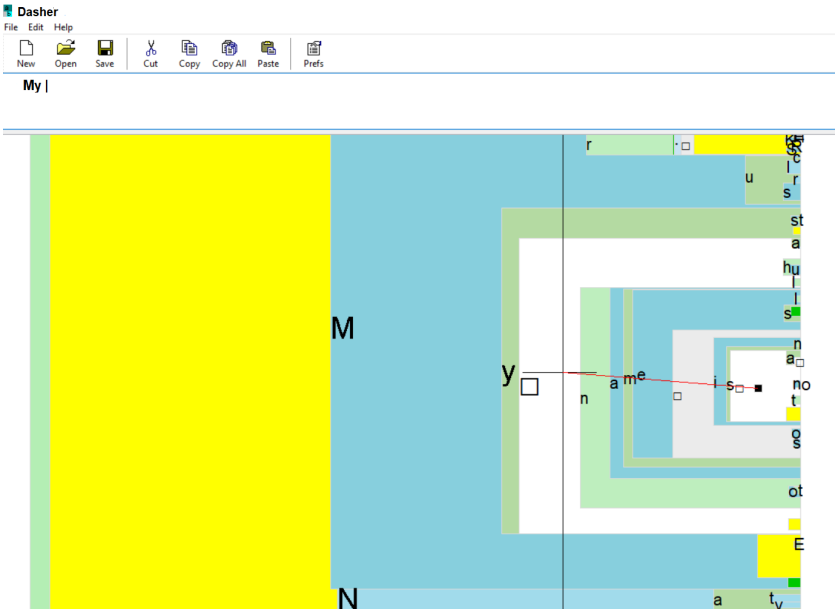

Figure 2.3: The Dasher [130] interface. The user has currently written "My ". The red line shows the direction a user would point in order to write "My name is ".

.

are shown to be coming towards the screen. Then the user steers the interface through the path of the letter sequence they want. In this case, a mouse pointer [129], eye-gaze [115], and even brain waves [135] can be used. Dasher uses a probabilistic model to predict the likely character combinations for the next piece of text. If a character or combination of characters is more probable, then it displays that character or combination of characters more prominently in the stream.

Figure 2.4 shows an eye-gaze keyboard interface. In this type of keyboard, a user's eye movement is tracked using an eye tracker. A user needs to look at a key for a certain amount of time to click it. Dwell time can vary but can be adjusted [74, 75, 86, 98, 99].

Figure 2.4: An eye-gaze keyboard interface [2].

2.2 Language Modeling

Language modeling is the process of determining the probability of a next word given a sequence of words. Language modeling is an integral part of speech recognition, machine translation, natural language generation, parts of speech tagging, information retrieval, and other applications. Given a word sequence $w_1 \ldots w_i$, the goal of a language model is to determine the probability $P(w_1 \ldots w_i)$. Predictive text entry interfaces normally show word predictions above the keyboard when a user is typing text. These word predictions are made by conditioning on the text the user has already entered. We use language modeling to offer the most probable word predictions in an interface.

2.2.1 N-gram Model

An n-gram is a sequence of n symbols where a symbol could be a character or a word. A 2-gram or *bigram* is a two-word sequence like `the quick`, `quick brown`, or `brown fox`, and a 3-gram or *trigram* is a three-word sequence like `the quick brown` or `quick brown fox`.

Suppose we are given the task to compute the probability of a word w given a word sequence or history h. If the history h is `the quick brown fox jumps` and the word w is `over`, then we can estimate the probability from the frequency counts of w and h given a large corpus:

$$P(over|the\ quick\ brown\ fox\ jumps) = \frac{C(the\ quick\ brown\ fox\ jumps\ over)}{C(the\ quick\ brown\ fox\ jumps)} \tag{2.1}$$

where C represents the count of a phrase.

Let us assume that we have a sequence of i words, $w_1^i = w_1 \ldots w_i$. We can determine the probability of the entire sequence by the chain rule of probability:

$$P(w_1^i) = P(w_1)P(w_2|w_1)P(w_3|w_1^2)\ldots P(w_i|w_1^{i-1}) \tag{2.2}$$

$$= \prod_{j=1}^{i} P(w_j|w_1^{j-1}). \tag{2.3}$$

The intuition behind n-gram modeling is that instead of computing the probability of a word given its entire history, we can approximate the history by just the last few words. For example, for a bigram model, we can approximate the probability of a word given all previous words, $P(w_i|w_1^{i-1})$ by calculating the conditional probability just of the preceding word

$P(w_i|w_{i-1})$. For example, in Equation 2.1, we can approximate the probability *P(over|the quick brown fox jumps)* as *P(over|jumps)*.

For a trigram language model, we approximate the conditional probability using only the preceding two words. For 4-gram, we approximate the conditional probability using the preceding 3 words, and so on. The general equation for the n-gram approximation is:

$$P(w_i|w_1^{i-1}) \approx P(w_i|w_{i-n+1}^{i-1}). \tag{2.4}$$

The simplest way to estimate these n-gram probabilities is to use Maximum Likelihood Estimation. For example, in case of a bigram model, given a word w_i and its preceding word w_{i-1}, we have:

$$P(w_i|w_{i-1}) = \frac{C(w_{i-1}w_i)}{\sum_w C(w_{i-1}w)}. \tag{2.5}$$

Since, the sum of all bigrams that start with a given word w_{i-1} must be equal to the unigram count for that word w_{i-1}, we get:

$$P(w_i|w_{i-1}) = \frac{C(w_{i-1}w_i)}{C(w_{i-1})}. \tag{2.6}$$

Smoothing. Usually we use a text corpus called a training set to count different n-grams and to generate a list of words or the vocabulary. Sometimes it may happen that we have some words in the vocabulary which appear in some different text corpus in an unseen context. For example, they may appear after a word they never appeared after in the training set. For such cases, to avoid assigning zero probabilities, we shave off a bit probability

mass from the frequent events and give it to the infrequent events. This process is called discounting or smoothing. Some smoothing functions include Laplace smoothing (add-1 smoothing), add-k smoothing, Stupid backoff [12], and Kneser-Ney smoothing [59].

Laplace smoothing or add-one smoothing adds one to each count. Add-k smoothing adds fractional count $k(0.5, 0.05, 0.1)$ instead of adding 1 to each count. A related term to smoothing is discounting where we discount (i.e. lower) some non-zero probability mass and give this mass to zero counts. In backoff, if we cannot estimate the probability using an n-gram we back off to lower-order n-grams. For example, we can use a trigram language model if we can estimate from the trigram, otherwise we use the bigram, otherwise the unigram. On the other hand, in interpolation we use the mixture of unigram, bigram, and trigram estimators. Currently, modified Kneser-ney smoothing method [17] has been shown to be high performance for n-gram modeling.

2.2.2 Cache Models

Cache models store the counts of recently used n-grams. In the most common case of a cache, we use an n-gram of size one. Since it is an n-gram of size one, it is referred to as unigram. The unigram probability distribution together with a background language model can be used to improve the quality of the next word predictions. A unigram cache could be of two types. The simplest one maintains a sliding window of words and can be stretched back to a certain window size or it uses all of user's previous data. The count of the words are updated as new user data comes in. The probability of a word w_i in a unigram cache can

be expressed as:

$$P_{cache}(w_i|w_1...w_{i-1}) = \frac{C(w_i)}{n} \tag{2.7}$$

where C represents count and n is the size of the cache.

We can use linear interpolation to merge the probability distribution of the unigram cache and a background language model (e. g. an n-gram model). We can express this mathematically as follows [9]:

$$P(w_i|w_1...w_{i-1}) = (1-\lambda)P_{cache}(w_i|w_1...w_{i-1}) + \lambda P_{background}(w_i|w_1...w_{i-1}) \tag{2.8}$$

where $0 \leq \lambda \leq 1$.

In Equation 2.8, λ represents how much weight of each model we should consider and it is a tunable parameter.

The second type of unigram cache is an exponentially decaying cache [21]. In this type of cache, the importance of a position decays exponentially with the distance from the current word. Mathematically,

$$P_{expCache}(w_i|w_1...w_{i-1}) = \beta \sum_{j=1}^{i-1} I_A\{w_i = w_j\} e^{-\alpha(i-j)} \tag{2.9}$$

where I is an indicator function such that $I_A = 1$ if w_i matches w_j, and 0 otherwise, α is the decay rate, and β is a normalizing constant. Similar to Equation 2.8, the background language model and the exponential decaying cache can be combined by linear interpola-

tion:

$$P(w_i|w_1...w_{i-1}) = (1-\lambda)P_{expCache}(w_i|w_1...w_{i-1}) + \lambda P_{background}(w_i|w_1...w_{i-1}) \quad (2.10)$$

where $0 \leq \lambda \leq 1$.

It has been shown that the cache models can be combined with neural network language models as well. Grave et al. [42] stored past hidden activations as memory and accessed them through a dot product with the current hidden activation. Merity et al. [78] introduced an architecture called pointer sentinel mixture architecture for neural sequence models which had the ability to either reproduce a word from the recent context or produce a word from a standard `softmax` classifier.

2.2.3 Prediction by Partial Matching (PPM)

Prediction by Partial Matching or PPM is a method to predict the next symbol given n previous symbols. Whereas an n-gram language model looks at previous n-1 symbols, a PPM with order n looks at n previous symbols. It is actually a text compression algorithm and it was first described by Cleary and Witten [22]. The main advantage of PPM compared to an n-gram is that it is easy to adapt on the fly. PPM uses a statistical context model to keep track of the symbols it has seen. It considers a context consisting of symbols of varying lengths. As each symbol is encoded, it generates a probability distribution over the next probable symbols. For example, a context occurring in a text may be *ever*. PPM algorithm counts the characters which appear after the context *ever*. If the context reappears in the text, then PPM uses the counts to predict the next probable character. In cases where PPM

can not estimate the probability distribution from the current context, it can back-off or escape to a shorter context. For example, if a probability distribution cannot be estimated from the context *ever*, PPM tries to estimate using *eve*, then *ev*, and so on.

2.2.4 Recurrent Neural Network Language Model (RNNLM)

N-gram models are simple yet effective language models to use in any kind of text entry system. They are fast, require no more than one iteration over training corpus, can build models on billions of words, and can use larger n with more data. But there are a couple of issues with n-gram.

First, n-gram language models cannot model long range dependencies. One would argue that a large n could be used to resolve this issue. But with a large n it would require a lot of space. Also, most of the larger n-grams being too infrequent would end up having probabilities close to zero. In real life, adjacent sentences tend to be on same topic. Even if the value of n is large, n-gram language models fail to use contextual clues from the previous words or across sentences. For example, when referring to the same entity across sentences, an n-gram model may not correctly predict the pronoun of the entity.

Second, n-gram models suffer from sparsity issues. N-grams cannot model words or instances it has not seen in the training examples. For example, if a n-gram model is asked to find the probability of a word `jumps` after the context `the quick gray fox` and if the context was not present in the training data, it will return a probability of zero.

PPM also suffers from both the issues. To overcome the shortcomings of an n-gram language model or PPM, a recurrent neural network language model (RNNLM) can be used. Bengio et al. [10] first proposed the idea of using artificial neural network for statistical lan-

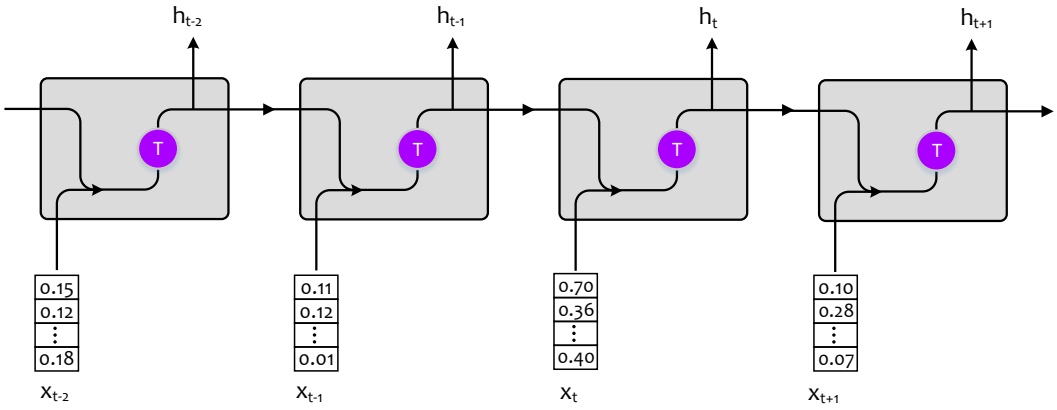

Figure 2.5: Standard recurrent neural network architecture. x_t denotes a vector representation of a character or a word at time step t. h_t denotes hidden state at time step t. T denotes an activation function.

guage modeling. They used a feed-forward neural network with fixed length context. Later Schwenk et al. [104] showed that neural network based models provide better performance than other existing systems. However, with Bengio's approach the context size was fixed and needed to be specified before training. This meant that the neural network could see a fixed number of words (5-10) from the past when predicting the next word. Mikolov et al. [83] proposed a recurrent neural network language model (RNNLM) which did not limit the context length. In this type of network, information can cycle for an arbitrarily long period of time.

A standard recurrent neural network architecture is shown in Figure 2.5. A recurrent neural network can be expressed by the following mathematical equation:

$$h_t = \varphi(Wx_t + Uh_{t-1}) \tag{2.11}$$

where,

h_t : hidden states at timestamp t

h_{t-1} : hidden states at timestamp t-1

W : weight matrix for input to hidden layer

U : weight matrix for hidden layer

φ : activation function

x_t : input vector at timestamp t.

In Equation 2.11 and Figure 2.5, x_t is an input vector at time t. When we deal with a recurrent neural network language model, we do not provide raw text as input to the RNN. Instead we convert the words or characters to a vector format. This representation is called a word/character embedding. Word embeddings allow words with similar meaning to have a similar representation. Word or character embeddings can be learned by the network like in Bengio et al. [10], but are often initialized using other algorithms like Word2Vec [80] or GloVe [92].

Recurrent neural networks are also not perfect. They suffer from major drawbacks such as vanishing or exploding gradients [47]. During training a recurrent neural network generally backpropagates to the previous hidden states and updates the model weights using gradient descent. As the context length increases the gradients become smaller and smaller if the initial gradient is less than 1.0. Consequently, the weights down the furthest context do not get updated much. This is called the vanishing gradient problem. On the other hand, if the initial gradient is larger than 1.0, then the gradients become larger and larger and result in an unstable network. This is called the exploding gradient problem. To solve these problems many variations of recurrent neural network has been proposed so far. These include using different activation functions (e. g. ReLU [3]), batch normalization [51], bidirec-

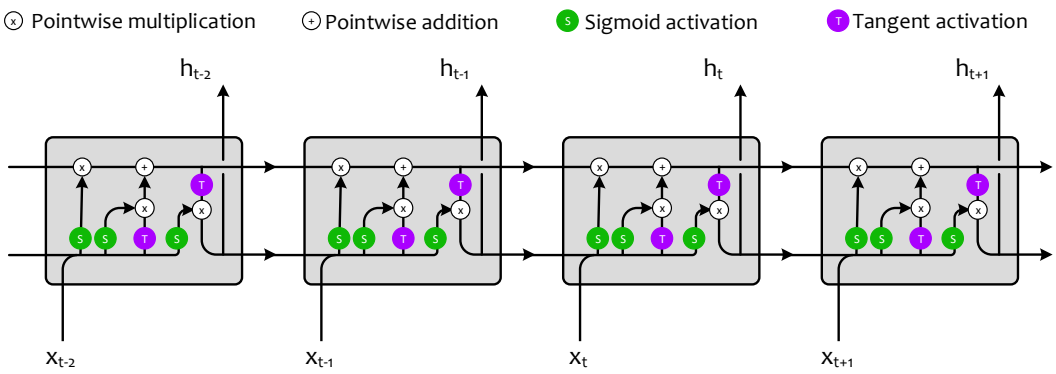

Figure 2.6: Recurrent neural network with LSTM units. LSTM units use different type of gates such as input gate, forget gate, cell state, and output gate and can keep track of the context in the hidden states through time.

tional RNNs [103], long short term memory (LSTM) network [48], and gated recurrent units (GRU) networks [20].

2.2.5 BERT

Even with LSTM or GRU units, the output of a recurrent neural network at a certain timestamp depends on the output of its previous states. A recurrent neural network works with sequential data. As a result, recurrent neural networks can only be trained from left-to-right or right-to-left and this slows down training. To allow parallel computation and to reduce training time, transformer models with a self-attention mechanism have been introduced [116]. Initially the idea of attention [6] and transformers were introduced in the context of machine translation. A machine translation architecture has two main components. One part of the system takes a sentence in a particular language as input (encoder) and the second part provides the translation of that sentence in another language as output (decoder). The attention mechanism maps different words in the output sentence to differ-

ent words in the input sentence and assigns higher weights to the words that are more important and relevant. On the other hand, self-attention quantifies the relative importance of each word compared to the other words in the input sentence.

Bidirectional Encoder Representations from Transformers or BERT [29] is a language model that introduced the idea of training a transformer bidirectionally. Whereas previous efforts looked at a text sequence from left-to-right or right-to-left [24, 49, 94], BERT first showed that language models can be trained from both directions at once and it can achieve a deeper sense of the language context. BERT introduced a technique called masked language modeling during training. The idea behind masked language modeling is hiding (masking) a word of a sentence and then telling the model to guess what word has been masked based on the surrounding words of the masked word.

2.3 How Text Entry Interfaces Use Language Models

So far we have discovered what is a language model and what are the different types of language models. Now we will discuss how different language models are integrated in a text entry interface and how the word predictions above a keyboard are generated.

The simplest approach to providing word predictions above a keyboard is to use the key labels from a user's input. Once the prefix of a word is inserted by a user, we can generate all the words starting with that prefix from a given dictionary of words. Then we can ask a character language model or a word language model to rank that list of words and suggest the top few results as predictions to the user.

The above approach works if the user does not make any mistakes while entering a prefix. But text entry interfaces are of different shapes and sizes and users make typing mis-

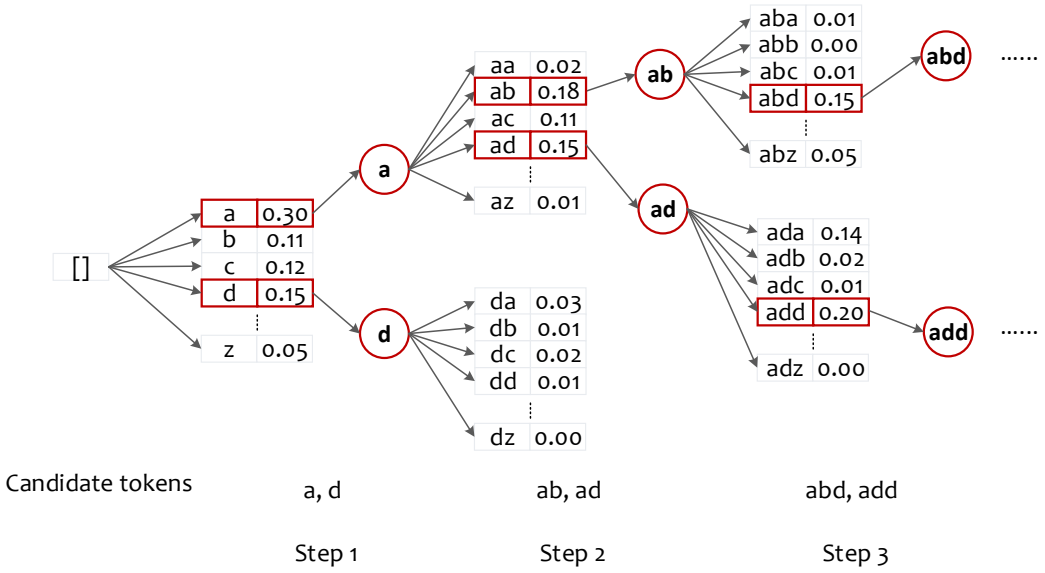

Figure 2.7: A decoder can generate a list of tokens using a beam search. In this figure a beam width of 2 is considered. For each token, the probability of the token according to a language model is shown. In each step of the beam search candidate tokens are pruned using a probability threshold and the beam width.

takes. For example, triggering a wrong key is very common in small form factor interfaces like smartwatches and smartphones. On such devices, we often press, touch, or tap a key that is not our intended key but might be adjacent to our target key. Therefore, a noise-tolerant system to handle such typing mistakes in the form of auto-correction is desirable.

We used the VelociTap decoder [126] to handle noisy input and for auto-correction. Instead of taking the key labels as input, VelociTap takes the noisy touch locations as input and searches (Figure 2.7) for the most likely word based on a probabilistic keyboard model, a character language model, and a word language model. We assume that the touch locations follow a two-dimensional Gaussian distribution centered at each key. Each key is assumed to have the same distribution. The distribution's variance in the horizontal and ver-

tical axes are independently controlled by two decoder parameters. For each tap, VelociTap calculates the likelihood of each key under the keyboard model. This likelihood is added to the probabilities from the decoder's character and word language models. The contribution of each language model is controlled by two additional parameters. The decoder also has two penalties allowing taps to be deleted, and characters to be inserted without a tap. For additional language models such as a unigram cache or a recurrent neural network language model we can introduce further tunable parameters to control how much each contributes to the total probabilities. Avid readers can see Vertanen [119] for a detailed overview of the decoding process.

2.4 Evaluation

Evaluation of a language model can be done extrinsically or intrinsically. In extrinsic evaluation, the language model is incorporated into an application and we measure how well the application works. For example, suppose we have two language models computed with two different approaches. We can incorporate these two language models in an end-system such as an AAC device and compare the language models by investigating how fast someone can communicate using each of the models.

Since performing an extrinsic evaluation in an end-system can be expensive, language models are often evaluated intrinsically. For an intrinsic evaluation of a language model we need a test set. Normally, the probabilities of a language model come from the corpus it is trained on. This is called a training corpus. We can measure the quality of a language model on some unseen test corpus. If we are given a corpus of text and we want to compare two different language models, we first divide the corpus into training and test sets. We then

train the parameters of both language models on the training set. After that, we compare the language models on the test set by observing how well the trained models model the test set.

2.4.1 Perplexity

When a model is predicting a character or a word for a given context, perplexity defines the degree of confusion of the model. Perplexity is one of the most important metrics in language modeling. When evaluating language models, instead of using raw probabilities we use perplexity. Perplexity measures the average number of choices a model has when predicting the next word. For example, perplexity of a phone number where all digits are equally probable is 10.

Mathematically, the perplexity of a language model on a test set is the inverse probability of the test set, normalized by the number of words:

$$PP(W) = P(w_1 w_2 \ldots w_n)^{-\frac{1}{N}} \tag{2.12}$$

$$= \sqrt[N]{\frac{1}{P(w_1 w_2 \ldots w_n)}} \tag{2.13}$$

$$= \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i | w_1 \ldots w_{i-1})}} \text{ (using the chain rule).} \tag{2.14}$$

The higher the probability of a model on a test set, the lower the perplexity. In language modeling, our goal is to maximize the probability of the test set, in other words, we want to minimize the perplexity.

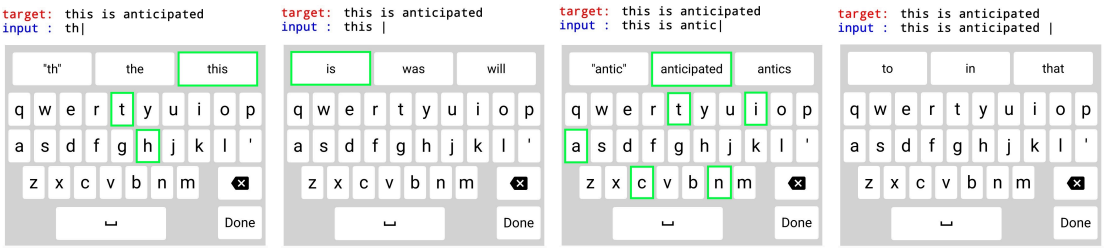

Figure 2.8: Text input progression with suggestion slots. A green bounding rectangle around a key or a slot represents that the user has tapped this key or slot. In the second image, a list of most probable words was offered based on the previous word `this`. For the phrase *this is anticipated*, there is a total of 10 taps.

2.4.2 Keystroke Savings

Another metric for evaluating a language model is keystroke savings. Perplexity does a good job indicating whether a language model is best suited to test data. A good language model would show a reduction in the perplexity on the test data and a bad language model will show an increase on the test data. However, in practice, perplexity gains may not necessarily represent if a language model will perform better in a text entry interface. In this case, keystroke savings metric is closer to measuring useful changes to a word prediction based text entry interface.

Keystroke savings measures the percentage reduction in keys pressed compared to letter-by-letter text entry. It can be expressed by the following formula:

$$KS = \frac{keys_{normal} - keys_{with\ prediction}}{keys_{normal}} \times 100\%. \tag{2.15}$$

Figure 2.8 shows an example where it took 10 taps to enter the phrase `this is anticipated`. There are a total of 19 characters in the phrase including the spaces. We can use the equa-

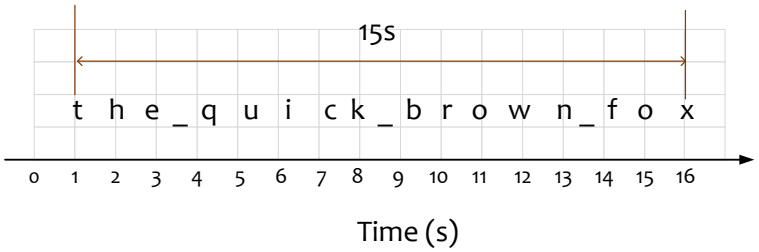

Figure 2.9: Input progress of a phrase with time. '_' represents a space.

tion 2.15 to calculate the keystroke savings. In this case, the keystroke savings is 47.4%.

2.4.3 Entry Rate

In text entry research, an interface is normally evaluated by examining how quickly and accurately a user can enter text using that interface. The metric that is used to measure how fast a user can type is entry rate. Entry rate is typically expressed by words-per-minute (WPM). WPM denotes the number of words a user can type in a minute using a text entry application. Here, a word is defined as consisting of five characters including spaces.

As for example, if a user takes 15 seconds to enter the phrase *the quick brown fox* (Figure 2.9) containing 19 characters, then the calculation of WPM will be:

$$WPM = \frac{19-1}{15\ seconds} \times \frac{60\ seconds}{1\ minute} \times \frac{1\ word}{5\ characters} = 14.4. \tag{2.16}$$

In the above equation, we subtract one from the total number of characters since timing starts after the first character is entered and ends after the final character is entered.

2.4.4 Error Rates

The metric that is used to measure how accurate a user can enter text is error rate. Usually the user performs the text entry task by copying a stimuli phrase. An error rate is calculated by comparing the stimuli and the text the user enters. We will use three types of error rates:

(i) Character Error Rate (CER): Character error rate or CER is the number of insertions, substitutions, and deletions required to transform user text into the reference text. More formally, CER is the ratio of total edit distance to convert user text to reference text and the number of characters in the original text. We typically multiply CER by 100 to show it as a percentage. But it is not actually a percentage, it is a rate that can be over 100% in the case of really broken input with many inserted characters in the user text.

$$CER = \frac{EditDistance\,(user\ text,\ reference\ text)}{number\ of\ characters\ in\ the\ reference\ text} \times 100\% \tag{2.17}$$

```
Reference: t h e  q u i c k  b r o w n  f o  x
User text: t h e  w u i c k  b r  w n  f a u x
```

Figure 2.10: A reference text versus a user text aligned to view character mismatch.

For example, let's assume, for the reference text *the quick brown fox*, a user typed *the wuick brwn faux*. In this case, if we align the two texts like in Figure 2.10, we can see that it requires two substitutions (in *wuick*, w→q and in *faux*, a→o), one insertion (o in brwn), and one deletion (u from faux) making a total of 4 edits to change the user text to the original text. The number of characters in the reference phrase including space between words is 19. Therefore, in this case the CER is $(4/19) \times 100.0$ or 21.05%.

(ii) Word Error Rate (WER): Word error rate or WER is similar to CER but instead

of characters we calculate the number of incorrect words in the user text. For example, if we compare the reference text and user text discussed in CER, out of four words in the user text only one word matches the reference phrase. So, the WER in this case is $(3/4) \times 100.0$ or 75.0%.

(iii) Sentence Error Rate (SER): Sentence error rate or SER is the percentage of sentences that are not completely correct.

In the next few chapters, we will use the evaluation metrics described above to determine how good a language model is and how users perform on a text entry task when the language models are used in a system. We will use perplexity as a primary metric to determine the quality of a language model. We will use keystroke savings metric to further validate if different language models can reduce user keystrokes when they are used in an end-system or in a simulated end-system. Lastly, we will use the entry rate and error metrics to evaluate user performance when a user is given the task to copy a phrase using a text entry interface with optimized language models.

3

Investigating Speech Recognition for Improving Predictive AAC

3.1 Introduction

Predictive AAC devices normally use a language model to try and make suggestions of likely upcoming text. These predictions are usually made based solely on the text entered

by the AAC user. They typically ignore the two-way nature of conversation which can offer many contextual clues.

In this chapter, first we investigate how to record and recognize the speech of a partner communicating with the AAC user. Then we investigate if speech recognition on partner speech improves two-sided conversational language modeling.

3.2 Related Work

Predictive AAC devices typically use an n-gram language model. The performance of this model largely depends on the training data being closely matched to a user's text. But for practical, ethical, and privacy issues, there is a scarcity of text written by AAC users. Researchers have resorted to training LMs on data from news articles [114] or phone transcripts [128]. Another option is the large amounts of text that can be mined from the internet, e.g. tweets, blog posts, or Wikipedia articles. While such web data may be informal or have other artifacts such as abbreviations, researchers have used filtering methods such as cross entropy difference selection [85] to select training data for AAC language models [123].

Recently, recurrent neural network language models (RNNLMs) have achieved state-of-the-art performance over traditional n-gram language models. RNNLMs have been shown to better model long range dependencies when combined with techniques such as long short-term memory [48] or gated units [20]. Further gains have also been achieved by interpolating n-gram models [82] and other techniques [81, 84].

In addition to using textual context, previous AAC work has also investigated using face detection [52], vision [53], and location [28] as context for AAC predictions. But lim-

ited work has been done to predict AAC user's response based on the speech of the AAC user's communication partner. Wisenburn [136, 137] created a program called Converser and used speech recognition to identify the speaking partner's words. This input was then parsed by a noun phrase identification system. Identified noun phrases, along with relevant static messages, were displayed to the user. This provided users with a faster communication rate compared to a system that did not use partner speech. In our work, we also perform speech recognition on the partner's speech. However, we will use recognition results as context to our language models in hopes of better predicting an AAC user's upcoming text.

3.3 Speech Data Collection

Our first step was to obtain text and audio data reasonably representative of everyday person-to-person conversations. In this section, we detail how we collected this data. Further, we designed our collection to answer the practical question of how and where a microphone might be located for recording a partner's speech.

As a starting point for our spoken dialogue collection, we used the text dialogues collected by Vertanen [117]. These dialogues were invented by workers on Amazon Mechanical Turk. The dialogue started with a question invented by one of the workers. Subsequent workers then extended the dialogue by another turn until a total of six turns were completed. Table 3.1 shows an example dialogue. The original collection had 1,419 dialogues. We removed 265 dialogues we deemed potentially offensive, resulting in a set of 1,154 dialogues.

A:	Did you call the theater?
B:	So sorry, I forgot to call the theater.
A:	You can just go online.
B:	That's true, I'll do that now.
A:	What movie is it that you want to see?
B:	The lord of the rings.

Table 3.1: A dialogue created by Amazon Turk workers.

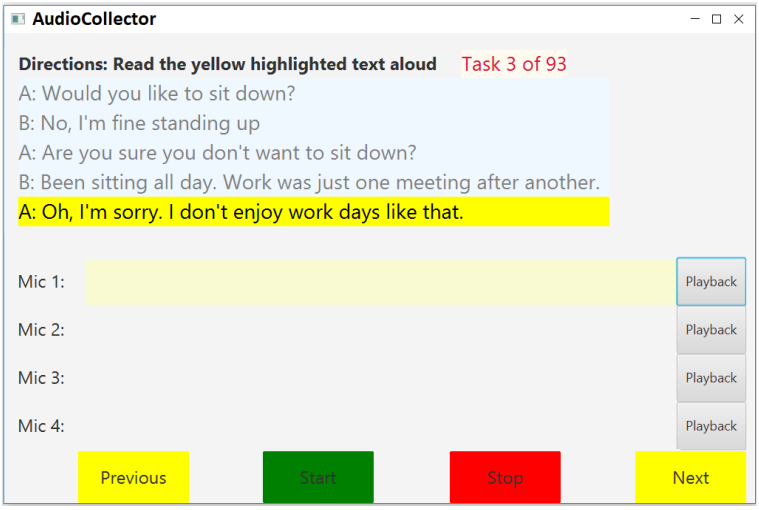

Figure 3.1: The application used to record dialogues.

3.3.1 Audio Data Collection

The dialogue data from [117] consisted only of text. We wanted to investigate whether a partner's speech could improve an AAC device's predictions. We designed a desktop application to record audio data of participants speaking turns in the text dialogues. The application highlighted the current turn we wanted the participant to speak. Any previous turns of the dialogue were also shown as context. The application recorded from three microphones simultaneously:

- Headset — A Logitech H390 USB noise cancelling headset microphone.
- Laptop — The built-in microphone of a 13" 2015 MacBook Pro laptop.
- Conference — A MXL AC404 USB conference microphone. This microphone was positioned behind the laptop at a distance of approximately 0.9 m from the participant.

The application allowed the participant to re-record any utterances in which they misspoke. We analyzed just the last recording for each dialogue turn. Audio was recorded at 44.1 kHz. We recruited 14 participants via convenience sampling. Four self-reported as male, ten as female. The average age was 36. Participant 5 reported having a foreign accent. Each participant took part in an approximately half-hour session and was paid $10. Participants sat at a desk with a laptop in quiet office. They were allowed to adjust their chair so they could comfortably operate the laptop.

Participants first recorded three practice dialogues. We did not analyze the practice dialogues. Each participant then completed half the turns in 28 additional dialogues. The subsequent participant completed the other half of the turns of the same 28 dialogues. In total, we collected 1,176 utterances constituting both sides of 196 dialogues. We have made our filtered text dialogues, audio recordings, recognition results, and Java audio collection application available to other researchers[*].

3.3.2 Speech Recognition Experiments

We performed speech recognition using two commercially available speech recognizers, Google Cloud Speech-to-Text and IBM Watson Speech-to-Text. We performed speech

[*] https://digitalcommons.mtu.edu/data-files/1

recognition on audio from each of the three different microphones. We computed the Word Error Rate (WER) of each recognition result against its reference transcript.

The reference transcripts included various numeric characters representing times or amounts. We found the recognition results on such turns were variable. Sometimes the recognizer returned numeric transcriptions and sometimes numbers were spelled out as words. Since our language models were not trained on text with numbers or use a number pseudo word, we dropped all dialogues if any of its reference turns had a number in it. This reduced the number of dialogues from 196 to 160.

As shown in Table 3.2, the mean WER on the three different microphones using the Google recognizer was Laptop 7.3%, Headset 7.0%, and Conference 8.9%. IBM's recognizer had higher error rates with Laptop at 10.5%, Headset at 10.7%, and Conference at 16.0%.

Figure 3.2 shows the WER for each participant using the Google speech recognizer and audio from the Headset microphone. 9 of the 14 participants had a lower mean WER of 5.5%. This was driven by the fact that 84.0% of their utterance turns were recognized with no errors.

We recorded our audio in a quiet office. We also wanted to explore how our methods might work in noisier locations. To do this, we injected a recording of street noise into our clean audio data. We used the SoX Sound eXchange utility to add in the street noise at three different volume levels: 0.1, 0.2, and 0.3. Figure 3.3 shows the mean word error rates on recordings with no noise and at the three noise levels. Even at noise volume level 0.3, both recognizers' mean word error rates using the Headset and Laptop microphones stayed below 40%. However, the mean word error rates using the Conference micro-

	Microphone		
	Laptop	Headset	Conf.
Google	7.3±1.0	7.0±1.0	8.9±1.2
Watson	10.5± 1.2	10.7±1.2	16.0±1.6

Table 3.2: Word Error Rate (WER %) using different microphones and speech recognizers. Results are formatted as mean ± 95% bootstrap confidence intervals.

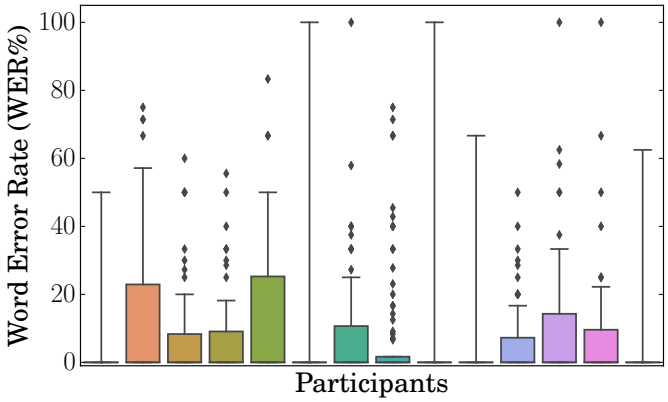

Figure 3.2: Participants' per utterance WER using the Google recognizer and audio from a headset microphone.

phone started deteriorating more sharply with increasing noise.

3.4 Language Modeling Experiments

We now investigate how to use language models to better predict turns in our dialogue collection. Recall we recorded both sides of 196 of the dialogues from our set of 1,154 dialogues. After dropping dialogues with numbers, we arrived at a test set of 160 dialogues with audio data. We created text-only training and development sets from the remaining 958 dialogues. From these dialogues, we dropped 128 that contained numbers. We randomly selected 160 from the remaining dialogues as a development set and 670 as a training

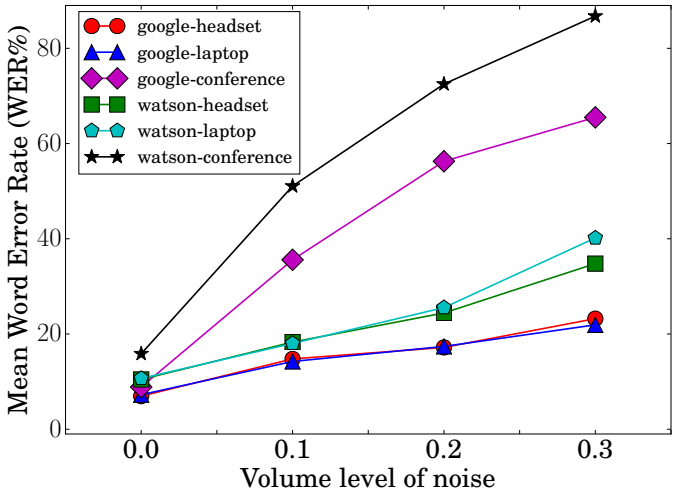

Figure 3.3: WER on audio dialogue turns without noise and with three different injected noise levels.

set.

Our language modeling experiments used a vocabulary of 35 K words. The vocabulary consisted of the most frequent known English words occurring in 50 M words of sentences parsed from Twitter. Any words not in this vocabulary were mapped to an unknown word token. We converted text to lowercase and removed punctuation aside from apostrophe. Throughout, we report the per-word perplexity of our test set (160 dialogues, 960 turns, 7.1 K words). We excluded the sentence end pseudo-word from our calculations.

3.4.1 N-gram Language Models

We took each turn in the training set as an independent training example (4,020 turns, 30 K words). We trained a 4-gram interpolated modified Kneser-Ney model using SRILM [108, 109]. As shown in Table 3.3, the perplexity on the test set was 211.8. For comparison,

Training data	Words	Perplexity
Twitter, small amount of data	30 K	417.3
Crowd dialogues	30 K	211.8
Twitter, large amount of data	50 M	96.0
+ CE diff. 25% dialogues	50 M	91.0
+ CE diff. 50% dialogues	50 M	86.8
+ CE diff. 75% dialogues	50 M	83.5
+ CE diff. 100% dialogues	50 M	83.5
+ Optimized CE threshold	50 M	77.4

Table 3.3: N-gram perplexity varying training data.

we trained a 4-gram model on 30 K words of random Twitter data collected via Twitter's streaming API between 2009–2015. The Twitter model had a much higher perplexity of 417.3.

An approach to filtering an out-of-domain training data is cross-entropy difference selection [85]. This approach calculates the cross-entropy of individual sentences under an in-domain and an out-of-domain model trained on similar amounts of data. We trained our in-domain model on between 25–100% of the text in our Turk dialogue training set.

We selected 50 M words of Twitter data below a certain cross-entropy difference threshold. We used an initial threshold of -0.3. The more negative the threshold, the more sentences had to resemble in-domain text in order to be selected. As shown in Table 3.3, using more in-domain data reduced perplexity though gains eventually flattened. Finally, we used all the in-domain data to search for the optimal cross-entropy difference threshold on the development set. The optimal threshold of -0.06 further lowered perplexity to 77.

Model	PPL Sentence	PPL Dialogue
Twitter RNNLM	179.0	129.3
+ GRUs	167.8	122.8
+ NCE	172.2	111.9
+ maximum entropy	123.7	84.1
+ Twitter 4-gram LM	75.2	71.5
+ unigram cache	75.2	68.5

Table 3.4: Perplexities with added features. We reset the RNNLM between each sentence or after each dialogue.

3.4.2 RNN Language Models

Next, we investigated training Recurrent Neural Network Language Models (RNNLMs) on the cross-entropy difference selected Twitter data. We trained our models using the Faster RNNLM toolkit[†]. For each model type, we trained 10 RNNLMs with different random initialization seeds. We report the perplexity on the test set of the model that had the lowest perplexity on the development set. Unless otherwise noted, we used the default hyperparameters of Faster RNNLM.

During evaluation we reinitialized the RNNLM after every sentence or after every six-turn dialogue. This allowed us to observe how much the model was adapting to a particular dialogue while avoiding allowing the model to adapt to the general style of our Turk dialogues.

As shown in Table 3.4, a model trained with 250 sigmoid units had a perplexity of 129.3 on each dialogue. Switching to 250 Gated Recurrent Units (GRUs) [20] reduced perplexity to 122.8. Switching to Noise Contrastive Estimation (NCE) [18] further reduced per-

[†] https://github.com/yandex/faster-rnnlm

plexity to 111.9. Training a maximum entropy language model of size 1000 and order 4 in the RNN reduced perplexity substantially to 84.1.

We interpolated our best RNNLM with our best previous n-gram model. We optimized the mixture weights with respect to our development set. This further reduced perplexity to 71.5. We also investigated a unigram cache [41]. Similar to the RNNLM, we reset the cache after each sentence or after each dialogue. The cache model provided a small reduction in perplexity to 68.5. The mixture weights were: n-gram 0.55, RNNLM 0.42, and unigram cache 0.04.

Comparing the result columns in Table 3.4, we see consistently higher perplexities when the RNNLM was evaluated on sentences instead of on entire dialogues. In particular, the RNNLM was substantially worse with a perplexity of 123.7 on sentences versus 84.1 on dialogues. This demonstrates the ability of the RNNLM to adapt to aspects of the text over a longer time horizon.

3.4.3 Two-sided Dialogue Language Models

We now turn to training language models on two-sided dialogues. Since our Amazon Turk dialogue collection is relatively small, we instead used dialogues from movies [25]. We created a training set of 83 K dialogues consisting of 305 K turns and 3.2 M words. We introduced a pseudo-word to denote speaker changes. We excluded this speaker change word from our perplexity calculations. We treated the set of turns making up a dialogue as a single "sentence" during training and testing. We evaluated models on each dialogue in our Turk test set (the same set used previously). In the case of RNNLMs, we reset the model after each dialogue.

Model	PPL
Movie dialogue 7-gram	138.5
Movie dialogue RNNLM	129.1
Turk dialogue RNNLM	185.5
Mixture, dialogue models	104.3
Mixture, Twitter + dialogue + cache	66.3

Table 3.5: Perplexity of models trained on two-sided dialogues and mixtures of dialogue and twitter models.

We first tested 4-gram through 8-gram n-gram models. The 4-gram had the highest perplexity of 139.2 The 7-gram model had the best perplexity of 138.5 (Table 3.5). Next we trained a RNNLM on the movie dialogues using 300 GRU units, NCE, and with a maximum entropy model of size 1000, order 4. The RNNLM had a lower perplexity of 129.1. This again highlights the ability of the RNNLM to better model long-range dependencies and/or topics compared to the n-gram model.

We also trained an RNNLM on just the Turk dialogues. We used 100 GRU units, NCE, and a maximum entropy model with 100 units and an order of 4. This model had a perplexity of 185.5. We think this model's worse performance reflects the substantially smaller amount of training data. By interpolating these three dialogue models, we obtained an even lower perplexity of 104.3. The mixture weights were: Movie 7-gram 0.24, Movie RNNLM 0.40, and Turk RNNLM 0.37.

Our two-sided models were trained on modest amounts of data. To see if they still offered gains in combination with models trained on substantially more Twitter data, we interpolated all our models. The mixture weights were: Twitter n-gram 0.43, Twitter RNNLM 0.32, movie dialogue n-gram 0.05, movie dialogue RNNLM 0.10, Turk dialogue RNNLM 0.06, and unigram cache 0.04. The mixture model's perplexity was 66.3, a modest gain

compared to the 68.5 obtained using a mixture of the Twitter models and unigram cache. It does however represent a more substantial gain compared to the 77.4 of the best n-gram only model. This shows that having access to both sides of a dialogue combined with the adaptive nature of RNNLMs may offer improved predictive AAC.

3.4.4 Impact of Speech Recognition Errors

In real-time person-to-person conversations, we cannot expect to have a perfect transcript of the other side of the conversation. We now investigate the impact of speech recognition errors on the performance of our language models. We did this by measuring the perplexity on two copies of the test set. In the first copy, we replaced the transcript of the even number dialogue turns with the speech recognition result of one of our participants speaking that turn. In the second copy, we replaced the odd number turns. We report the perplexity calculated from the odd turns from the first copy and the even turns from the second copy.

The entire six turns were provided to the language models for both copies to allow the model to condition on prior turns (including any speech errors). We reset the RNNLMs and unigram cache model between each dialogue. We used the previous best ensemble of six models which had a perplexity on the test dialogues of 66.3. We tested injecting the speech recognition results from the three microphones, two recognition engines, and four noise levels (none, 0.1, 0.2, and 0.3).

As shown in Figure 3.4, the perplexity of our ensemble of models only increased slightly when we replaced the reference transcripts with speech recognition results based on noise-free audio. For example, the far-field conference microphone had a WER of 8.9%. However, the errors introduced by recognition only slightly increased the perplexity of the di-

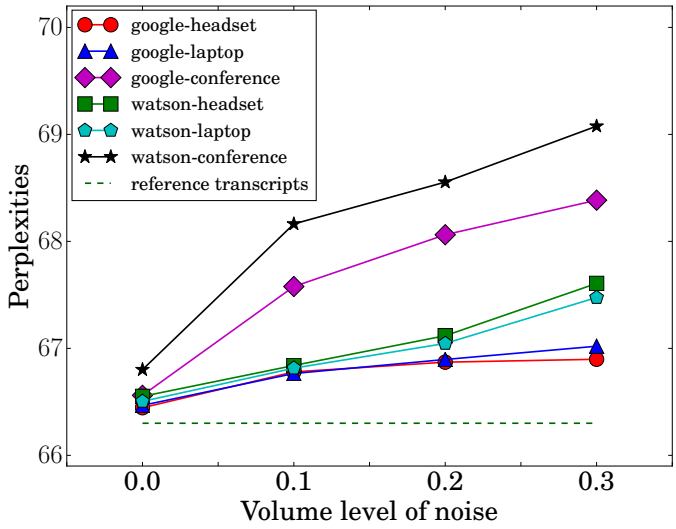

Figure 3.4: Perplexities using speech recognition on partner turns rather than reference transcripts. Results for no added noise, and for three levels of injected noise.

alogues from 66.3 to 66.6. Similar to WER in Figure 3.3, as the level of injected noise increased, perplexities also increased.

3.5 Discussion and Limitations

In this work, we conducted an initial investigation into the feasibility of performing speech recognition on an AAC user's speaking partner. We found that whether audio was captured from a wired headset or from a far-field microphone, we could recognize conversational-style utterances with error rates between 7–16%. We found Google's speech engine provided more accurate recognition than the IBM Watson recognizer. However, IBM's engine offers other benefits such as exposing probabilistic information about recognition results (e.g. a word confusion network). Such information might be leveraged to help avoid conditioning a predictive AAC interface on erroneous regions of a partner's speech recognition

result.

Our participants were given the verbatim text for each of their dialogue turns. As such, we can expect they spoke more fluently than one could expect in a spontaneous conversation. Further, we only collected audio in a quiet environment. While our results seem robust to artificially added noise, it remains to be seen if this holds for real-world noisy environments. As such, our error rates probably represent a lower-bound of what could be expected. Nonetheless, it is reassuring that our language models predict the non-speaking side's text with only minimal perplexity loses despite relying on text obtained via speech recognition.

Thus far we have focused on ascertaining whether there is a potential advantage to conditioning on recognition of the speaking side. Whether the perplexity gains we showed will result in actual practical improvements in the auspices of a predictive AAC interface remain to be seen. Further work is needed to understand whether these language model gains will result in, for example, better word predictions that actually save a user keystrokes. Even more work is needed to validate if end-user performance improves.

The use of speech recognition by an AAC device also has obvious privacy implications. This may require the AAC device or user to allow partners to opt-in to having their voice recognized. Further, our current work used cloud-based speech recognition. Users may prefer to have their speech recognized locally on device. Local recognition may also be necessary to avoid network latency or to allow use without network connectivity.

Our goal here was to demonstrate some of the building blocks necessary for modeling everyday conversational-style text. While we made some effort to optimize our models (e.g. tuning mixture weights on development data), further improvements are certainly

possible. For example, we did not conduct an extensive search for the best hyperparameters used during RNNLM training. Further, we need to investigate whether our methods and results scale to substantially more training data.

Our results show the benefits of language models based on recurrent neural networks. In particular, we found even when trained on non-dialogue data, RNNLMs adapted to the content of our short dialogues, providing good gains compared to an n-gram model. Further, we showed how a small in-domain corpus can be used to optimize models for everyday conversations. Despite our relatively small amount of two-sided dialogues data (3.2 M words of movie dialogues), we obtained improvements compared to using models trained only on much more non-dialogue data (50 M words of Twitter). In the end, we found an ensemble of n-gram and RNNLMs trained on sentence and dialogues combined with a unigram cache model provided the best performance.

3.6 Conclusions

AAC users often face challenges in taking part in everyday conversations due to their typically slow text entry rates. Predictions can provide an opportunity to accelerate their communication rate, but it is crucial these predictions be as accurate as possible. Leveraging real-world contextual clues offers one route to improving these predictions. In this work, we found speech can be accurately recognized with a variety of microphone configurations that might be deployed on an AAC device. Further, we found the error rates of current state-of-the-art recognizers allowed predictions nearly as good as having the verbatim text of the partner's turn. We think this work provides promising results showing a partner's speech can provide context to improve an AAC device's predictions. In the next chapter we

will see if it is also true for a real-world like use scenario.

4

Dwell-based Text Entry and Partner Speech Context

4.1 Introduction

One important design decision in a text interface is how many prediction slots we should provide in the interface. The intuitive decision might be that the number of prediction

slots in an interface should be as many as the interface can fit. But previous work [26, 37, 113, 121, 128] has shown increasing number of prediction slots does not necessarily improve performance. With large number of slots, there is an associated cognitive cost while looking for the right suggestion text. Also with increasing number of slots we have smaller slot targets which make selecting a slot difficult.

In this chapter, we examine how many slots we should use in a desktop dwell keyboard. In the previous chapter, we showed language models can achieve perplexity gains with partner speech as context. But perplexity gain is not always a good indicator of the performance during actual interactions on a device. For example, previous work [16] in the speech recognition domain has shown that the performance with perplexity measure did not correlate the performance with word error rate. In this chapter, we will re-examine if using partner speech as context improves performance but instead of measuring performance improvement using perplexity, we will use keystroke savings.

4.2 Related Work

In this chapter, we present two experiments. Our first experiment is a crowdsourced dwell keyboard experiment with multiple slot choices. In this experiment we used a desktop keyboard application running in a web browser. We had users dwell over a key for 1000 ms using the mouse pointer in order to click it. Although not exactly similar, our intention was to have a keyboard that works like a dwell keyboard with eye-gaze input. Our choice of 1000 ms dwell-time was based on previous work [75]. Past work has examined different choices of dwell time. Some work has used a static dwell time while some other approached with a dynamic dwell time. Majaranta et al. [75] has shown 1000 ms is a long enough dwell

time to prevent accidental activation. Mott et al. [86] investigated cascading dwell times (dynamic dwell times based on next likely key and its location) for eye gaze typing on a Microsoft Surface Pro 3. They conducted a longitudinal study with people without physical disabilities and found cascading dwell times significantly improved performance than a static dwell time. In their study participants started with a 600 ms dwell time and maximum dwell time was 1000 ms. The average cascading dwell time at the end of first session was 395 ms and at the end of the eight session was 334 ms. Other work [74, 98, 99] also investigated adjustable dwell times with longitudinal studies with able-bodied users. The dwell time found after the last session varied between 200 ms to 300 ms. We did not choose a adjustable dwell time because we ran a single session with each user and we were more interested in finding the appropriate number of slots for a text entry interface than the dwell time.

4.3 Experiment 1: Crowdsourced Dwell Keyboard Study

To examine how many slots are appropriate for a text entry interface designed for AAC users we conducted a user study using a desktop web application.

4.3.1 Design

We designed a keyboard application and ran the application in a browser. The keyboard interface was similar to a touchscreen mobile keyboard but we could control the number of suggestion slots above the keyboard. The keyboard contained all the characters from a to z, a spacebar, a backspace button, and a done button to move to the next phrase. There were 2–9 suggestion slots above the keyboard and a user saw only the same number of slots

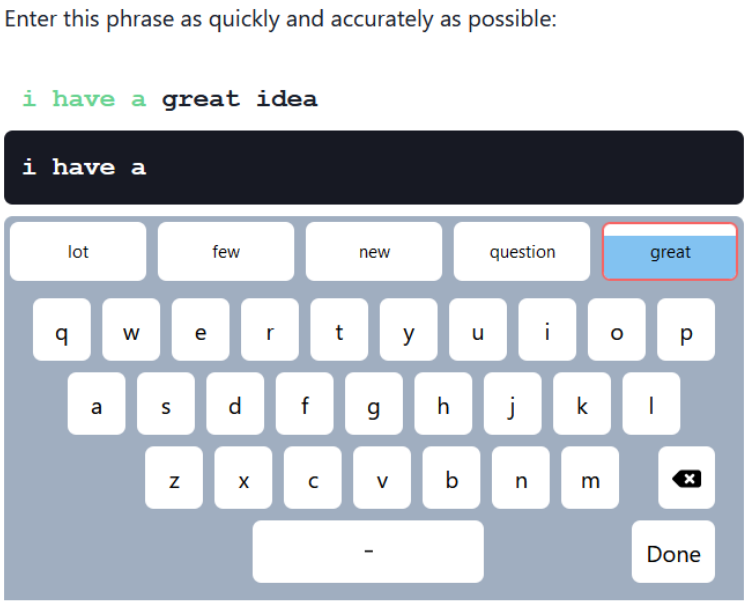

Figure 4.1: The dwell keyboard interface. In this example, the user is entering the text '*i have a great idea*'. The user has entered '*i have a*' so far and currently dwelling on the fifth suggestion slot containing the word '*great*'. The red rectangle indicates that the mouse pointer is currently hovering on this key and the blue shade fills the entire key after 1000ms (one second) has passed.

assigned to them throughout their session. If at any particular time there were word predictions less than the number of slots for that participant, it would show only the number of slots that were required to show the word predictions. Among the suggestion slots, one slot was always dedicated to show the literal text i. e. the series of letters nearest to each tap that the user entered before accepting a suggestion or clicking on the spacebar. The suggestions were shown above the keyboard in real-time. We used the VelociTap [126] decoder to provide the word suggestions. We hosted VelociTap on a server. The dwell keyboard application running on the user's web browser communicated with the decoder after each key press and updated the suggestion slots accordingly.

A user needed to dwell on a key or a suggestion slot using the mouse pointer for one second (1000 ms) in order to click it. When a user hovered on a key, a red rectangle around the key highlighted it and and a green shade started filling the key. If the user kept the mouse pointer more than one second on the key, the green shade filled the whole area of the key and a click sound was played. Then the character representing the key or the word representing the suggestion slot was added to the text field above the keyboard. Sometimes, during the input process, a suggestion slot was highlighted purple. This indicated that the text in that suggestion slot is the best suggestion based on what series of letters the user had entered and according to the language model we used. In this situation, if the user entered the spacebar, it would select the text from the highlighted suggestion slot. We did this to imitate the iPhone keyboard. In other cases, clicking the spacebar would select the text from the literal slot. Figure 4.1 shows the dwell keyboard interface with five suggestion slots.

Using the dwell keyboard, we ran a between subject user study with the number of suggestion slots as the independent variable. There were eight conditions: two slots through nine slots.

4.3.2 Procedure

We recruited 219 Amazon Mechanical Turk [*] workers and the workers were assigned the task of copying a series of phrases using our dwell keyboard. We instructed each worker to enter text as quickly and as accurately as possible. Workers wrote phrases written by people with ALS for voice banking purposes [23]. Each participant was assigned a number from 2–9 and they saw that particular number of suggestion slots above their keyboard through-

[*] https://www.mturk.com/

out their session. After entering each phrase, the participants saw the error rate and entry rate on that phrase. After completing all the phrases, they saw the average entry and error rate. We paid each worker $3.50 and it took approximately 20 minutes for each worker to complete their tasks. Each participant entered 32 phrases in total with the first two being practice phrases. During evaluation we did not consider the participants' performance on the practice phrases. A participant did not see the same phrase twice.

We logged each participant's interaction with the dwell keyboard. For example, what key they were hovering on, which suggestion slot they used, how many times they used backspaces, and how much time it took to get a response from the server. For each participant, we measured the average response time from their corresponding log file. We also calculated the percentage of response times that were more than 1000 ms. If that percentage was more than five, we replaced that participant with a new worker. These participants were likely to be experiencing networking delays. Due to networking delays a participant might not have had the best predictions available during typing. We replaced those participants so that their performance did not affect the overall input performance.

By analyzing the log files we also found some participants were experiencing issues due to a bug. The bug automatically selected a second word prediction immediately after a user selected a prediction and moved the mouse pointer to an area above the keyboard. To identify a phrase where this issue occurred, we saw if there were two consecutive taps on two suggestion slots within less than one second. Since participants dwelled on a key for at least one second, the time between two valid taps must be more than one second. We determined the number of phrases where a participant had the issue. If a worker had the issue in more than five phrases, we replaced that worker. Otherwise, we removed the problematic

Slots	Entry rate (WPM)	Error rate (CER%)	Backspaces-per-char	Keystroke Savings
2	9.1 ± 0.7	0.75 ± 0.40	0.04 ± 0.02	32.0 ± 5.4
3	10.2 ± 0.9	1.56 ± 0.80	0.02 ± 0.02	45.2 ± 4.1
4	9.7 ± 1.0	2.10 ± 1.11	0.04 ± 0.02	44.0 ± 6.4
5	10.7 ± 0.7	0.56 ± 0.34	0.03 ± 0.02	47.5 ± 4.9
6	10.8 ± 1.1	0.83 ± 0.32	0.02 ± 0.02	51.0 ± 4.8
7	11.0 ± 1.0	1.27 ± 0.98	0.03 ± 0.02	50.5 ± 4.8
8	10.3 ± 1.0	1.21 ± 1.18	0.03 ± 0.01	47.9 ± 6.1
9	10.5 ± 1.0	1.18 ± 0.74	0.03 ± 0.01	50.0 ± 6.4

Table 4.1: Entry rate, character error rate, backspaces-per-character, and keystroke savings result from dwell keyboard Amazon Mechanical Turk study. $\pm$ values denote 95% user-wise bootstrap confidence intervals.

phrases from the corresponding worker's data. After removing 59 problematic workers and 80 problematic phrases, we ended up having 160 users and 4720 clean phrases where the bug did not occur. For each number of slots from two to nine, we had 20 participants in the final set.

4.3.3 Results

Table 4.1 and Figure 4.2 shows the input performance of 160 participants using eight different conditions. We calculated entry rate using words-per-minute or WPM (equation 2.16). Participants entered about 9–11 words-per-minute using different number of slots. A one-way ANOVA revealed that there was not a statistically significant difference in entry rate between at least two slots ($F_{7,152} = 1.151$, $p = 0.334$). Users entering text with 2-slots had the minimum entry rate with 9.1 WPM and users with 7-slots had the maximum entry rate with 11.0 WPM.

We calculated error rate using character error rate or CER (equation 2.17). CERs were

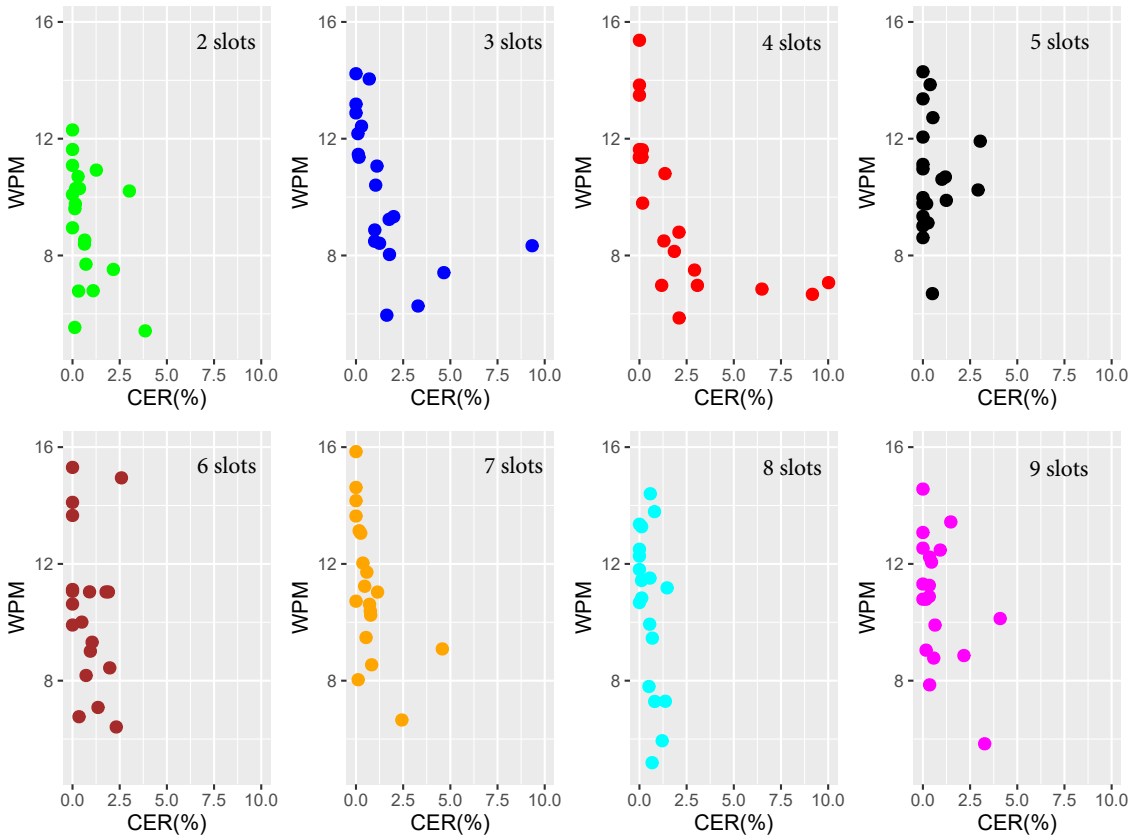

Figure 4.2: WPM versus CER plot for each slot. Each dot represents a participant.

low and ranged between 0.56%-2.10%. A one-way ANOVA test was not significant ($F_{7,152}$ = 1.014, p = 0.424).

We can see an upward trend with the entry rate but it plateaued after 7-slots. We can also see that the entry rate with 4-slots did not follow the trend. 4-slot users also had the highest error rate. We investigated to see why this happened. We plotted participants' average entry rate versus average error rate using different slots and found eight 4-slot users had an average entry rate lower than 8 WPM (Figure 4.2). Among these users, five had a CER of more

than 2.5%. We think the slow entry rate using 4-slots was due to human error.

We calculated the backspaces-per-character with all slots. Since it took participants at least one second to click a key, participants could stop dwelling on a key if it was less than a second and they were dwelling on a wrong key. Still participants made mistakes and sometimes selected a wrong key or a wrong word suggestion. We wanted to see how frequent participants were using backspaces to make corrections. Average backspaces-per-character was low during all slots. It was roughly four backspaces per 100 characters.

We calculated the keystroke savings on each phrase for each of the number of slots. Keystroke savings was maximum with 6-slots at 51% but after that it flattened with increasing number of slots.

4.3.4 Approximate Time Spent per Tap

Given our text entry interface and the dwell time, we were interested to know how much time the participants spent on each key. For each phrase, we calculated the elapsed time between the time the participant started dwelling on the first letter of the phrase and the time the participant dwelled on the done button. We divided the elapsed time by the total number of taps including tapping the done button to calculate the spent time per tap. Then we calculated the average time spent per tap for each participant. The average time spent per tap was 2298 ms, with a minimum of 1609 ms, and a maximum of 5273 ms. We can use this average time spent per tap to approximate the entry rate for a given keystroke savings. If the average time spent per tap is T_{tap} ms and the keystroke savings is $KS\%$, then

the approximated entry rate in words-per-minute can be found by the following formula:

$$\text{Approximate entry rate} = \frac{100}{(100 - KS) \times \frac{T_{tap}}{1000} \times \frac{1}{60} \text{ minute}} \times \frac{1}{5} \text{ words} \qquad (4.1)$$

For example, for a keystroke savings of 55% and a 2298 ms time spent per tap, it would translate to an entry rate of 11.6 words-per-minute.

4.4 Experiment 2: Using Partner Speech Context in a Simulated Keyboard

In Chapter 3, we performed speech recognition on the partner speech and used the speech recognition results as context to a language model. We evaluated the language models with and without partner speech context and perplexity results suggest that the language models performed better in predicting the AAC user's next word when they were informed by the partner speech context. However, perplexity is an intrinsic evaluation metric and it does not always correlate with the performance of an extrinsic evaluation metric [16]. Therefore, we wanted to further examine the performance of various language models by deploying them in a downstream task, such as, in a simulated text entry application.

In this experiment, we will use the language models in the simulated text entry application and ask the language models to predict the next word given a context. We will also provide additional context to the language models in the form of partner speech context. We will then evaluate the language models based on the top three to five word predictions they make and if any of the predictions matches a target word and if thus reduces the number of keystrokes. In this case, we will consider a simulated keyboard where an ideal user enters each character of a given sentence progressively. We will assume the ideal user is always true to their intent, does not make any typing mistake, and always picks a word from the list of

A:	What are we having for breakfast?
B:	Pancakes and bacon.
A:	Sorry, I'm allergic to bacon.
B:	Should I make sausage instead?
A:	No, I don't like it.
B:	Well, what do you want?

Table 4.2: An example of a dialogue between an AAC user and a communicating partner. During training and evaluation we removed punctuation and converted the cases to lower case.

word predictions if it matches their intended word.

4.4.1 Data

From Chapter 3, we had the recorded audio clips and speech recognition transcripts of 196 dialogues. We converted the transcripts to lower case, removed punctuation, and filtered out dialogues which contained characters not in a-z and apostrophe. We ended up with 151 dialogues.

4.4.2 Language Models

We used a 12-gram character language model trained by Vertanen [118] as our baseline model. This model was trained from text collected from different sources including Subtitle [†], Reddit, Common Crawl, and Twitter. Due to the large size of the final 12-gram model, we used a medium sized (213 MB compressed) pruned version.

We used the Reddit dialogue dataset [79] to train our RNNLMs. This dataset contained text data from 1000 sub-reddits with over 75,000 subscribers in each. In this dataset, up to two dialogues were sampled per post and each dialogue had at least four turns. We chose

[†] http://opus.nlpl.eu/OpenSubtitles.php

this dataset because we thought it would be good for modeling two way conversations such as ours. We trained two recurrent neural network language models on this dataset. One RNNLM was trained on text containing the text of one side of the dialogues. The other RNNLM was trained on text containing the text of both sides of the dialogues. We added a <ns> delimiter in each dialogue to indicate the change of speaker.

Consider the dialogue shown in Table 4.2. It has two sides (A and B) and six turns. When training the one-sided RNNLM we took the turns from only side A or side B. In this case, an example training instance was: `what are we having for breakfast <ns> sorry i'm allergic to bacon <ns> no i don't like it`. On the other hand, for training the two-sided RNNLM we considered each turn of the dialogue. In this case, a training instance was a text containing all the turns separated by the <ns> delimiter. Before training the RNNLMs, we removed punctuation and converted case to lower case.

4.4.3 Experimental Setup

In this experiment, we will examine the following cases:

(i) We will use only the n-gram language model. We will treat each turn in a dialogue as an independent input and calculate keystroke savings on them.

(ii) We will examine if rescoring with an RNNLM trained on one-sided dialogue data and conditioning on previous turns of the AAC user improves entry rate compared to (i). For example, in Table 4.2, if we want to calculate the keystroke savings on the 6^{th} turn, we will provide turn two and turn four as context to the one-sided RNNLM.

(iii) We will test if rescoring with an RNNLM trained on two-sided dialogue data and conditioning on previous turns from both sides (speech recognition for partner's turns) further improves entry rate compared to (ii). In this case, we will test using noisy speech recognition results versus with reference transcripts. For example, in Table 4.2, let us assume side A is the AAC user and side B is the communicating partner. Side A is using an AAC device and side B is speaking. To calculate keystroke savings on the 5^{th} turn, we will run speech recognition on the 2^{nd} and 4^{th} turn. Then we will provide turn one, speech recognition result from turn two, turn three, and speech recognition result from turn four as context to the two-sided RNNLM. In another case, we will provide the reference text from all the previous turns as context.

For experiment (iii) we used the speech recognition results from Google Cloud Speech-to-Text described in Section 3.3.2. In this case, speech recognition was performed on the audio collected using the headset microphone. The word error rate on the speech recognition results was 7.0%.

4.4.4 Procedure

We used a program to simulate a user entering text on a keyboard. The program used the VelociTap decoder [126] to handle the simulated user input. We assume the simulated user enters text progressively character-by-character and makes use of the word predictions offered by VelociTap. We also assume that the simulated user does not make any error during writing. For each character input, VelociTap generates a list of the most probable words based on a touch model, a character language model, and a given list of words. There were 100K words in the list.

To test our two hypotheses, we used the one-sided RNNLM and the two-sided RNNLM with the base 12-gram character language model. For uncommon words or proper nouns, VelociTap might not offer a good list of word predictions. In such cases, we assume that the simulated user enters all the characters of that uncommon word. To facilitate this, there is a slot with literally what they have typed thus far. We call this a literal slot. We tested three to five suggestion slots including the literal slot. For each sentence in the test set, the simulated user provided progressive input and selected a target word as soon as it was offered. Our simulation program automatically entered a space after a word upon selection of a word prediction. We trimmed the space after the final word. We calculated the number of taps it required to complete the sentence and the number of actual characters including spaces in the sentence. From these numbers then we calculated the keystroke savings (KS) using Equation 2.15.

To determine how much of the one-sided RNNLM and the two-sided RNNLM contribute we created a development set using the DailyDialog dataset [67]. Then we calculated the scale factors by simulating on one-sided text and on text from the both sides of the dialogues in the development set.

Since using different number of slots participants' performance was similar in section 4.3.3, we decided to simulate the experiments with three, four, and five slots. In all these three cases, one slot was always a literal slot. For each number of slots, we ran four tests. First, we considered each turn of the dialogues in our evaluation set as independent turns and calculated keystroke savings (KS) using only the 12-gram character language model. Second, we only considered the turns of one side of the dialogues and conditioned on the previous turns of that very same side. For example, to calculate KS on the 3^{rd} turn of a di-

alogue, we used the 1^{st} turn as context. To calculate the KS on the 4^{th} turn, we used the 2^{nd} turn as context. In this case, we used the one-sided RNNLM (Section 4.4.2) with the 12-gram character language model. Third, we used the turns from the both sides of the dialogues. However, when calculating the results on turns of one side, we used the speech recognition results from the opposite side. For example, to calculate keystroke savings on turn four of a dialogue, we provided speech recognition results from the 1^{st} turn, reference text from the 2^{nd} turn, and speech recognition results from the 3^{rd} turn. In this case, we used the two-sided RNNLM with the 12-gram character language model. Fourth, we used the reference texts from the both sides of the dialogues. In this case also the two-sided RNNLM was used.

For each test described above, we took the approximate time spent per tap (2298 ms) from section 4.3.4 and estimated the entry rate in words-per-minute from the calculated keystroke savings.

4.4.5 Results

Table 4.3 shows the results from our experiment. Using three slots and considering each turn independently the baseline 12-gram language model had a keystroke savings of 51.2%. Considering previous turn of the same side as context and using the one-sided RNNLM it increases to 52.0%. Using the two-sided RNNLM and partner speech recognition results as context, the keystroke savings further increases but only slightly to 52.5%. Noticeably, this keystroke savings is on par with the keystroke savings when a two-sided RNNLM and two-sided dialogue transcript is used. We can see similar trend in the keystroke savings results when four and five slots were used. Partner speech context helps improve performance

Model	Slots	KS (%)	Est. entry rate (WPM)
12-gram	3	51.2±0.6	10.7
+RNNLM (1-sided)	3	52.0±0.6	10.9
+RNNLM (2-sided, partner speech)	3	52.5±0.6	11.0
+RNNLM (2-sided, ref transcript)	3	52.4±0.6	11.0
12-gram	4	54.8±0.6	11.6
+RNNLM (1-sided)	4	55.4±0.6	11.7
+RNNLM (2-sided, partner speech)	4	55.9±0.6	11.8
+RNNLM (2-sided, ref transcript)	4	55.9±0.6	11.8
12-gram	5	57.5±0.5	12.3
+RNNLM (1-sided)	5	57.8±0.6	12.4
+RNNLM (2-sided, partner speech)	5	58.2±0.5	12.5
+RNNLM (2-sided, ref transcript)	5	58.3±0.5	12.5

Table 4.3: Keystroke savings and estimated entry rate results from the simulated keyboard experiments. ± values denote sentence-wise 95% bootstrap confidence intervals [11].

in comparison to using only the context from the same side but the performance gain is small.

If we look at the approximated entry rates for each slot and different experimental setups, the improvement in entry rate is also small.

To examine if the performance gain is more pronounced when the context is larger, we conducted the four simulation experiments described in Section 4.4.4 again but only on the last turn of the dialogues. Table 4.4 shows the results. From the results we can see that for each slot keystroke savings and entry rate improves with different experimental setups. When 3-slots are used the baseline model had a keystroke savings of 49.9% and an estimated entry rate of 10.4 WPM. Using the one-sided RNNLM and text from the one side of a dialogue increased keystroke savings to 51.2% and estimated entry rate to 10.7 WPM. With two-sided RNNLM and partner speech context keystroke savings further increased

Model	Slots	KS (%)	Est. entry rate (WPM)
12-gram	3	49.9±1.4	10.4
+RNNLM (1-sided)	3	51.2±1.4	10.7
+RNNLM (2-sided, partner speech)	3	52.1±1.8	11.0
+RNNLM (2-sided, ref transcript)	3	52.8±1.5	11.1
12-gram	4	53.6±1.3	11.3
+RNNLM (1-sided)	4	55.2±1.4	11.7
+RNNLM (2-sided, partner speech)	4	55.9±1.3	11.8
+RNNLM (2-sided, ref transcript)	4	56.5±1.3	12.0
12-gram	5	56.9±1.3	12.1
+RNNLM (1-sided)	5	57.4±1.3	12.3
+RNNLM (2-sided, partner speech)	5	58.1±1.3	12.5
+RNNLM (2-sided, ref transcript)	5	58.7±1.3	12.6

Table 4.4: Keystroke savings and estimated entry rate results on the last turn of each dialogue from the simulated keyboard experiments. ± values denote sentence-wise 95% bootstrap confidence intervals [11].

to 52.1% and estimated entry rate increased to 11 WPM. In comparison to the baseline model it was about 2.6% increase in the keystroke savings. When compared to the one-sided RNNLM, it was about 1.8% increase in keystroke savings. For 4-slots and 5-slots we can see similar trends.

4.5 Discussion

In this work we first conducted a crowdsourced user study with able-bodied rate limited user to determine the optimal number of slots in a text entry interface. We used a variety of slot choices ranging from two to nine but did not find any choice that provides a significant performance improvement. Except with only two slots, participants entered text at 10-11 words-per-minute and had keystroke savings of 44–51%. With increasing number of slots,

the performance seemed to flatten after 5 slots. This is similar to the finding in Vertanen et al. [121].

From the dwell keyboard user study, we also calculated the average time a user took to actuate a tap. This helped us estimate in an ideal input method, how much overhead is associated with checking and selecting slots. Using this measure then we were able to calculate estimated entry rates in Section 4.4 using only the keystroke savings metric.

We used a static dwell-time of 1000 ms for the crowdsourced user study. As we discussed past work has also investigated dynamic dwell time with eye-gaze input. But since we used the mouse pointer to have the participants dwell on the key we did not use a dynamic dwell time. Future direction of this experiment would be to use different dwell times for example, 500 ms or 750 ms and see the performance. Our technique of dwelling using a mouse pointer was different from a traditional eye-gaze and dwell technique. Traditional eye-gaze would involve an actual eye or head tracker. Also able-bodied participants may differ from actual AAC users. Therefore, our slot study needs further validation with eye-gaze keyboard and with rate-limited users.

We conducted a simulation experiment examining whether conditioning on partner speech context improves language model performance. From the results, we found speech context improves performance but we did not find any substantial gains. This could be due to the RNNLM training set. We trained our one-sided and two-sided RNNLM on Reddit text. The text we trained on are not dialogues per se, but are user comments sampled from different Reddit topics. Sometimes these comments have longer form of text. Therefore, they might not be a good representative of text similar to the colloquial conversational text. We think better gains are possible if the two-sided conversational language models were

trained on data that more resembles everyday conversations.

4.6 Conclusion

In this chapter, we first conducted a user study with Amazon Mechanical Turk workers asking them to enter text on a web keyboard by dwelling on a key for one second using a mouse pointer. We offered different number of suggestion slots to the workers and examined with what number of slots the workers could enter text faster. We did not find an optimal number and we suggest choosing any of 3, 4, or 5 slots. However, we believe more work is still needed to justify these choices. For example, conducting user studies with an eye-gaze keyboard and directly involving rate-limited users.

We also investigated adding partner speech context to neural language models to improving next word predictions. Our simulation results yielded small performance gain when evaluated against the number of keystrokes that can be saved during the input process. But we think this is promising. We believe better gains are possible with better language models trained on large and more suitable text dataset.

5

Accelerating Text Communication via Abbreviated Sentence Input

5.1 Introduction

Experienced desktop and touchscreen typists can often achieve fast and accurate text input by simply typing all the characters in their desired text. For AAC users, such quick and

precise input is difficult due to their motor ability. They may use a virtual touchscreen keyboard, but their touch locations may be slow and inaccurate, e.g. people with Cerebral palsy. Other users may need to click keys by pointing at them with a head- or eye-tracker and dwelling for a fixed time, e.g. people with amyotrophic lateral sclerosis (ALS).

When a person's typing is slow or inaccurate, word completions may provide more efficient input. Word completions predict the most probable words based on the current typed prefix. However, monitoring predictions carries a cognitive cost and may not always improve performance [114]. Further, monitoring predictions can be difficult without visual feedback. Eyes-free text input can be slow for users who are visually-impaired [88], and even slower for users who are motor- and visually-impaired [87]. Finally, eyes-free text input may be needed in future augmented reality (AR) interfaces where visual feedback is limited or non-existent (e.g. due to lighting or device limitations). In audio-only AR, it is still possible to type on an invisible virtual keyboard [127, 142].

All these cases motivate our interest in exploring alternatives to conventional word completion. In this chapter, we investigate accelerating input by allowing users to skip typing spaces and mid-word vowels. We decided to abbreviate in this manner based on past results on touchscreen text input without spaces [120, 126], and a study we present here in which 200 people abbreviated email messages. Our interaction approach of abbreviation is similar to features in commercial assistive interfaces (e.g. Grid 3, NuVoice, Lightwriter). Our whole utterance prediction approach is similar to features in touchscreen phone keyboards and in commercial assistive interfaces (e.g. dwell-free sentence input in Tobii Communicator 5).

We modified a probabilistic recognizer to accurately expand abbreviated input by 1) im-

proving our language models by selecting well-matched training data via a neural network, 2) modifying the search to model the insertion of mid-word vowels, and 3) adding a neural language model to the search. We validate our method in computational experiments on over six thousand sentences typed on touchscreen devices. We found that even when 28% of letters were omitted, we recognized sentences with no errors 70% of the time. Selecting from the top three sentences, user could obtain their intended sentence 80% of the time.

Finally, we compare word completion and abbreviated sentence input in a user study. In this study, users had to dwell for one second to trigger a tap. We found sentence input was slightly slower than using word completions, but still saved substantial time compared to typing all the characters. Users obtained their desired sentence 68% of the time.

5.2 Related Work

Abbreviated input. Demasco & McCoy [27] investigated expanding uninflected words (e.g. "apple eat john") into syntactic sentences (e.g. "the apple is eaten by john"). Gregory et al. [43] created abbreviation codes (e.g. "rmb" = "remember"). Users selected words from a menu or by typing a code's letters. Typing codes was the most efficient. Pini et al. [95] detected abbreviated phrases using a Support Vector Machine and expanded them via a Hidden Markov Model (HMM). Their detector and expander were 90% and 95% accurate respectively. Users decreased keystrokes and input time by 32% and 26% respectively.

Shieber & Nelken [106] allowed users to drop non-initial vowels and repeated consonants. This deleted 26% of the total characters. Using an n-gram word language model and a spelling transducer for each word, they expanded abbreviated text at an error rate of 3.3%. Our work differs in that we: 1) removed spaces between words, 2) did not remove consecu-

tive consonants, 3) used a character language model with no fixed vocabulary.

Tanaka-Ishii et al. [111] explored Japanese text input with digits. They used an HMM to expand a sequence of digits into characters. Users saved 35% of keystrokes typing on a mobile phone. Han et al. [44] also used an HMM to expand abbreviations learned from a corpus of Java code. Their approach did not require memorizing abbreviations and provided incremental feedback while typing.

In two studies with 31 users, Willis et al. [133, 134] identified common abbreviation behaviors such as vowel deletion, phonetic replacement, and word truncation. They did not release their data and it was on a relatively small number of people. Based on their work, we conducted an abbreviation study with 200 users and also share our data.

Data selection. Mismatch between the training and target text domains can lead to suboptimal language models. A variety of methods have been developed to address this problem. Lin et al. [68], Gao et al. [36], and Yasuda et al. [138] used language modeling and in-domain perplexity to select training data. In this approach, a language model is trained on a small in-domain dataset. Training instances from an out-of-domain dataset are selected if they are below some perplexity threshold.

Other work has investigated data selection using cross-entropy or cross-entry difference between in- and out-of-domain datasets [5, 76, 85, 101, 105, 123]. In this approach, an in-domain and out-of-domain language models are first trained. Sentences are selected based on a cross-entropy threshold or cross entropy difference calculated from the two language models.

Hildebrand et al. [46] and Lü et al. [70] applied information retrieval based techniques to select data. Other method include selecting based on infrequent n-gram occurrences

[38, 90], or Levenshtein distance and word vectors [19].

Duh et al. [33] employed the data selection method of Axelrod et al. [5], which builds upon Moore & Lewis [85]'s approach. The main distinction is that they used neural language models for selection rather than n-gram models. Chen & Huang [13], Peris et al. [93], and Chen et al. [14] selected based on convolutional and bidirectional long short-term memory neural networks.

Bidirectional neural models like BERT [29] has proven effective in many natural language tasks. Ma et al. [71] used BERT for domain-discriminative data selection. Hur et al. [50] used BERT for domain adaptation and instance selection for disease classification. Our selection method is similar to these methods but focuses on selecting conversational-style sentences.

Decoding noisy input. Text entry interfaces often use a probabilistic decoder to infer a user's text from time sequence data [61, 126, 140, 141]. Typically, a keyboard likelihood model and a language model prior are used to infer a user's text from input with incorrect, missing, or extra characters. To date, these approaches have mostly used n-gram language models.

Ghosh & Kristensson [39] corrected typos in tweets to a low character error rate of 2.4% by using a character convolutional neural network, an encoder with gated recurrent units, and a decoder with attention. The twitter typo data contained sequences with a similar number of characters to the target. In our work, we show acceptable character error rate can be achieved on input not only with typos, but also with missing spaces and mid-word vowels. We show the advantage of using a recurrent neural network language model (RNNLM) directly in the decoder's search or to rescore hypotheses.

5.3 Free-form Abbreviation Study

To better understand how people do free-form abbreviation, we conducted a study on Amazon Mechanical Turk. As a pilot, we had 26 workers abbreviate an email from the Enron mobile data set [122]. We designed our instructions based on Willis et al. [133]. Workers abbreviated the same email three times. Each time the worker was asked to abbreviate in three ways: heavily, as little as possible, or as they saw fit.

In our pilot, we found workers abbreviated similarly regardless of instructions. Thus, we designed a single set of instructions for our main study that asked workers to imagine they were using artificially intelligent (AI) software that was good at guessing their intended text from an abbreviated form. They were told to shorten words by removing or changing letters, but they should avoid shortening words that might be hard for the system to guess and that they should not omit words entirely. See appendix A.1 for our instructions. Our supplementary data* contains all the data from the study.

We recruited 200 workers who each abbreviated ten emails. In our analysis, we used 1,308 of the 2,000 emails. We filtered out emails that did not have the same number of words as their original emails. This filtering helped us to align the sentences by word. Punctuation was removed except apostrophes and at signs. We lowercased the text.

5.3.1 Abbreviation Behavior

We found 90% of abbreviated words were an in-order subsets of their full spelling. On average, 21% of a word's letters were deleted. Of these, 16% were consonants and 42% were vowels. In the set of six common letters in English, `e t o a i n`, consonants were less likely

*https://aclanthology.org/attachments/2021.acl-long.514.OptionalSupplementaryMaterial.zip

to be deleted than vowels. Surprisingly, the six least common letters, z q x j v k were often deleted. Considering letter position in words, 14% of first letters, 35% of last letters, and 90% of middle letters were deleted.

Our study confirmed some of our initial beliefs about how people would do free-form abbreviation. We found people deleted vowels more frequently than consonants and people usually retained the first letter of words. Other aspects we found surprising such as the frequent deletion of uncommon letters. The percentage of middle letters deleted was high. One reason for this was some workers persistently only used the first letter of each word.

5.3.2 Initial Automatic Expansion Experiment

We selected 564 passages where each word was an in-order subset of the full word. We implemented a search that proposed inserting all characters at all positions in words in workers' input. The search was guided by the language models described in Vertanen et al. [126]. We used beam search to keep the search tractable. See table A.2 in the appendix for example input and the expanded output.

We measured accuracy using character error rate (CER). As shown in Figure 5.1, the expansion had a CER of less than 5% for compression of up to 30%. Beyond that, much of the input was only the first letter of each word and our algorithm simply imagined probable text consistent with the provided letters. We think these results are promising given our search simply proposed the insertion of all characters at all positions.

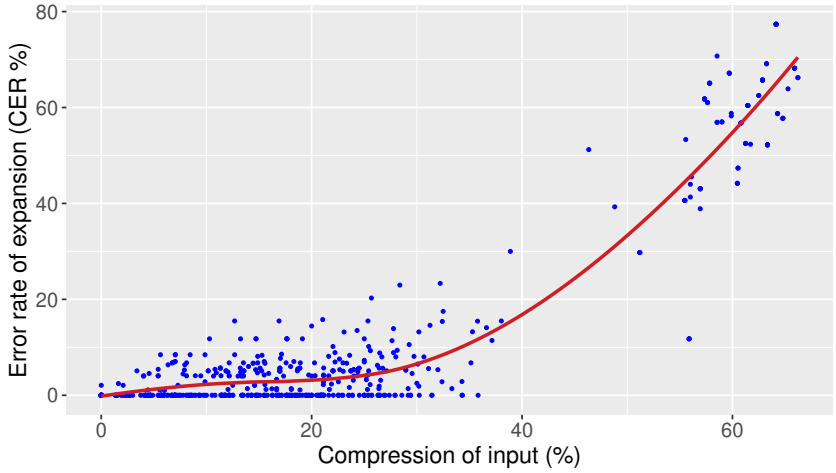

Figure 5.1: Error rate of automatic expansion with increasing abbreviation of the input.

5.4 Conversational Language Modeling

We think abbreviated input may most benefit users with slow input. From this point on, we focus on optimizing our system for use by AAC users. AAC users may not be able to speak due to a condition such as ALS. AAC users' slow input rate make taking part in conversations difficult [4]. Sentence abbreviation may be particularly useful for short phrases with predictable language.

Our search-based approach to abbreviation expansion relies crucially on a well-trained language model. For a language model to work well it needs to be trained on data that is suited to the target domain. Ideally we would train our language models on large amounts of conversational communications written by AAC users. For privacy and ethical reasons, it is difficult to find large amounts of such data. Therefore, in this section, we explore selecting training data from an out-of-domain dataset using a small amount of in-domain AAC-like data.

5.4.1 Selecting Training Data

As our in-domain set, we used 29 K words of AAC-like crowdsourced messages [123]. For our out-of-domain training set, we used one billion words of web text from Common Crawl[†]. We only kept sentences consisting of A–Z, apostrophes, spaces, commas, periods, question marks, and exclamation point. We compared three ways to select training sentences:

Random selection. We randomly selected sentences until we reached 100 million characters.

Cross entropy difference selection. Following Moore & Lewis [85], we trained an in-domain 4-gram word language model on our AAC-like data, and an out-of-domain 4-gram model on a random subset of web text (disjoint from the training set). We calculated the cross-entropy difference of training sentences using the in- and out-of-domain models. We selected the highest scoring sentences until we reached 100 million characters.

BERT selection. BERT is a language representation model built using self-attentive transformers [29]. We took the in- and out-of-domain data from the previous step and labeled each sentence based on its set. We then trained a binary classifier using `bert-base-uncased`[‡]. We ran our classifier on each sentence in the training set yielding the probability of a sentence belonging to the in-domain set. We selected the top sentences until we reached 100 million characters.

[†] https://commoncrawl.org/

[‡] https://github.com/google-research/bert/

Method	Words sent.	OOV (%)	Enron ppl	Daily ppl	Enron CER
Random	29.8	1.21	4.81	3.31	7.04
CE diff.	14.3	0.32	4.57	3.11	6.00
BERT	11.1	0.39	4.53	3.05	5.75

Table 5.1: Impact of selection method on training sentences and performance of letter language models.

5.4.2 Comparison of Selection Methods

As shown in Table 5.1, random sentences from Common Crawl averaged 30 words. The cross-entropy difference and BERT methods selected shorter sentences of 14 and 11 words respectively. This is likely good given our goal of supporting short, conversational messages. For comparison, sentences averaged 13 words in the in-domain AAC set and 10 words in DailyDialog [67]. DailyDialog consists of two-sided everyday dialogues.

We calculated the out-of-vocabulary (OOV) rate with respect to a vocabulary of 100 K words. Our randomly selected sentences had a much higher 1.2% OOV rate compared to cross-entropy and BERT selected data at 0.3% and 0.4% respectively (Table 5.1). Again this suits our purpose as we suspect abbreviated input is best suited for sentences without uncommon words. For comparison, the OOV rates of DailyDialog and our AAC-like set were both low at 0.2%. See the appendix for samples of sentences selected by each method.

We trained 12-gram character language models with Witten-Bell smoothing on each 100 million character training set. We trained without count cutoffs and did not prune the models. The binary BerkeleyLM [91] size of the random, cross-entropy difference, and BERT models were 1.7 GB, 1.3 GB, and 1.2 GB respectively.

We evaluated these character language models on the Enron mobile [122] and DailyDi-

alog [67] datasets. Before evaluation, we split each dialog turn in DailyDialog into single sentences and randomized their order. We calculated the average per-character *perplexity* of these two datasets. As shown in Table 5.1, the cross-entropy and BERT models had perplexities around 6% lower than the random model with the BERT model having the lowest perplexity.

We also compared the recognition accuracy of the three language models using the recognizer and data to be described in the next section. As shown in Table 5.1 (right column), these perplexities reductions did translate into improvements in recognition accuracy on touchscreen input where spaces and 50% of mid-word vowels were removed.

5.5 Recognizing Noisy Abbreviated Input

We now describe how we used our optimized language models to recognize noisy abbreviated input.

5.5.1 Decoder Details and Improvements

We extended the VelociTap touchscreen keyboard decoder [126]. VelociTap searches for the most likely text given a sequence of 2D taps. Each tap has a likelihood under a 2D Gaussians centered at each key. Taps can be deleted without generating a character by incurring a deletion penalty. Adding characters to a hypothesis incur penalties based on a character language model.

The decoder can insert characters without consuming a tap. A general insertion penalty allows all possibles characters to be inserted. The decoder also has separate space and apostrophe insertion penalties. We extend this further by adding a *vowel insertion penalty* for

inserting the vowels: `a`, `e`, `i`, `o`, `u`. However, this penalty is only used if the prior character is not a space. This models that vowels should not be skipped at the start of words.

The search is performed in parallel, with different threads extending partial hypotheses. When a hypothesis consumes all taps, it is added to an n-best list. To keep the search tractable, a configurable beam controls whether partial hypotheses are pruned. A wider beam searches more thoroughly, but at the cost of more time and memory.

To date, VelociTap has only used n-gram language models. We extend the decoder to use a recurrent neural network language model (RNNLM) either as a replacement for the character n-gram during search, or to rescore the n-best list. When used for rescoring, we compute the log probability of each sentence under the RNNLM. We multiply this probability by an *RNNLM scale factor* and add the result to a hypothesis' log probability.

We trained an RNNLM on the BERT-selected training data. After a hyperparameter search, we settled on 512 LSTM [48] units, a character embedding size of 64, two hidden layers, a learning rate of 0.001, and a dropout probability of 0.5. We trained using the Adam optimizer [57]. On the Enron Mobile and DailyDialog test sets, our RNNLM had a perplexity of 4.50 and 2.64 respectively.

To allow efficient hypothesis extension during RNNLM-based search, we augmented our partial hypotheses to track the state of the neural network. However, as we will see, RNNLM search required substantial memory and computation time. While we experimented with using a GPU for RNNLM queries, we found parallel CPU search was faster.

5.5.2 Touchscreen Data and Simulation Details

We tested our improvements on noisy, abbreviated, touchscreen keyboard input. We wanted noisy input to ensure our system was robust to mistakes AAC users may make when typing (e.g. when using a mouth stick or an eye-tracker). We created a test and development set using data collected on touchscreen phones [126, 127] and watches [120, 121]. We limited our data to sentences from the Enron Mobile set. We concatenated taps to create single sentence sequences without spaces. We removed sentences where the number of taps did not match the length of its reference. This resulted in a test and development set of 6,631 and 731 sentences respectively.

We played back taps to our decoder, deleting mid-word vowels with a given *vowel drop probability*. We tested drop probabilities of 0.5 and 1.0. In our test set, 17.7% of characters were spaces. With a drop probability of 0.5, 27.9% of characters (including spaces) were deleted. If all mid-word vowels were dropped, 38.2% of characters were saved. For the n-gram search and RNNLM rescoring setups and two drop probabilities, we tuned decoder parameters to minimize CER on the development set. Tuning used a random restart hill-climbing approach. We tuned each of the four setups for 600 CPU hours. Due to the computational costs, we used the parameters found for the n-gram search for the RNNLM search.

We report the character error rate (CER), as well as word error rate (WER), and sentence error rate (SER) on our test set. We also report the Top-5 SER which is the lowest SER of the top five hypotheses. We searched in parallel using 24 threads on a dual Xeon E5-2697 v2 server. This large number of threads mainly sped up the RNNLM search.

Decoder search	Drop prob.	CER (%)	WER (%)	SER (%)	Top-5 SER(%)	Decode time (s)	Mem (GB)
n-gram search	0.5	5.7 ± 0.3	12.4 ± 0.5	35.1 ± 1.1	22.0	0.21	40.7
+ RNNLM rescore	0.5	4.4 ± 0.2	9.5 ± 0.5	27.7 ± 1.1	16.5	0.34	52.1
RNNLM search	0.5	4.3 ± 0.2	9.3 ± 0.5	27.6 ± 1.1	15.4	24.05	353.2
n-gram search	1.0	9.5 ± 0.4	19.0 ± 0.7	45.5 ± 1.2	30.3	0.03	41.4
+ RNNLM rescore	1.0	8.0 ± 0.3	15.5 ± 0.6	38.5 ± 1.2	24.2	0.09	52.9
RNNLM search	1.0	8.2 ± 0.3	15.8 ± 0.6	41.5 ± 1.2	26.3	1.09	52.4

Table 5.2: Error rates and decoder performance using different search methods and vowel drop probabilities. ± values denote sentence-wise 95% bootstrap confidence intervals [11].

5.5.3 Recognition Results

As shown in Table 5.2, using the RNNLM in the search instead of the n-gram model reduced error rates by 23% and 12% relative for a vowel drop probability of 0.5 and 1.0 respectively. This however came at a much higher cost with decoding taking much longer and requiring more memory. Using the n-gram model for search and rescoring with the RNNLM resulted in similar error rates to searching with the RNNLM, but only caused modest increases in decode time and memory.

Dropping half of vowels, we recognized the correct sentence 72% of the time using RNNLM rescoring. If we assume an interface allowing selection from the top five results, this increased to 85%. Dropping all vowels was harder; we recognized the correct sentence only 59% of the time. Providing the top five sentences increased this to 74%.

Interestingly, our vowel drop probability 1.0 setups were faster. We investigated this by varying the tuned beams, measuring CER on the development set. We found for drop 0.5, a narrower beam increased CER while a wider beam provided no gain. For drop 1.0, a narrower beam also increased CER, but even a modestly wider beam increased CER slightly

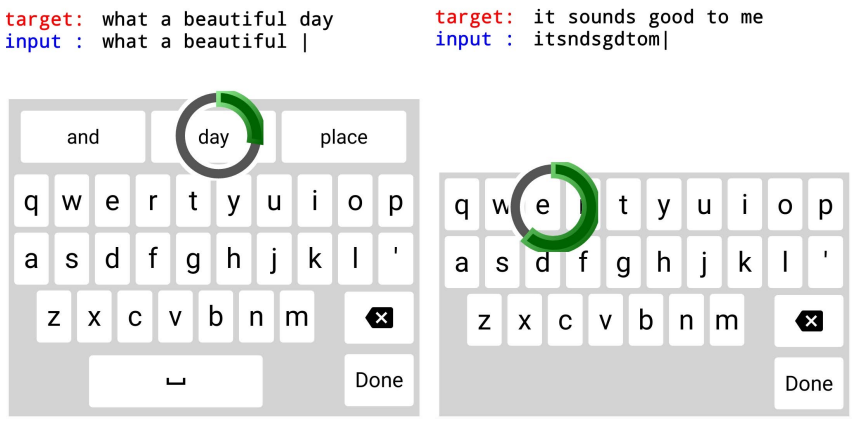

Figure 5.2: The word (left) and sentence (right) keyboard modes from our user study. The circle is centered on the user's touch location with a green arc showing progress towards the one second dwell time.

(3% relative). The tuned penalty for vowel insertion was small (0.8 probability). We observed in sentences with errors at a narrow beam, a wider beam sometimes resulted in more inserted vowels. This may have allowed more probable text, but ultimately a higher CER. This suggests we may need a more nuanced model of how users abbreviate, e.g. by penalizing contiguous vowel insertions.

5.6 User Study

Thus far, we tested abbreviated sentence input only in offline experiments. To see if our method offers competitive performance in practice, we conducted a user study using a touchscreen web application.

Metric	Entry rate (WPM)	Error rate (CER %)
Word	9.9 ± 1.5 [6.6, 12.4]	0.3 ± 0.5 [0.0, 2.5]
Sentence	9.0 ± 1.5 [5.7, 11.5]	7.2 ± 5.4 [1.0, 23.6]
Statistical test	$t(27) = -3.92$, $r = 0.60$, $p < 0.001$	$t(27) = 6.72$, $r = 0.79$, $p < 0.001$

Table 5.3: User performance in each condition in our user study. Results formatted as: mean ± SD [min, max].

5.6.1 Design

We designed a touchscreen keyboard that runs in a mobile web browser. The keyboard has two modes:

Word — This mode has the keys A–Z, apostrophe, spacebar, and backspace (Figure 5.2, left). The keyboard has three prediction slots above the keyboard. The left slot shows the exact letters typed. The center and the right slots show predictions based on a user's taps and any previous text. Predictions and recognition occur after each key press. Pressing the spacebar normally selects the left slot. Similar to the iPhone keyboard, if a user's input is noisy and we predict an auto-correction with high probability, we highlight this slot instead. In this case, pressing spacebar selects the auto-correction. A done button signals completion of a sentence.

Sentence — This mode is similar but has no spacebar or suggestion slots (Figure 5.2, right). Input is recognized only after the done button is pressed.

To simulate users with a slow input rate, users had to dwell on a key for one second to click it. We chose one second because this is a common default setting in dwell-based eye typing, for example, 1.2 seconds in Tobii Communicator. We display a progress circle around a user's finger location showing the dwell time. After a click, the keyboard border flashes and the nearest key is added to the text area above the keyboard.

Due to memory and computation requirements, we ran our decoder on a server at our university. The keyboard client makes requests to the server to recognize input. In word mode, at the start of each key press, we request predictions for the keyboard slots. In sentence mode, we request sentence recognition at the start of pressing the done button. By making the server request at the start of a key press, we effectively eliminated the need to wait for predictions. The average round trip time for requests in our user study was 0.41 s (sd 0.21) in the word mode and 0.58 s (sd 0.29) in sentence mode.

5.6.2 Procedure

We recruited 28 Amazon Mechanical Turk workers. The study took 30–40 minutes. Workers were paid $10. We also offered a $5 bonus for the fastest 10% of workers in each condition subject to having a CER below 5%. This was a within-subject experiment with two counterbalanced conditions: WORD and SENTENCE. The conditions used the word and sentence mode of the keyboard respectively.

Workers typed 26 phrases in each condition. The first two were practice phrases which we did not analyze. Workers wrote phrases written by people with ALS for voice banking purposes [23]. We used phrases with 3–6 words (1,182 total phrases). Workers received a random set of phrases and never wrote the same phrase twice.

5.6.3 Results

Table 5.3 and Figure 5.3 show results and statistical tests. We calculated *entry rate* in words-per-minute (WPM). We considered a word to be five characters including space. We measured the entry time from a worker's first tap until they finished dwelling on the done but-

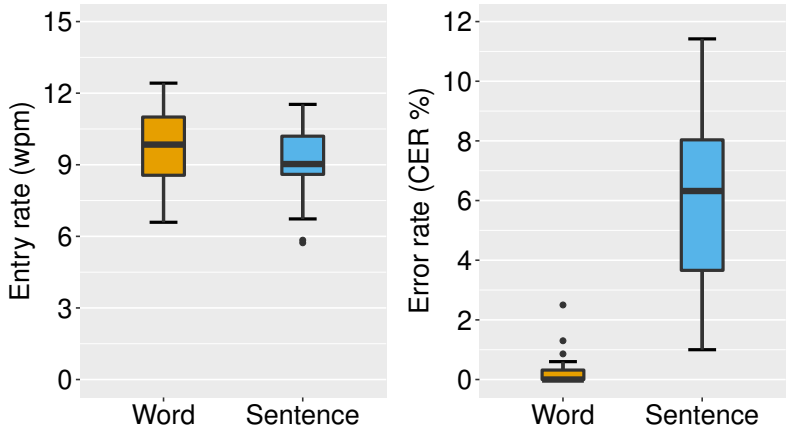

Figure 5.3: Entry and error rate in our user study.

ton. The entry rate in WORD was faster at 9.9 WPM versus SENTENCE at 9.0 WPM. This difference was significant (Table 5.3).

As shown in Figure 5.4, participants started out slower in SENTENCE compared to WORD, but the entry rate gap closed as they wrote more phrases. We averaged performance in the first eight and last eight phrases. In WORD, the entry rate was 9.7 WPM in the first set and 9.9 WPM in the last set. In SENTENCE, the entry rate was 8.6 WPM in the first set and 9.6 WPM in the last set. This is promising, as perhaps with more practice, sentence abbreviation might achieve comparable speed but without requiring monitoring of word predictions.

Participants were less accurate in SENTENCE with a CER of 7.2% versus 0.3% in WORD. This difference was significant (Table 5.3). Participants obtained a completely correct phrase 97% of the time in WORD, but only 68% in SENTENCE. We think the lower accuracy in SENTENCE was mostly due to some users abbreviating phrases too aggressively. In phrases recognized completely correctly, the compression rate was 35%. In phrases with

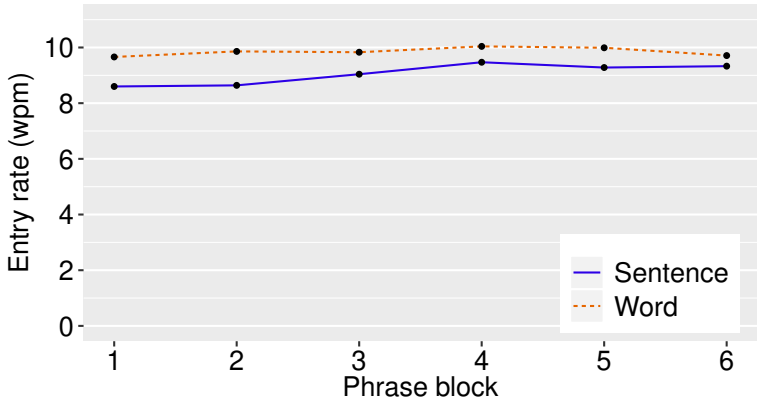

Figure 5.4: Entry rates for each block of four phrases.

recognition errors, the compression rate was 43%.

We classified phrases in SENTENCE according to their input length versus the reference length minus spaces and mid-word vowels. 252 phrases had the correct length, 162 were longer, and 258 were shorter. These sets correspond to phrases that were likely correctly abbreviated, under-abbreviated, and over-abbreviated. The error rates of these sets were 3.2%, 2.1%, and 14.0% respectively. We found five workers over-abbreviated 20 or more phrases. Removing these workers lowered the overall CER to 5.7%. While not as accurate as word input, sentence input did have acceptable accuracy when users abbreviated as instructed.

Individual user performance was variable (Figure 5.5). 16 participants achieved 0% CER in WORD and all but two had a CER below 1%. While in SENTENCE, no participant achieved 0% CER and five participants had a high CER of over 10%.

Using backspace, participants could fix incorrect letters or misrecognized words. The number of backspaces per final output character was low at 0.02 in both conditions. Thus, it appears participants precisely targeted keys, likely as a result of the slow input induced by the dwell time.

Figure 5.5: Participants' entry and error rate in each condition of the user study.

5.7 Discussion

We set out to show we could accelerate the writing of short and reasonably predictable phrases by combining sentence-at-a-time recognition with aggressive abbreviation. In our final user study, we found our method did not quite beat a conventional keyboard with word predictions. However, users in our study likely had substantial experience doing word-at-a-time input on their phones. It appears users got faster at abbreviated sentence input even during the brief study session. By the last eight phrases, users were only 3% relative slower using sentence abbreviated input compared to word-at-a-time input with word completions. When users provided abbreviated input consisting of all the correct letters except mid-word vowels, 90% of these phrases were expanded correctly.

We observed the abbreviation behaviour of a large number of non-AAC users and designed a system supporting the most common behaviours. While we could have tried to learn abbreviation behaviors from actual AAC user data, this presents a number of issues. First, actual text from AAC users is difficult to obtain for ethical and privacy rea-

sons. While it may be possible to obtain such text via donations from AAC users and from online sources, such sources lack visibility into how the user actually produced the text (e.g. did they use word completions?). Further, the propensity to abbreviate may be influenced by the particular AAC interface used. Second, even if we could source AAC abbreviated text, we would have no reliable way to determine the unabbreviated text. We could have asked AAC users to complete our abbreviation study, but this would have introduced more noise (incorrect key presses) that would have complicated our first study's goal of discovering natural abbreviation behaviors. It would have also limited the number of people we learned behaviors from. In this phase our goal was to discover what letters humans think are the most information carrying in a passage of text. While we suspect abbreviation strategies of AAC users would be similar, this would benefit from validation with AAC users.

We tested our method on touchscreen data recorded in previous studies on phones and watches, and in a web-based crowdsourced user study. We think our method mainly would benefit users who have a slow input rate; fast typists may only be slowed by the cognitive overheads of deciding what letters to omit or by disrupting their muscle memory for typing familiar words. This led us to limiting the input rate in our study by requiring users dwell for one second. While this study allowed us to confirm our abbreviation method is competitive with a conventional keyboard with word predictions, this needs validation with users with actual input rate limits. AAC user interaction may feature more imprecise key presses, more accidental key presses, and may introduce complications related to attending to word predictions (e.g. the "midas touch" problem in eye tracking). Further, we only tested one input rate, it is possible our method may be better or worse at different input speeds. We think our approach may also offer advantages for eyes-free text input, but this also needs

comparison against conventional eyes-free input approaches (e.g. iPhone's VoiceOver feature).

We investigated abbreviation by omitting mid-word vowels. We did not investigate other forms of abbreviation such as phonetic replacement (e.g. "you" → "u") or removal of consonants. Our model may benefit from more sophisticated modeling on how and when vowels are inserted (e.g. penalizing repeated vowel insertions). Ideally improved models would be based on data collected by users engaged in actual abbreviated input. As our results show, correctly inferring the intended sentences was challenging even when we asked users to obey a few simple behaviours, namely removing spaces and mid-word vowels. While an ideal system would support a wide-range of abbreviation behaviors and even adapt to individuals, we suspect this may be challenging given our current lack of training data on this task.

In our initial study, participants abbreviated email text that was displayed visually. An alternative approach would be to play audio of the text. While this might be a more realistic abbreviation task, it also presents practical challenges to participants such as remembering the text and spelling any difficult words. Perhaps an even more externally valid approach would be to have workers compose novel abbreviated sentences. This would require another step to obtain the unabbreviated compositions [35, 124]. Given we now have a competent initial system, it would be interesting to undertake such a data collection effort.

Our results suggest a simple correction interface based on selecting from the top sentences would often, but not always work. Designing an efficient and easy-to-use interface for correcting a few words within such sentence results would be interesting future work. This might be especially challenging to design for users with diverse motor abilities.

We used language models trained on only 100 M characters of text. While this allowed us to compare the efficacy of the language model types and decoder configurations, substantially more training data is available along with neural architectures that scale to large training sets, e.g. GPT-2 [97]. We suspect further recognition accuracy gains are possible for abbreviated, noisy input by incorporating such models. Further, we could likely obtain additional improvements from the n-gram model by training on more data and then pruning the model to reduce its size. We avoided doing this in this work to fairly compare the n-gram and RNN language models when trained on the same amount of text.

Our language model training data was drawn from Common Crawl. We used a corpus of AAC-like crowdsourced messages to select training sentences from Common Crawl. Other sources of training data such as Twitter or Reddit are likely more conversational in style. It would be interesting to investigate whether data selecting from a more targeted large-scale training source provides additional improvements in language modeling.

We did not specifically investigate how our method would support text containing difficult words such as acronyms or proper names. Users can often anticipate and alter their input behavior to avoid auto-correct errors, e.g. by force [131], by long pressing a key [121], or by switching to a precise input mode [32]. Similarly, our abbreviated input method needs a way to specify words that should not be expanded or auto-corrected.

At the onset, we did not know that our proposed abbreviation technique would be competitive to conventional word completion. The results from our user study tell us we need to make further improvements to our recognition, better train users to abbreviate in supported ways, and conduct a longitudinal evaluation. Further, testing an abbreviated input prototype with AAC users will undoubtedly lead to new insights. This paper is a first step

in producing a viable prototype for testing with users with rate-limited input abilities.

5.8 Conclusion

We explored accelerating text communication by abbreviated sentence input. We conducted a user study to learn how users abbreviate. We showed the efficacy of a neural classifier to select conversational-style training instances from a large text corpus. We found that dropping spaces and mid-word vowels can provide compression of sentences from 28% to 38%. Such abbreviated and noisy input can often be expanded correctly 59% to 72% of the time. We also showed how the accuracy of a statistical virtual keyboard decoder can be improved by using a neural language model to re-rank the top recognition results. Finally, after practice, users wrote only slightly slower using sentence abbreviated input at 9.6 words-per-minute compared to a conventional keyboard with word predictions at 9.9 words-per-minute. If a phrase was abbreviated by removing spaces and mid-word vowels, our system expanded the abbreviated input to the intended phrase 90% of the time.

6

Language Model Personalization

6.1 Introduction

In the previous chapters, to accelerate the input of a rate-limited user we have worked on finding a good dataset for training language models, investigated incorporating different statistical and neural language models, and made use of contextual data such as partner speech. We have also examined the number of prediction slots on the interface and explored

what number of suggestion slots is suitable for an AAC interface. We explored different systems that benefit from a set of generalized language models which work for every user. These models were also static and once we selected a model for a certain interface, it did not change with time. However, every person is different and has a unique style of writing. Moreover, while writing, people tend to repeat the same word or even the same phrase over the course of time. Therefore, it is desirable to have a model that can change and adapt to a user's writing behaviour with time.

In this chapter, we investigate adapting language models to a person's writing history for improved text entry performance. We conduct offline simulation experiments on a publicly available and chronological text dataset. We assume an ideal user entering text on a text entry interface and we make improvements to an error-tolerant decoder. We adapt various language models according to the user's past written text and show performance gains using keystroke savings.

6.2 Related work

Fowler et al. [34] explored the effects of language modeling with and without personalization on touchscreen keyboards. In order to explore the impact of language model personalization in touchscreen keyboard applications, they simulated a large group of individuals (taken from the Enron Corpus [58]) typing their messages in chronological order. The simulated user was made to provide sloppy input, but could take advantage of a word completion model. They showed a background language model can reduce typing word-error-rate from 38.4% to 5.7%. A personalized language model using a cache or exponential decaying cache further reduced this error rate to 4.6%.

Tanaka-ishii [110] compared four adaptive language models. These models are based on concepts originally proposed by Bell et al. [8] in the text compression domain and are similar to cache models originally proposed by Kuhn and De Mori [63] in the speech domain. The models are: 1) unigram cache, 2) move to front (MTF), 3) adaptive co-occurrence, and 4) adaptive n-gram: prediction by partial match (PPM). Among these four adaptive models they found PPM to be the best.

Clarkson et al. [21] provided two techniques for language model adaptation: mixture based language models and cache based models. Both models reduced perplexity significantly over a trigram model. For the mixture model, the training data was partitioned into several components and a trigram language model was constructed for each component. The final model was a linear combination of the component language models. For the cache model, a unigram cache and an exponentially decaying cache [21] were investigated.

Grave et al. [42] proposed a neural network adaptation method which is similar to the cache model [62]. It involved storing the recent hidden activations in memory and using them as representation for the context to retrieve their corresponding word. The model required no training, could be used on any pre-trained neural network, and could be scaled to thousands of memory cells.

Lee et al. [139] Focused on personalizing recurrent neural network language models (RNNLMs) using text collected from social network via crowdsourcing. In this approach, a crawler collected data from a user's social network. The data included the social network posts of the user, the posts of their friends, and the posts authored by many other different users. The personalized language model was based on these data.

Smart Compose [15] by Gmail provides real-time suggestions that helps users compose

emails quickly. Smart Compose was tested in different language modeling architectures. For example, encoding context via an encoder and then providing it to a language model, merging available context and then providing it to a language model, and sequence-to-sequence prediction. It was also tested with different neural models such as RNNs and transformers [116]. Personalization was achieved by training a light-weight n-gram language model with Katz-backoff [54] on each user's personal email data.

Li et al. [66] explored recurrent neural network model adaptation for conversational speech recognition. They proposed two adaptation models. The first model is a conversational cache model estimated by counting words in a conversation and further interpolated with a background unigram distribution estimated from a training text set. In the second model they trained a deep neural network on conversational transcripts to predict word frequencies given word frequencies from first pass decoded word lattices.

Li et al. [65] investigated two ways to adapt transformer based language models for automatic speech recognition. The first approach is fine tuning. In this approach, a transformer language model is trained on a large corpus of text. Then the model is adapted to a small target domain via fine-tuning. The second approach is a mixture of dynamically weighted models. In this approach, first a number of language models are trained on source and target domains. Then the models are fused together by linear interpolation and dynamic weighting.

King and Cook [56] trained a background language model on data from blog posts. Then they evaluated three personalization techniques: i) continue training the background language model on a user's text, ii) train a personalized model on a user's data and interpolate with the background model, and iii) prime the state of the background model with

text from the user. They trained both n-gram and LSTM based RNNLMs for background and user specific models. Perplexity results showed improved performance with all three techniques over the background model.

While the common practice in language model adaptation is to use two different models trained on one in-domain (the adaptation corpus) and one out-of-domain (background language model corpus) dataset, Wen et al. [132] proposed a RNNLM personalization paradigm involving a third dataset. This dataset was built from a user's social network data crawled from a crowdsourcing platform and was used to fine-tune a model. They showed that personalized RNNLMs trained using user oriented features reduced perplexity on the test set compared to the models trained on the previous two corpora.

From the past work it is clear that language model personalization is not new. But except from Fowler et al. [34], to our knowledge, no work has applied language model personalization on finger-touch based virtual keyboards. Most approaches also evaluated the personalized models based on perplexity which is an intrinsic evaluation metric. But intrinsic evaluations sometimes do not correlate with the downstream evaluation metrics, for example, with word error rate [16]. Our work, in this chapter, mostly resembles the work in Fowler et al. and we have improved upon their work by adding more adaptive models such as prediction by partial matching (PPM) and recurrent neural network language models (RNNLMs).

Fowler et al. evaluated their models on large scale human like input on touch keyboards using keystroke savings (KS) and word error rate (WER) metric. In this work, we used character error rate (CER) beside those two metrics. There are a couple of key differences between our work and theirs. The most pronounced difference is that we have used three

adaptive models and combinations of these models in comparison to their unigram cache only model. Whereas Fowler et al. only showed WER results with noisy input, we provided KS, WER, and CER results.

6.3 Language Model Personalization

6.3.1 Development and Test Sets

We used the recently released Enron email dataset [40] that Fowler et al. [34] curated. The released dataset contains text files arranged in two subsets: a development set and a test test. Each file in each subset represents text written by an Enron employee in a chronological order. Originally Fowler et al. used data from 90 users with 45 users in each of the dev set and test set. But they could not release data from one user due to a permission issue. For our simulation experiments, we stripped all non-alphabetical characters except the apostrophe and converted the text to lower case. We did not remove apostrophes like Fowler et al. since we thought using contractions in chat or email messages is very common.

We also divided the dataset into development set and evaluation set. We had the same 45 users' data in the development set and the same 44 users' data in the evaluation set. We used the development set for finding appropriate scale factors of the language models. During evaluation, Fowler et al. used each of the test users' entire text data. But we did not use a user's entire text during evaluation. One reason for doing so was that the test users had disproportional amount of text ranging from 3K words to 140K words. We thought evaluating on the disproportional amount of data might not reflect the true effect on the performance. We wanted to accurately measure the variability in performance among the different users with respect to the different adaptation models. This would have been a lot

more variable if each user had vastly different amounts of test data.

Hence, we found the user who had the minimum number of words in their text and selected the same amount of text for other users. We ended up with having 3040 words in each of the test users' text. We further divided each user's text into two parts. In the remaining part of this chapter, we would refer the first half of a user's text as seed text and the second half as the test text. During evaluation, we always evaluated on the second half of the users' text. However, sometimes we fed the seed text to the adaptive models, updated their states, and then used the updated models to evaluate on the test text. We would refer these models as primed models. When the models did not use the seed text, we would refer those models as models with a flat start.

6.3.2 Background Language Model

We used a 12-gram character language model that was trained on data collected from blog posts, forum, and twitter [125]. We chose the large 12-gram character language model*. We chose this language model because it was trained on texts that are similar to the texts we write in email communication and were similar to the Enron data. The 12-gram model had 40.9 M character n-grams and the training set contained 504 M words. Fowler et al. used a bigram word language model as the background model. The model contained 8.5M word n-grams. We also used a 6-gram character language model from the above source to compare the performance against the bigram word language model. The 6-gram character language model had 3.6 M character n-grams.

*https://keithv.com/software/mobiletext/

6.4 Experiments

In this section, we simulate touch input on a smartphone keyboard where an ideal user provides the touch input and selects a word prediction from the list of suggestions above the keyboard. We assume the ideal user only enters lower case characters and behaves in two ways. In the first case, the ideal user enters all the words of a sentence one-by-one. The user starts by entering a character of a word and observes the text in the suggestion slots. The user does not provide any incorrect input and is always assumed to choose a word from the offered list of word predictions if it matches with the user's intended word.

In the second case, we assume the ideal user is prone to making errors and might not hit a key correctly every time. But similar to the first case, even with spatial noise we assume the user selects the target word as soon as it is shown in one of the suggestion slots. Before we jump into the simulation experiments, let us first have a look at the various components of the simulation process.

Decoder. We used the VelociTap decoder [126] for simulating touch typing with Enron data. We had a program that used the decoder and the decoder handled the simulated touch input. For an input sentence, the program first mapped each letter of the sentence to the 2D coordinates of its corresponding key in a keyboard and considered it as a touch location. Touch locations could be noisy or without any noise. When touch locations were not noisy, the coordinates of a letter were deterministic and we told the decoder not to substitute this letter with any other character during the decoder's search. For noisy cases, we randomly sampled the touch coordinates of a letter from a 2D Gaussian distribution. To sample the touch coordinates we used a similar QWERTY keyboard Fowler et al. used ex-

cept we had the apostrophe key in the second row right beside the 1 key. The width of a single key was 6.16mm and the height was 9.42mm. The standard deviations of the 2D Gaussian distribution centered at each key were set to 1.97 mm in the x axis and 1.88 mm in the y axis.

Then touch locations were given as input to the VelociTap decoder. For each touch location, VelociTap would generate an n-best list of word hypotheses according to a touch model, the 12-gram character language model, and a dictionary of words. We used a dictionary containing 168K words. We choose this to match the size of the vocab used by Fowler et al. We used the parameters tuned on thousands of smartphone touch data described in Vertanen [119]. We used a n-best size of 100. Depending on the adaptation models we used, we further rescored the n-best list with only one or with an ensemble of the adaptation models. We set n-best size high so that we could have a list containing less probable words also. Having a less probable word in the n-best list is useful since if it is not in the n-best list it will not be rescored even if it is more probable according to the rescoring models. We also did not use the adaptation language models directly in the decoder's search like the 12-gram character language model. As we have seen in Chapter 1.2, using language models for example, a recurrent neural network language model during search would have been expensive.

The decoder would offer three word suggestions after each progressive touch input. When the touch input corresponded to the prefix of a word, one suggestion always provided the literal text mapped by the touch locations to the keys in the keyboard. The other two suggestions were populated from the n-best list. When there were no pending touch input, the decoder would offer the next most likely words given the previous words.

Adaptive language models. We considered several language models for personalization:

i) **Neural Language Model.** We trained a recurrent neural network language model (RNNLM) [10] with LSTM [48] units on the same training data that was used to train the n-gram background language model. We did a hyperparameter search on small amount of data to find the optimal parameters of the RNNLM. The final RNNLM had a character embedding size of 64, 512 LSTM units, one hidden layer, and a dropout rate of 0.55. We used the Adam optimizer [57] with a learning rate of 0.001. Although we used the RNNLM as an adaptive model, but since we did not backpropagate through the previous hidden states and update the states, the RNNLM was about ensembling a static model with weak adaptation via maintain the hidden states.

ii) **Unigram cache.** Since the number of words in our test files was only about 1.5K words, we did not limit the size of the cache. We also did not use an exponentially decaying cache since it performed similar to a unigram cache with infinite size in Fowler et al. [34]. For each of the simulation experiments in section 6.4.1 and 6.4.2, we considered two situations. First, the unigram cache had a flat start. We started from an empty cache and evaluated on the test files. Second, we primed the cache with the seed files and evaluated on the test files.

iii) **Prediction by partial matching (PPM).** We adapted the PPM implementation from Google [†] which was implemented following the algorithm used in Dasher [129]. We used a PPM order of six and a order of 12. Similar to the unigram cache,

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

we had the PPM models adapt on the test files and also adapt on the seed files.

Given a series of touch locations, VelociTap first searched for the probable word hypotheses and generated a list of 100 best word hypotheses. During search we only used the background language model. After we had the 100 best word hypotheses, we further asked the adaptive models to calculate the probability of each word in the list. Then we interpolated the probability scores from the background language model and the adaptive language models. We used only one adaptive model or a combination of adaptive models during this process. When we used the adaptive models, it re-ranked the 100 best word hypotheses. To determine the contribution of each adaptive models to the adaptation process, we required to know the mixture weights or scale factors of the different models. We used the text of six users from the development set (Section 6.3.1) to determine these scale factors.

Evaluation metrics. After a final list of 100 best word hypotheses were generated, the top three words were offered to the simulated user. During our simulation experiment, the simulated user made optimal use of any available suggestion slots to try and obtain the intended text. We further assumed that the simulated user did not use any backspace. If the simulated user did not get the desired word during input, we used the most likely word. Since the simulated user did not use backspaces, this left uncorrected words during the input process especially when touch locations had spatial noise. We used three metrics to evaluate the simulated user's performance. The first is character error rate (CER) of the final text. The second is word error rate (WER) of the final text. The third is keystroke savings. We calculated keystroke savings using the formula described in Equation 2.15. We assumed selection of a suggestion slot requires one keystroke and adds any following space.

	Flat start	Primed
Model	KS (%)	KS (%)
Background (6-gram)	39.3 ± 0.3	39.3 ± 0.3
Background (12-gram)	45.6 ± 0.3	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 + PPM-12 + RNNLM	48.7 ± 0.3	49.4 ± 0.3

Table 6.1: Keystroke savings (KS) on the Enron test data when all the adaptive models had a flat start versus when the models were primed. No spatial noise was applied to the simulated touches. $\pm$ values denote 95% user-wise bootstrap confidence intervals.

6.4.1 Simulating Smart Touch Keyboard Typing

In the first experiment, we simulated touch typing of an ideal user without any spatial noise. In this case, we assumed the user always touched their intended keys.

Results. Table 6.1 shows simulation results with no spatial noise and with three suggestion slots. We evaluated the noiseless test input with keystroke savings (KS). In this case, character error rate (CER) and word error rate (WER) is zero since there is no noise. Column 2 shows the results on the test data when all the adaptive models started without any adaptation. From the results, we can see that using a 12-gram background character language model we gained a maximum of 45.6% keystroke savings. If a lower order 6-gram background character language model was used, the keystroke savings deteriorated to 39.3%. We calculated the results with the 6-gram character language model since it was roughly equivalent to the bigram language model Fowler et al. used.

We used the adaptive models with the 12-gram character language model. When adap-

tive models had a flat start, a unigram cache and a recurrent neural network language model improved keystroke savings to 46.3% and 46.8%. When the unigram cache and the RNNLM were primed with a user's previous text, it improved the keystroke savings slightly to 46.7% and 46.9% respectively. But using PPM as the adaptation model, we had a substantial gain in the keystroke savings. A PPM with order six had a keystroke savings of 47.4% and with order 12 had a keystroke savings of 47.7% when started flat. The keystroke savings improved further to 48.2% and 48.6% when the models were primed with users' previous written text. An ensemble of unigram, RNNLM, and PPM order 12 had the highest keystroke savings at 48.7% with flat start and 49.4% with priming. For both these cases, the mixture weights were: 12-gram character language model 0.30, unigram cache 0.25, PPM-12 0.15, and RNNLM 0.30.

6.4.2 Simulating Noisy Smart Touch Keyboard Typing

In this simulation experiment, we simulated touch typing of an ideal user susceptible to making errors when selecting a target key.

Results. Table 6.2 shows the simulation results on the noisy touch input with three suggestion slots. The middle section of the table shows results when the adaptive models had a flat start and the bottom section shows results when the models were primed. We evaluated the noisy test input with keystroke savings (KS), character error rate (CER), and word error rate (WER) metrics. If we compare the results with the results from Table 6.1, we can see that keystroke savings dropped around 2% when spatial noise was introduced in the touch input. But similar to noiseless input, keystroke savings improved when an adaptive model or an ensemble of adaptive models were used. Again, improvement on keystroke savings

	Spatial noise		
Model	KS (%)	CER (%)	WER (%)
Background (6-gram)	37.3 ± 0.3	0.97 ± 0.04	2.02 ± 0.08
Background (12-gram)	43.6 ± 0.3	1.15 ± 0.06	2.17 ± 0.09
Background (12-gram) and flat start			
+ Unigram	45.1 ± 0.5	1.10 ± 0.05	2.01 ± 0.08
+ RNNLM	45.1 ± 0.3	1.12 ± 0.05	2.07 ± 0.09
+ PPM-6	45.6 ± 0.3	0.74 ± 0.04	1.51 ± 0.06
+ PPM-12	45.8 ± 0.3	0.75 ± 0.04	1.52 ± 0.06
+ Unigram + PPM-12 + RNNLM	46.9 ± 0.3	0.77 ± 0.04	1.53 ± 0.07
Background (12-gram) and primed			
+ Unigram	45.3 ± 0.5	0.89 ± 0.04	1.66 ± 0.06
+ RNNLM	45.2 ± 0.3	1.04 ± 0.05	1.91 ± 0.08
+ PPM-6	46.5 ± 0.3	0.68 ± 0.03	1.37 ± 0.06
+ PPM-12	46.9 ± 0.3	0.68 ± 0.03	1.36 ± 0.06
+ Unigram + PPM-12 + RNNLM	47.8 ± 0.3	0.69 ± 0.03	1.40 ± 0.06

Table 6.2: Keystroke savings (KS), character error rate (CER), and word error rate (WER) on the Enron test data when all the adaptive models had a flat start (middle section) versus when the models were primed (bottom section). Spatial noise applied to simulated touches. ± values denote 95% user-wise bootstrap confidence intervals.

with unigram and RNNLM were minimal when the models were primed (KS of 45.3% and 45.2%) versus when they had a flat start (KS of 45.1% and 45.1%). When the models were not primed, a six order PPM had a KS of 45.6%, a 12 order PPM had a KS of 45.8%, and an ensemble of a unigram, a RNNLM, and a 12 order PPM had a KS of 47.8%. The models had KS of 46.5%, 46.9%, and 47.8% respectively when they were primed with users' previous text. For the ensemble of models, the mixture weights were: 12-gram character language model 0.30, unigram cache 0.25, PPM-12 0.15, and RNNLM 0.30 during flat start and when the models were primed.

Figure 6.1 shows the graph on keystroke savings with noisy input comparing the back-

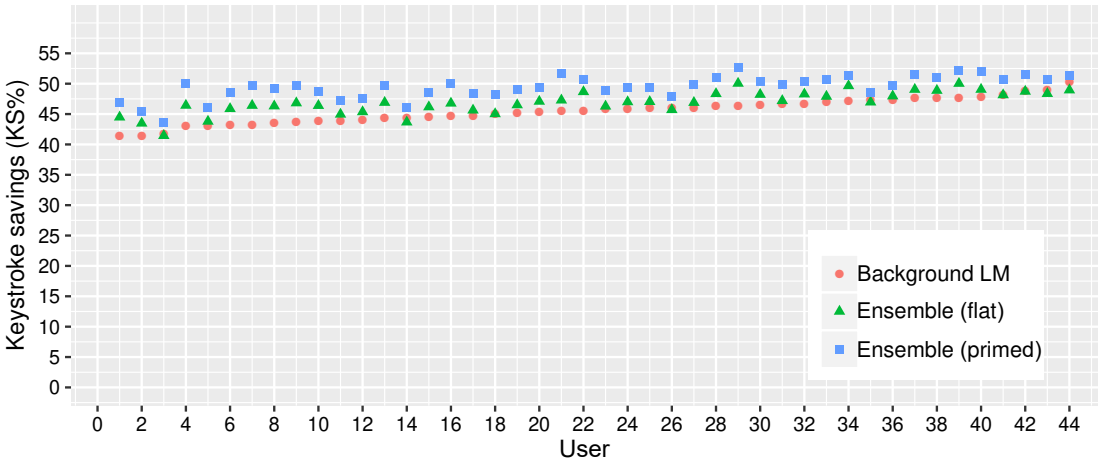

Figure 6.1: Graph showing keystrokes savings for each user in three different settings and with noisy touch data. Ensemble (flat) represents the combination of background language model, unigram cache, RNNLM, and PPM-12 with a flat start. Ensemble (primed) represents when the models were primed with the seed text. Users are sorted according to the keystroke savings using the 12-gram background language model.

ground language model versus the ensemble model with and without priming. It is clear from the graph that the ensemble model without priming beats the background only model and the ensemble model with priming beats the ensemble model without priming.

Recall how some uncorrected words were left in the entered text when spatial noise was introduced in the input. The uncorrected words contributed to character and word errors in the final text. When adaptive models were used, the character error rates and word error rates were low compared to the background only model. When adaptive models were used, the character error rates with and without priming were less than 1.12%. On the other hand, word error rates were less than 2.07%. We also calculated the literal character error rate and literal word error rate. Literal character error rate and literal word error rate represent the error rates if we used only the mapping of touch locations to keys in the keyboard

without any auto correction. The literal character error rate with the noisy touch input was 9.46% with a 95% CI of 0.02% and the literal word rate rate was 39.3% with a 95% CI of 0.27%. The literal word error rate is consistent with 38.4% that Fowler et al. reported. They did not report the literal character error rate. Our best adaptive model with primed data reduced the character error rate to only 0.69% and the word error rate to 1.40%. These are about 92.7% reduction from the literal character error rate and 96.4% reduction from the literal word error rate. As opposed to the character error rate and word error rate with the background 12-gram character language model, the reductions are about 40% and 35.5% respectively.

In our experiment, similar to Fowler et al. , we used three suggestion slots to offer word prediction to the ideal user. But in Chapter 4 we saw that we can increase the number of suggestion slots without affecting the text entry performance for able-bodied users with a slow input rate. We thought if we increase the number of suggestion slots, then how would that affect the performance in our noisy touch input simulation experiment. Therefore, we increased the number of slots to four and five and ran experiments with the best ensemble model we found during our evaluation. We also primed the ensemble model with seed text. With noisy input and with four suggestion slots, the ensemble model had a keystroke savings of 51.4% and with five suggestion slots the ensemble model had a keystroke savings of 53.7%.

6.5 Discussion

In this work, we tested various language model adaptation techniques on historical text data and simulated the behaviour of an ideal user entering text on a touchscreen keyboard.

We found that a simulated touch user can enter text accurately even if the user makes mistakes by not hitting the target key all the time. We also saw that using a model that adapts to the user's progressive text improves next word prediction. For best results, an ensemble of different adaptive language models such as a background language model, a unigram cache, an RNNLM, and a PPM model is desirable. But in case the RNNLM is costly to use in a system, acceptable performance can be achieved by accompanying PPM with varying orders with the background language model.

We also found that our best setting, an ensemble of different language models, had a keystroke savings of 49.4% with no noise and 47.8% with noise. These are about 8.3% and 9.6% increase respectively from using only the best background language model. Fowler et al. had a similar 8.6% relative gain using a cache model and with no noise. They did not report the relative gain on the keystroke savings with noisy input. Although we tested on different amount of text data compared to Fowler et al. , our small 6-gram background character language model was not on par with the background bigram model they used. But our large background model, a 12-gram character language model, as evidenced by the keystroke savings result was better than the bigram model. Our adaptive models further improved performance over this large and long-span background language model.

We found that the recurrent neural network language model when used as an adaptive model did not perform better than PPM. Even a lower order PPM which modeled only six previous characters performed better than the RNNLM. It could be for a couple of reasons. First, the RNNLM was trained on the same data the background 12-gram character language model was trained on. As such the effect of RNNLM as an adaptive model was more like a unigram cache and it showed similar performance to a unigram cache. Per-

haps training a RNNLM on more diverse and a different data set than the background model would improve performance further. Second, while using the RNNLM we did not reset the hidden states and carried on providing the previous hidden states as input to the next states. However, we did not backpropagate through the hidden states and update the weights. In our opinion, updating the hidden states and fine-tuning the RNNLM frequently with the user's written text might improve performance.

6.6 Conclusion

In this work, we conducted simulation experiments on touchscreen keyboards with noisy input. 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. Our results suggest that personalizing language models with the user's previous written text can improve performance. With noiseless touch input the best personalized model can achieve a 8.3% increase in keystrokes savings and with sloppy input can achieve a 9.6% increase in keystroke savings in comparison to using only a n-gram character language model. Also personalized models can reduce the character error rate and the word error rate by 36-40% as opposed to the background only model. In a real user interface, high keystroke savings and low error rates will be helpful to help users write fast without using backspaces or other corrective features too frequently.

7

Conclusion

7.1 Discussion

This dissertation presented a set of techniques to accelerate the input of a rate-limited AAC user. At first we looked at performing speech recognition on an AAC user's speaking partner. We tested three microphone deployment options and performed speech recognition on the recorded audio clips with two commercial speech recognizers namely Google and

IBM Watson. Then we investigated whether performing speech recognition on the partner speech and using the speech recognition results as context improves two-sided conversational language modeling. We found using partner speech context improves performance even with recognition word error rates of $7 - 16\%$. This improvement was nearly as good as when reference transcripts were used as context.

While using partner speech context provided promising results, there were no word prediction results. We evaluated the language models with partner speech context based on perplexity metric. It was not clear if it would work in the same way in an actual text entry application. To examine this we conducted experiments using a simulated user entering text on a keyboard. Before the simulation experiment we conducted a crowdsourced user study to determine the number of suggestion slots we should use. We had the users enter text on a web keyboard by dwelling on a key using a mouse pointer for one second. We offered different number of suggestion slots to the users. We found user performance was similar when 3–9 slots were used. From the study, we also estimated the average time the users spent tapping a key. Then we had a hypothetical user enter text on a keyboard. The hypothetical user was assumed to be taking part in a two-way conversation. To provide word predictions to the user we used several options: 1) a background language model and context from the already written text in that turn, 2) the background language model, a RNNLM trained on one-sided text from a dialogue, and context from the user's previous turns, and 3) the background language model, a RNNLM trained on both side's text from a dialogue, and context from the user's previous turns plus context from the other side's speech recognition results. We found partner speech context improved the user keystroke savings but the gain was small.

Next, we examined if we could reduce an AAC user's burden of entering all the letters in a sentence by allowing them to occasionally skip letters from the words. We conducted a crowdsourced user study to know how people abbreviate. Then inspired from past work and by analyzing the user behaviour in a crowdsourced study we decided to have user enter text by skipping mid-word vowels and space between words. We conducted simulation experiments to confirm that a sentence can be recovered even if we drop some or all mid-word vowels and spaces between words. We also proposed a deep neural based data selection technique to sample training data for a low resource target domain from Twitter. Finally, we conducted a crowdsourced study where able-bodied user entered text by dwelling over a key for a second to click it. They entered text by the standard word-by-word approach with suggestions and our abbreviation approach. After practice, users wrote only slightly slower using sentence abbreviated input at 9.6 words-per-minute compared to a conventional keyboard with word predictions at 9.9 words-per-minute in the last eight phrases. If a phrase was abbreviated by removing spaces and mid-word vowels, our system expanded the abbreviated input to the intended phrase 90% of the time.

Finally, we examined language model personalization. We collected a publicly available text dataset which contained a set of users' chronological written text. We simulated a hypothetical user who went through each of the test user's chronological written text and entered them progressively using a touchscreen keyboard. We further assumed the hypothetical user entering text with and without any error. We used a language model or an ensemble of language models to provide word predictions to the simulated user. We allowed language models to adapt to a test user's previous text. We found allowing language models to adapt to a test user's previous text improved the simulated user's keystroke sav-

ings rate. With noiseless touch input our best personalized model achieved a 8.3% increase in keystrokes savings and with noisy input achieved a 9.6% increase in keystroke savings in comparison to using only a n-gram character language model.

7.2 Future Work and Limitations

The series of work presented in this dissertation provides a few insights on accelerating text input for users who have speech- or motor-impairments. The ultimate goal of the series of research discussed here is to design an interface so that rate-limited users can input text faster and with ease. But there are limitations and the future work can focus resolving these limitations. Although at the end of each chapter we have already discussed about the limitations and scope of future work for each technique, below we discuss a few points in general:

The first and foremost limitation of our work is we did not involve rate-limited users in any of our user study. From a usability perspective it is desirable to test a system by the set of target users. However, directly involving rate-limited users in a study is difficult because it is hard to find such users and there are ethical and privacy concerns. In this dissertation, we tested ideas which are still in the primary phase. Before testing the ideas we did not know if the ideas would work or not. Therefore, we were hesitant to involve actual rate-limited users. Our results from different simulation experiments and user studies tell us that there is still room for improvement. Still we can develop a viable prototype based on what we have learnt so far. We can have this prototype tested by rate-limited AAC users.

Second, to simulate dwell we had able-bodied users hover on a key using a mouse pointer or press a key for one second. But a rate-limited AAC user typically uses eye-gaze or row-

column scanning with a switch to actuate a key. Therefore, our approaches need further validation. For example, instead of using a mouse pointer or a long finger press, it needs to be seen whether our interfaces work similarly with eye-gaze and dwell.

Third, we relied on remote servers. In Chapter 3, we relied on cloud based services such as Google Cloud Speech-to-Text and IBM Watson Speech-to-Text for speech recognition. Our text entry interfaces used in the user studies in Chapter 4 and chapter 5 relied on a decoder and we hosted this decoder in a remote server. Networking delays or service interruptions are very common and can negatively affect the users interacting with a text entry interface. For example, we had to replace participants in Chapter 4 who experienced networking delays. Since our text entry interfaces work with real-time input, it is expected that we provide real-time word predictions to the users. In some cases, a user might also want to use the device without network connectivity. For such cases, hosting a speech recognizer or the decoder in the user device will be more beneficial.

Fourth, we lacked appropriate data. The quality of word predictions specific to a target domain largely depends on data the predictive model was trained on. Since our text entry interfaces are mainly focused on accelerating input for AAC users, we expect our models to provide word predictions relevant to AAC users' text. But for making such word predictions the models need to be trained on text similar to the text the AAC users normally use. We did not have a good dataset representative of AAC users' text. We used text from different sources such as Common Crawl, Twitter, and Reddit. We even came up with a data selection technique in Chapter 5 to find AAC user-like text. Still these text data were not from actual rate-limited users. Future work can focus more on collecting AAC users' text. One way to obtain such text could be via donations from AAC users and stakeholders

(family members, speech and language pathologists, teachers, or physicians).

7.3 Final Remarks

In this dissertation, we examined different techniques for accelerating everyday text communication for people using alternative and augmentative communication (AAC) devices. Based on different user studies, simulation experiments, and empirical evidence we have shared different insights on designing AAC text entry interfaces. We believe these will be helpful to modify existing AAC text entry interfaces and will also work as a guide to future research directions.

A

Appendix

A.1 Abbreviation Study Instructions

In our first crowdsourced study in Chapter 5, we had 200 workers abbreviate a series of ten email messages. Figure A.1 shows the complete instructions we gave to workers.

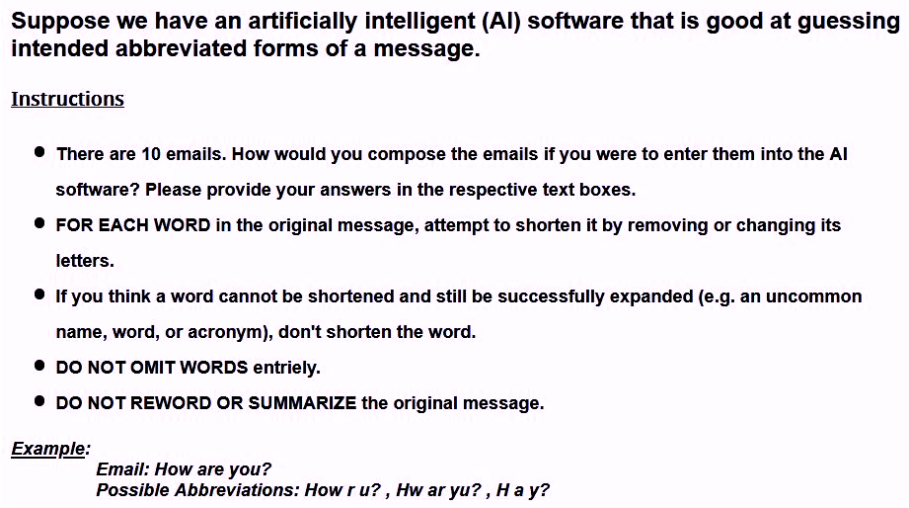

Figure A.1: Instructions given to workers in our free-from abbreviation crowdsourced study in Chapter 5.

A.2 Error Rate to Compression

In our final crowdsourced study in Chapter 5, we plotted participants' character error rate against increasing abbreviation in the SENTENCE condition. With increasing compression, the error rate also increased (Figure A.2).

A.3 Selecting Training Data

Table A.1 shows a list of example sentences selected using three ways described in Chapter 5: random selection, cross-entropy difference selection, and BERT selection.

A.4 Recognizing Noisy Abbreviated Input

Table A.2 shows some examples from our recognition experiments using the RNNLM rescoring configuration in Chapter 5.

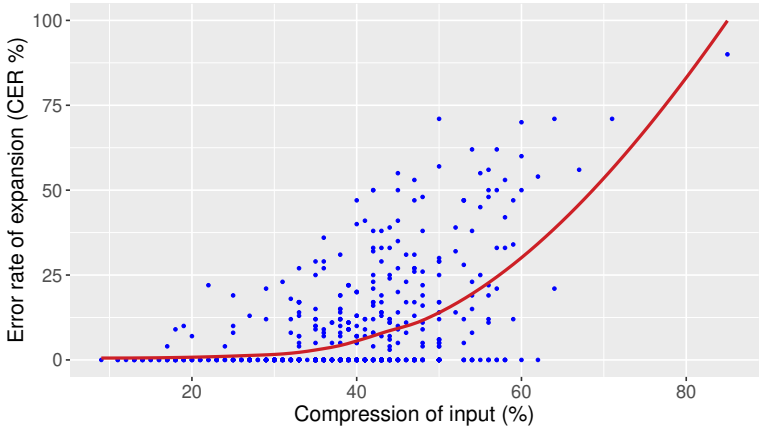

Figure A.2: Error rate of automatic expansion with increasing abbreviation of the input in the final study in Chapter 5.

Random selection	
Random 1:	i'm a huge fan of your work it's really well done.
Random 2:	the main challenge is to integrate more and more qubits to silicon chips.
Cross-entropy difference selection	
Top 1:	what's for dinner? what's for lunch?
Top 2:	how's things going?
Mid 1:	want to know what happened during fish robert ed fish's life?
Mid 2:	do you want to work at a job or do you want to play at a passion.
Bottom 1:	think super mario bros.
Bottom 2:	is there a form applicants should submit, or should they just send an email with their resume?
BERT selection	
Top 1:	you don't like doing homework?
Top 2:	you need money?
Mid 1:	i am very proud of you for what you are doing with your life right now.
Mid 2:	imagine my terrible position!
Bottom 1:	she has a bachelor's degree in political science from lincoln university in oxford, pennsylvania.
Bottom 2:	good relations are answers. bad relations are disasters.

Table A.1: Examples of selected text data using three different approaches described in Chapter 5. For BERT and cross-entropy difference selection Top, Mid, and Bottom represent the absolute positions in the ordered list according to their scores.

Vowel drop probability	Example	
0.5	Reference:	`hopefully this can wait until monday`
0.5	Input:	`hopfllltthisvamwwtujrlmndy`
0.5	Recognition:	`hopefully this can wait until monday`
0.5	Reference:	`let it rip`
0.5	Input:	`ltutrp`
0.5	Recognition:	`let it rip`
0.5	Reference:	`should systems manage the migration`
0.5	Input:	`shldsystensmnferhemgratin`
0.5	Recognition:	`she'd systems manager he migration`
1.0	Reference:	`could you see where this stands`
1.0	Input:	`cldyusewhtwrgsstnda`
1.0	Recognition:	`could you see where the stands`
1.0	Reference:	`florida is great`
1.0	Input:	`flrdaushrt`
1.0	Recognition:	`florida is great`
1.0	Reference:	`they are more efficiently pooled`
1.0	Input:	`yhyare'rrefgvmyluplf`
1.0	Recognition:	`they are more egg vinyl hold`

Table A.2: Abbreviated and noisy input and the resulting recognition results from Chapter 5. The input text represents the closest key to each tap observation in our data. Recognition errors are underlined.

References

[1] (2018). Augmentative and Alternative Communication (AAC). https://www.asha.org/public/speech/disorders/aac/. Accessed: 2018-11-23.

[2] (2022). The Eyegaze Edge. https://eyegaze.com/products/eyegaze-edge/. Accessed April 10, 2022.

[3] Agarap, A. F. (2018). Deep Learning using Rectified Linear Units (ReLU). *arXiv preprint arXiv:1803.08375*.

[4] Arnott, J. L., Newell, A. F., & Alm, N. (1992). Prediction and Conversational Momentum in an Augmentative Communication System. *Communications of the ACM*, 35(5), 46–57.

[5] Axelrod, A., He, X., & Gao, J. (2011). Domain Adaptation via Pseudo In-Domain Data Selection. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing* (pp. 355–362). Edinburgh, Scotland, UK.: Association for Computational Linguistics.

[6] Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural Machine Translation by Jointly Learning to Align and Translate. *arXiv preprint arXiv:1409.0473*.

[7] Baljko, M. & Tam, A. (2006). Indirect Text Entry Using One or Two Keys. In *Proceedings of the 8th international ACM SIGACCESS conference on Computers and accessibility* (pp. 18–25).

[8] Bell, T. C., Witten, I. H., & Cleary, J. G. (1990). *Text Compression*. Englewood Cliffs, NJ, USA: Prentice Hall.

[9] Bellegarda, J. R. (2004). Statistical Language Model Adaptation: Review and Perspectives. *Speech communication*, 42(1), 93–108.

[10] Bengio, Y., Ducharme, R., Vincent, P., & Jauvin, C. (2003). A Neural Probabilistic Language Model. *Journal of Machine Learning Research*, 3(Feb), 1137–1155.

[11] Bisani, M. & Ney, H. (2004). Bootstrap Estimates for Confidence Intervals in ASR Performance Evaluation. *Proceedings of the IEEE Conference on Acoustics, Speech, and Signal Processing*, (pp. 409–411).

[12] Brants, T., Popat, A. C., Xu, P., Och, F. J., & Dean, J. (2007). Large Language Models in Machine Translation. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*.

[13] Chen, B. & Huang, F. (2016). Semi-supervised Convolutional Networks for Translation Adaptation with Tiny Amount of In-domain Data. In *Proceedings of The 20th SIGNLL Conference on Computational Natural Language Learning* (pp. 314–323). Berlin, Germany: Association for Computational Linguistics.

[14] Chen, B., Kuhn, R., Foster, G., Cherry, C., & Huang, F. (2016). Bilingual Methods for Adaptive Training Data Selection for Machine Translation. In *Proceedings of the Association for Machine Translation in the Americas* (pp. 93–103).

[15] Chen, M. X., Lee, B. N., Bansal, G., Cao, Y., Zhang, S., Lu, J., Tsay, J., Wang, Y., Dai, A. M., Chen, Z., Sohn, T., & Wu, Y. (2019). *Gmail Smart Compose: Real-Time Assisted Writing*, (pp. 2287–2295). Association for Computing Machinery: New York, NY, USA.

[16] Chen, S. F., Beeferman, D., & Rosenfeld, R. (1998). Evaluation Metrics for Language Models.

[17] Chen, S. F. & Goodman, J. (1999). An Empirical Study of Smoothing Techniques for Language Modeling. *Computer Speech & Language*, 13(4), 359–394.

[18] Chen, X., Liu, X., Gales, M. J., & Woodland, P. C. (2015). Recurrent Neural Network Language Model Training with Noise Contrastive Estimation for Speech Recognition. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, ICASSP '15 (pp. 5411–5415).

[19] Chinea-Rios, M., Sanchis-Trilles, G., & Casacuberta, F. (2018). Creating the Best Development Corpus for Statistical Machine Translation Systems. In *Proceedings of the 21st Annual Conference of the European Association for Machine Translation* (pp. 99–108).: European Association for Machine Translation.

[20] Chung, J., Gulcehre, C., Cho, K., & Bengio, Y. (2014). Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. *arXiv preprint arXiv:1412.3555*.

[21] Clarkson, P. R. & Robinson, A. J. (1997). Language Model Adaptation Using Mixtures and an Exponentially Decaying Cache. In *IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 2 (pp. 799–802).

[22] Cleary, J. & Witten, I. (1984). Data Compression Using Adaptive Coding and Partial String Matching. *IEEE Transactions on Communications*, 32(4), 396–402.

[23] Costello, J. M. (2014). Message Banking, Voice Banking and Legacy Messages. *Boston Children's Hospital, Boston, MA*.

[24] Dai, A. M. & Le, Q. V. (2015). Semi-supervised Sequence Learning.

[25] Danescu-Niculescu-Mizil, C. & Lee, L. (2011). Chameleons in Imagined Conversations: A New Approach to Understanding Coordination of Linguistic Style in Dialogs. In *Proceedings of the 2nd Workshop on Cognitive Modeling and Computational Linguistics* (pp. 76–87).: Association for Computational Linguistics.

[26] Darragh, J. J., Witten, I. H., & James, M. L. (1990). The Reactive Keyboard: A Predictive Typing Aid. *Computer*, 23(11), 41–49.

[27] Demasco, P. W. & McCoy, K. F. (1992). Generating Text from Compressed Input: An Intelligent Interface for People with Severe Motor Impairments. *Communications of the ACM*, 35(5), 68–78.

[28] Demmans Epp, C., Djordjevic, J., Wu, S., Moffatt, K., & Baecker, R. M. (2012). Towards Providing just-in-time Vocabulary Support for Assistive and Augmentative Communication. In *Proceedings of the 2012 ACM International Conference on Intelligent User Interfaces* (pp. 33–36).: ACM.

[29] Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)* (pp. 4171–4186). Minneapolis, Minnesota: Association for Computational Linguistics.

[30] Diaz-Tula, A. & Morimoto, C. H. (2016). Augkey: Increasing Foveal Throughput in Eye Typing with Augmented Keys. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems* (pp. 3533–3544).

[31] Dolic, J., Pibernik, J., & Bota, J. (2012). Evaluation of Mainstream Tablet Devices for Symbol based AAC Communication. In *KES International Symposium*

on Agent and Multi-Agent Systems: Technologies and Applications (pp. 251–260).: Springer.

[32] Dudley, J. J., Vertanen, K., & Kristensson, P. O. (2018). Fast and Precise Touch-Based Text Entry for Head-Mounted Augmented Reality with Variable Occlusion. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 25(6).

[33] Duh, K., Neubig, G., Sudoh, K., & Tsukada, H. (2013). Adaptation Data Selection using Neural Language Models: Experiments in Machine Translation. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)* (pp. 678–683). Sofia, Bulgaria: Association for Computational Linguistics.

[34] Fowler, A., Partridge, K., Chelba, C., Bi, X., Ouyang, T., & Zhai, S. (2015). Effects of Language Modeling and Its Personalization on Touchscreen Typing Performance. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems* (pp. 649–658). New York, NY, USA: Association for Computing Machinery.

[35] Gaines, D., Kristensson, P. O., & Vertanen, K. (2021). Enhancing the Composition Task in Text Entry Studies: Eliciting Difficult Text and Improving Error Rate Calculation. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, CHI '21 New York, NY, USA: Association for Computing Machinery.

[36] Gao, J., Goodman, J., Li, M., & Lee, K.-F. (2002). Toward a Unified Approach to Statistical Language Modeling for Chinese. *ACM Transactions on Asian Language Information Processing*, 1(1), 3–33.

[37] Garcia, L., de Oliveira, L., & de Matos, D. (2014). Word and Sentence Prediction: Using the Best of the Two Worlds to Assist AAC Users. *Technology and Disability*, 26(2-3), 79–91.

[38] Gascó, G., Rocha, M.-A., Sanchis-Trilles, G., Andrés-Ferrer, J., & Casacuberta, F. (2012). Does More Data Always Yield Better Translations? In *Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics*, EACL '12 (pp. 152–161). USA: Association for Computational Linguistics.

[39] Ghosh, S. & Kristensson, P. O. (2017). Neural Networks for Text Correction and Completion in Keyboard Decoding. *arXiv preprint arXiv:1709.06429*.

[40] Google (2021). Enron Personalization Validation Set. https://github.com/google-research-datasets/EnronPersonalizationValidation. Accessed April 10, 2022.

[41] Grave, E., Joulin, A., & Usunier, N. (2016). Improving Neural Language Models with a Continuous Cache. *arXiv preprint arXiv:1612.04426*.

[42] Grave, E., Joulin, A., & Usunier, N. (2017). Improving Neural Language Models with a Continuous Cache. In *5th International Conference on Learning Representations, ICLR 2017, April 24-26, 2017*: OpenReview.net.

[43] Gregory, E., Soderman, M., Ward, C., Beukelman, D. R., & Hux, K. (2006). AAC Menu Interface: Effectiveness of Active versus Passive Learning to Master Abbreviation-Expansion Codes. *Augmentative and Alternative Communication*, 22(2), 77–84.

[44] Han, S., Wallace, D. R., & Miller, R. C. (2009). Code Completion from Abbreviated Input. In *Proceedings of the 2009 IEEE/ACM International Conference on Automated Software Engineering*, ASE '09 (pp. 332–343). USA: IEEE Computer Society.

[45] Higginbotham, D. J., Shane, H., Russell, S., & Caves, K. (2007). Access to AAC: Present, Past, and Future. *Augmentative and alternative communication*, 23(3), 243–257.

[46] Hildebrand, A. S., Eck, M., Vogel, S., & Waibel, A. (2005). Adaptation of the Translation Model for Statistical Machine Translation Based on Information Retrieval. In *Proceedings of the 10th EAMT Conference: Practical Applications of Machine Translation* (pp. 133–142). Budapest, Hungary: European Association for Machine Translation.

[47] Hochreiter, S. (1998). The Vanishing Gradient Problem during Learning Recurrent Neural Nets and Problem Solutions. *International Journal of Uncertain, Fuzziness and Knowledge-Based Systems*, 6(2), 107–116.

[48] Hochreiter, S. & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9(8), 1735–1780.

[49] Howard, J. & Ruder, S. (2018). Universal Language Model Fine-tuning for Text Classification. In *ACL*: Association for Computational Linguistics.

[50] Hur, B., Baldwin, T., Verspoor, K., Hardefeldt, L., & Gilkerson, J. (2020). Domain Adaptation and Instance Selection for Disease Syndrome Classification over Veterinary Clinical Notes. In *Proceedings of the 19th SIGBioMed Workshop on Biomedical Language Processing* (pp. 156–166). Online: Association for Computational Linguistics.

[51] Ioffe, S. & Szegedy, C. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *International Conference on Machine Learning* (pp. 448–456).: PMLR.

[52] Kane, S. K., Linam-Church, B., Althoff, K., & McCall, D. (2012). What We Talk About: Designing a Context-aware Communication Tool for People with Aphasia. In *Proceedings of the 14th international ACM SIGACCESS conference on Computers and accessibility* (pp. 49–56).: ACM.

[53] Kane, S. K. & Morris, M. R. (2017). Let's Talk About X: Combining Image Recognition and Eye Gaze to Support Conversation for People with ALS. In *Proceedings of the 2017 Conference on Designing Interactive Systems* (pp. 129–134).: ACM.

[54] Katz, S. (1987). Estimation of Probabilities from Sparse Data for the Language Model Component of a Speech Recognizer. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 35(3), 400–401.

[55] Kaufmann, T., Völker, S., Gunesch, L., & Kübler, A. (2012). Spelling is Just a Click Away–a User-centered Brain-computer Interface Including Auto-calibration and Predictive Text Entry. *Frontiers in Neuroscience*, 6, 72.

[56] King, M. & Cook, P. (2020). Evaluating Approaches to Personalizing Language Models. In *Proceedings of the 12th Language Resources and Evaluation Conference* (pp. 2461–2469). Marseille, France: European Language Resources Association.

[57] Kingma, D. P. & Ba, J. (2015). Adam: A Method for Stochastic Optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

[58] Klimt, B. & Yang, Y. (2004). The Enron Corpus: A New Dataset for Email Classification Research. In *Proceedings of the 15th European Conference on Machine Learning*, ECML'04 (pp. 217–226). Berlin, Heidelberg: Springer-Verlag.

[59] Kneser, R. & Ney, H. (1995). Improved Backing-off for M-gram Language Modeling. In *ICASSP*, volume 1 (pp. 181e4).

[60] Koester, H. H. & Simpson, R. C. (2014). Method for Enhancing Text Entry Rate with Single-switch Scanning. *Journal of Rehabilitation Research and Development*, 51(6), 995.

[61] Kristensson, P. O. & Zhai, S. (2004). SHARK^2: A Large Vocabulary Shorthand Writing System for Pen-based Computers. In *Proceedings of the 17th Annual ACM Symposium on User Interface Software and Technology*, UIST '04 (pp. 43–52). New York, NY, USA: ACM.

[62] Kuhn, R. (1988). 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*, COLING '88 (pp. 348–350). USA: Association for Computational Linguistics.

[63] Kuhn, R. & De Mori, R. (1990). Cache-based natural language model for speech recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12, 570–583.

[64] Levine, S., Gauger, J., Bowers, L., & Khan, K. (1986). A Comparison of Mouthstick and Morse Code Text Inputs. *Augmentative and Alternative Communication*, 2(2), 51–55.

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

[66] Li, K., Xu, H., Wang, Y., Povey, D., & Khudanpur, S. (2018). Recurrent Neural Network Language Model Adaptation for Conversational Speech Recognition. In *Proceedings of Interspeech 2018* (pp. 3373–3377).

[67] Li, Y., Su, H., Shen, X., Li, W., Cao, Z., & Niu, S. (2017). DailyDialog: A Manually Labelled Multi-turn Dialogue Dataset. In *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 1: Long Papers)* (pp. 986–995). Taipei, Taiwan: Asian Federation of Natural Language Processing.

[68] Lin, S.-C., Tsai, C.-L., Chien, L.-F., Chen, K.-J., & Lee, L.-S. (1997). Chinese Language Model Adaptation Based on Document Classification and Multiple Domain-Specific Language Models. In *Proceedings of European Conference on Speech Communication and Technology* (pp. 1463–1466).

[69] Lin, Y.-L., Wu, T.-F., Chen, M.-C., Yeh, Y.-M., & Wang, H.-P. (2008). Designing a scanning on-screen keyboard for people with severe motor disabilities. In K. Miesenberger, J. Klaus, W. Zagler, & A. Karshmer (Eds.), *Computers Helping People with Special Needs* (pp. 1184–1187). Berlin, Heidelberg: Springer Berlin Heidelberg.

[70] Lü, Y., Huang, J., & Liu, Q. (2007). Improving Statistical Machine Translation Performance by Training Data Selection and Optimization. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)* (pp. 343–350). Prague, Czech Republic: Association for Computational Linguistics.

[71] Ma, X., Xu, P., Wang, Z., Nallapati, R., & Xiang, B. (2019). Domain Adaptation with BERT-based Domain Classification and Data Selection. In *Proceedings of the 2nd Workshop on Deep Learning Approaches for Low-Resource NLP (DeepLo 2019)* (pp. 76–83). Hong Kong, China: Association for Computational Linguistics.

[72] MacKay, D. J., Ball, C. J., & Donegan, M. (2004). Efficient Communication with One or Two Buttons. In *AIP Conference Proceedings*, volume 735 (pp. 207–218).: American Institute of Physics.

[73] MacKenzie, I. S. (2012). Modeling Text Input for Single-Switch Scanning. In K. Miesenberger, A. Karshmer, P. Penaz, & W. Zagler (Eds.), *Computers Helping People with Special Needs* (pp. 423–430). Berlin, Heidelberg: Springer Berlin Heidelberg.

[74] Majaranta, P., Ahola, U.-K., & Špakov, O. (2009). Fast Gaze Typing with an Adjustable Dwell Time. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 357–360).

[75] Majaranta, P., MacKenzie, I. S., Aula, A., & Räihä, K.-J. (2006). Effects of Feedback and Dwell Time on Eye Typing Speed and Accuracy. *Universal Access in the Information Society*, 5(2), 199–208.

[76] Mansour, S., Wuebker, J., & Ney, H. (2011). Combining Translation and Language Model Scoring for Domain-Specific Data Filtering. In *International Workshop on Spoken Language Translation (IWSLT) 2011*.

[77] McNaughton, D. & Light, J. (2013). The iPad and Mobile Technology Revolution: Benefits and Challenges for Individuals Who Require Augmentative and Alternative Communication.

[78] Merity, S., Xiong, C., Bradbury, J., & Socher, R. (2016). Pointer Sentinel Mixture Models.

[79] Microsoft (2019). DSTC8 Reddit Corpus. https://github.com/microsoft/dstc8-reddit-corpus. Accessed April 10, 2022.

[80] Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient Estimation of Word Representations in Vector Space. *arXiv preprint arXiv:1301.3781*.

[81] Mikolov, T., Deoras, A., Kombrink, S., Burget, L., & Cernockỳ, J. (2011a). Empirical Evaluation and Combination of Advanced Language Modeling Techniques. In *Proceedings of the International Conference on Spoken Language Processing* (pp. 605–608).

[82] Mikolov, T., Joulin, A., Chopra, S., Mathieu, M., & Ranzato, M. (2014). Learning Longer Memory in Recurrent Neural Networks. *arXiv preprint arXiv:1412.7753*.

[83] Mikolov, T., Karafiát, M., Burget, L., Černockỳ, J., & Khudanpur, S. (2010). Recurrent Neural Network based Language Model. In *Eleventh Annual Conference of the International Speech Communication Association*.

[84] Mikolov, T., Kombrink, S., Burget, L., Černockỳ, J., & Khudanpur, S. (2011b). Extensions of Recurrent Neural Network Language Model. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, ICASSP '11 (pp. 5528–5531).

[85] Moore, R. C. & Lewis, W. (2010). Intelligent Selection of Language Model Training Data. In *Proceedings of the ACL 2010 Conference Short Papers*, ACLShort '10 (pp. 220–224). Stroudsburg, PA, USA: Association for Computational Linguistics Association for Computational Linguistics.

[86] Mott, M. E., Williams, S., Wobbrock, J. O., & Morris, M. R. (2017). Improving Dwell-based Gaze Typing with Dynamic, Cascading Dwell Times. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems* (pp. 2558–2570).

[87] Nel, E.-M., Kristensson, P. O., & MacKay, D. J. C. (2019). Ticker: An Adaptive Single-Switch Text Entry Method for Visually Impaired Users. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 41(11), 2756–2769.

[88] Nicolau, H., Rodrigues, A., Santos, A., Guerreiro, T., Montague, K., & Guerreiro, J. a. (2019). The Design Space of Nonvisual Word Completion. In *The 21st International ACM SIGACCESS Conference on Computers and Accessibility*, ASSETS '19 (pp. 249–261). New York, NY, USA: Association for Computing Machinery.

[89] Oken, B. S., Orhan, U., Roark, B., Erdogmus, D., Fowler, A., Mooney, A., Peters, B., Miller, M., & Fried-Oken, M. B. (2014). Brain-Computer Interface with Language Model–Electroencephalography Fusion for Locked-in Syndrome. *Neurorehabilitation and neural repair*, 28(4), 387–394.

[90] Parcheta, Z., Sanchis-Trilles, G., & Casacuberta, F. (2018). Data Selection for NMT using Infrequent n-gram Recovery. In *Proceedings of the 21st Annual Conference of the European Association for Machine Translation* (pp. 219–227).: European Association for Machine Translation.

[91] Pauls, A. & Klein, D. (2011). Faster and Smaller N-gram Language Models. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies - Volume 1*, HLT '11 (pp. 258–267). Stroudsburg, PA, USA: Association for Computational Linguistics.

[92] Pennington, J., Socher, R., & Manning, C. D. (2014). GloVe: Global Vectors for Word Representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)* (pp. 1532–1543).

[93] Peris, Á., Chinea-Ríos, M., & Casacuberta, F. (2017). Neural Networks Classifier for Data Selection in Statistical Machine Translation. *The Prague Bulletin of Mathematical Linguistics*, 108(1), 283–294.

[94] Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., & Zettlemoyer, L. (2018). Deep Contextualized Word Representations. *arXiv preprint arXiv:1802.05365*.

[95] Pini, S., Han, S., & Wallace, D. R. (2010). Text Entry for Mobile Devices Using Ad-Hoc Abbreviation. In *Proceedings of the International Conference on Advanced Visual Interfaces*, AVI '10 (pp. 181–188). New York, NY, USA: Association for Computing Machinery.

[96] Polacek, O., Mikovec, Z., Sporka, A. J., & Slavík, P. (2011). Humsher: A Predictive Keyboard Operated by Humming. In *The Proceedings of the 13th International ACM SIGACCESS Conference on Computers and Accessibility* (pp. 75–82).: ACM.

[97] Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (2019). Language Models Are Unsupervised Multitask Learners. *OpenAI blog*, 1(8), 9.

[98] Räihä, K.-J. (2015). Life in the Fast Lane: Effect of Language and Calibration Accuracy on the Speed of Text Entry by Gaze. In *IFIP Conference on Human-Computer Interaction* (pp. 402–417).: Springer.

[99] Räihä, K.-J. & Ovaska, S. (2012). An Exploratory Study of Eye Typing Fundamentals: Dwell Time, Text Entry Rate, Errors, and Workload. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 3001–3010).

[100] Roark, B., Beckley, R., Gibbons, C., & Fried-Oken, M. (2013). Huffman Scanning: Using Language Models within Fixed-grid Keyboard Emulation. *Computer speech & language*, 27(6), 1212–1234.

[101] Rousseau, A. (2013). XenC: An Open-Source Tool for Data Selection in Natural Language Processing. *The Prague Bulletin of Mathematical Linguistics*, 100, 73–82.

[102] Sampath, H., Indurkhya, B., & Sivaswamy, J. (2012). A Communication System on Smart Phones and Tablets for Non-verbal Children with Autism. In *International Conference on Computers for Handicapped Persons* (pp. 323–330).: Springer.

[103] Schuster, M. & Paliwal, K. (1997). Bidirectional Recurrent Neural Networks. *IEEE Transactions on Signal Processing*, 45(11), 2673–2681.

[104] Schwenk, H. & Gauvain, J.-L. (2005). Training Neural Network Language Models on Very Large Corpora. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing* (pp. 201–208).: Association for Computational Linguistics.

[105] Schwenk, H., Rousseau, A., & Attik, M. (2012). Large, Pruned or Continuous Space Language Models on a GPU for Statistical Machine Translation. In *Proceedings of the NAACL-HLT 2012 Workshop: Will We Ever Really Replace the N-gram Model? On the Future of Language Modeling for HLT* (pp. 11–19). Montréal, Canada: Association for Computational Linguistics.

[106] Shieber, S. M. & Nelken, R. (2007). Abbreviated Text Input using Language Modeling. *Natural Language Engineering*, 13(2), 165–183.

[107] Simpson, R., Koester, H., & LoPresti, E. (2006). Evaluation Of An Adaptive Row/Column Scanning System. *Technology and disability*, 18(3), 127–138.

[108] Stolcke, A. (2002). SRILM – An Extensible Language Modeling Toolkit. In *Seventh International Conference on Spoken Language Processing* (pp. 901–904). Denver, CO.

[109] Stolcke, A., Zheng, J., Wang, W., & Abrash, V. (2011). SRILM at Sixteen: Update and Outlook. In *Proceedings of IEEE Automatic Speech Recognition and Understanding Workshop*, volume 5 of *ASRU '11*.

[110] Tanaka-ishii, K. (2007). Word-Based Predictive Text Entry Using Adaptive Language Models. *Natural Language Engineering*, 13(1), 51–74.

[111] Tanaka-Ishii, K., Inutsuka, Y., & Takeichi, M. (2001). Japanese Text Input System With Digits. In *Proceedings of the First International Conference on Human Language Technology Research*.

[112] Tanaka-Ishii, K., Inutsuka, Y., & Takeichi, M. (2002). Entering Text with a Four-button Device. In *COLING 2002: The 19th International Conference on Computational Linguistics*.

[113] Trnka, K., McCaw, J., Yarrington, D., McCoy, K. F., & Pennington, C. (2008). Word Prediction and Communication Rate in AAC. *Telehealth and Assistive Technologies (Telehealth/AT)*, (pp. 19–24).

[114] Trnka, K., McCaw, J., Yarrington, D., McCoy, K. F., & Pennington, C. (2009). User Interaction With Word Prediction: The Effects Of Prediction Quality. *ACM Transactions on Accessible Computing (TACCESS)*, 1(3), 17.

[115] Tuisku, O., Majaranta, P., Isokoski, P., & Räihä, K.-J. (2008). Now Dasher! Dash Away! Longitudinal Study of Fast Text Entry by Eye Gaze. In *Proceedings of the 2008 Symposium on Eye tracking Research & Applications* (pp. 19–26).

[116] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is All You Need. In *Advances in Neural Information Processing Systems* (pp. 5998–6008).

[117] Vertanen, K. (2017). Towards Improving Predictive AAC using Crowdsourced Dialogues and Partner Context. In *ASSETS '17: Proceedings of the ACM SIGACCESS Conference on Computers and Accessibility (poster)* (pp. 347–348).

[118] Vertanen, K. (2019). Character Language Models, December 2019. https://imagineville.org/software/lm/dec19_char/. Accessed April 10, 2022.

[119] Vertanen, K. (2021). *Probabilistic Text Entry-Case Study 3*, (pp. 277–320). Association for Computing Machinery: New York, NY, USA, 1 edition.

[120] Vertanen, K., Fletcher, C., Gaines, D., Gould, J., & Kristensson, P. O. (2018). The Impact of Word, Multiple Word, and Sentence Input on Virtual Keyboard Decoding Performance. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '18 (pp. 626:1–626:12). New York, NY, USA: ACM.

[121] Vertanen, K., Gaines, D., Fletcher, C., Stanage, A. M., Watling, R., & Kristensson, P. O. (2019). VelociWatch: Designing and Evaluating a Virtual Keyboard for the Input of Challenging Text. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, CHI '19 (pp. 1–14). New York, NY, USA: Association for Computing Machinery.

[122] Vertanen, K. & Kristensson, P. O. (2011a). A Versatile Dataset for Text Entry Evaluations Based on Genuine Mobile Emails. In *Proceedings of the 13th International Conference on Human Computer Interaction with Mobile Devices & Services*, MobileHCI '11 (pp. 295–298). New York, NY, USA: ACM.

[123] Vertanen, K. & Kristensson, P. O. (2011b). The Imagination of Crowds: Conversational AAC Language Modeling using Crowdsourcing and Large Data Sources. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP'11 (pp. 700–711). Edinburgh, Scotland, UK.: Association for Computational Linguistics.

[124] Vertanen, K. & Kristensson, P. O. (2014). Complementing Text Entry Evaluations with a Composition Task. *ACM Transactions of Computer Human Interaction*, 21(2), 8:1–8:33.

[125] Vertanen, K. & Kristensson, P. O. (2021). Mining, Analyzing, and Modeling Text Written on Mobile Devices. *Natural Language Engineering*, 27, 1–33.

[126] Vertanen, K., Memmi, H., Emge, J., Reyal, S., & Kristensson, P. O. (2015). 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*, CHI '15 (pp. 659–668). New York, NY, USA: ACM.

[127] Vertanen, K., Memmi, H., & Kristensson, P. O. (2013). The Feasibility of Eyes-free Touchscreen Keyboard Typing. In *Proceedings of the 15th International ACM SIGACCESS Conference on Computers and Accessibility*, ASSETS '13 (pp. 69:1–69:2). New York, NY, USA: ACM.

[128] Wandmacher, T., Antoine, J.-Y., Poirier, F., & Départe, J.-P. (2008). SIBYLLE, An Assistive Communication System Adapting to the Context and Its User. *ACM Transactions on Accessible Computing*, 1(1), 6:1–6:30.

[129] Ward, D. J., Blackwell, A. F., & MacKay, D. J. C. (2000). 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, UIST '00 (pp. 129–137). New York, NY, USA: Association for Computing Machinery.

[130] Ward, D. J. & MacKay, D. J. C. (2002). Fast Hands-free Writing by Gaze Direction. *Nature*, 418(6900), 838.

[131] Weir, D., Pohl, H., Rogers, S., Vertanen, K., & Kristensson, P. O. (2014). Uncertain Text Entry on Mobile Devices. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '14 (pp. 2307–2316). New York, NY, USA: ACM.

[132] Wen, T.-H., Heidel, A., yi Lee, H., Tsao, Y., & shan Lee, L. (2013). Recurrent Neural Network based Language Model Personalization by Social Network Crowdsourcing. In *Proc. Interspeech 2013* (pp. 2703–2707).

[133] Willis, T., Pain, H., & Trewin, S. (2005). A Probabilistic Flexible Abbreviation Expansion System for Users with Motor Disabilities. In *Proceedings of the 2005 International Conference on Accessible Design in the Digital World*, Accessible Design'05 (pp.4). Swindon, GBR: BCS Learning & Development Ltd.

[134] Willis, T., Pain, H., Trewin, S., & Clark, S. (2002). Informing Flexible Abbreviation Expansion for Users with Motor Disabilities. In *Proceedings of the 8th International Conference on Computers Helping People with Special Needs*, ICCHP '02 (pp. 251–258). Berlin, Heidelberg: Springer-Verlag.

[135] Wills, S. & MacKay, D. (2006). DASHER – An Efficient Writing System for Brain-Computer Interfaces? *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 14(2), 244–246.

[136] Wisenburn, B. & Higginbotham, D. J. (2008). An AAC Application using Speaking Partner Speech Recognition to Automatically Produce Contextually Relevant Utterances: Objective Results. *Augmentative and Alternative Communication*, 24(2), 100–109.

[137] Wisenburn, B. & Higginbotham, D. J. (2009). Participant Evaluations of Rate and Communication Efficacy of an AAC Application using Natural Language Processing. *Augmentative and Alternative Communication*, 25(2), 78–89.

[138] Yasuda, K., Zhang, R., Yamamoto, H., & Sumita, E. (2008). Method of Selecting Training Data to Build a Compact and Efficient Translation Model. In *Proceedings of the Third International Joint Conference on Natural Language Processing: Volume-II*.

[139] yi Lee, H., Tseng, B.-H., Wen, T.-H., & Tsao, Y. (2017). Personalizing Recurrent Neural Network Based Language Model by Social Network. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 25(3), 519–530.

[140] Zhai, S. & Kristensson, P. O. (2008). Interlaced QWERTY: Accommodating Ease of Visual Search and Input Flexibility in Shape Writing. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '08 (pp. 593–596). New York, NY, USA: Association for Computing Machinery.

[141] Zhai, S., Sue, A., & Accot, J. (2002). Movement Model, Hits Distribution and Learning in Virtual Keyboarding. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '02 (pp. 17–24). New York, NY, USA: Association for Computing Machinery.

[142] Zhu, S., Luo, T., Bi, X., & Zhai, S. (2018). Typing on an Invisible Keyboard. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '18 (pp. 439:1–439:13). New York, NY, USA: ACM.