Investigating the effects of corpus and configuration on assistive input methods

by

Samuel Hindmarsh

A thesis
submitted to the Victoria University of Wellington
in fulfilment of the
requirements for the degree of
Master of Engineering
in Software Engineering.

Victoria University of Wellington
2014

Abstract

Assistive technologies aim to provide assistance to those who are unable to perform various tasks in their day-to-day lives without tremendous difficulty. This includes—amongst other things—communicating with others. Augmentative and adaptive communication (AAC) is a branch of assistive technologies which aims to make communicating easier for people with disabilities which would otherwise prevent them from communicating efficiently (or, in some cases, at all). The input rate of these communication aids, however, is often constrained by the limited number of inputs found on the devices and the speed at which the user can toggle these inputs. A similar restriction is also often found on smaller devices such as mobile phones: these devices also often require the user to input text with a smaller input set, which often results in slower typing speeds.

Several technologies exist with the purpose of improving the text input rates of these devices. These technologies include ambiguous keyboards, which allow users to input text using a single keypress for each character and trying to predict the desired word; word prediction systems, which attempt to predict the word the user is attempting to input before he or she has completed it; and word auto-completion systems, which complete the entry of predicted words before all the corresponding inputs have been pressed.

This thesis discusses the design and implementation of a system incorporating the three aforementioned assistive input methods, and presents several questions regarding the nature of these

technologies. The designed system is found to outperform a standard computer keyboard in many situations, which is a vast improvement over many other AAC technologies. A set of experiments was designed and performed to answer the proposed questions, and the results of the experiments determine that the corpus used to train the system—along with other tuning parameters—have a great impact on the performance of the system. Finally, the thesis also discusses the impact that corpus size has on the memory usage and response time of the system.

Acknowledgements

First and foremost, I'd like to thank to my supervisors, whose knowledge and wisdom was essential to my completing this thesis. Special thanks also to my family, who have provided me with support throughout my research. Finally, a very special thanks goes out to all my friends, colleagues, office mates and fellow researchers who have all provided ideas and suggestions along the road.

Contents

1	Introduction	1
1.1	Types of assistive input	3
1.1.1	Ambiguous keyboard	4
1.1.2	Word prediction	5
1.1.3	Text auto-completion	6
1.1.4	Other assistive input technologies	7
1.2	Motivation	8
1.2.1	Brain-computer interfaces	10
1.3	Goals	12
1.3.1	Efficiency targets	13
1.4	Chapter summary	14
1.4.1	Layout of thesis	14
2	Background	17
2.1	Assistive input systems	18
2.2	Assistive text entry	19
2.2.1	Ambiguous keyboards	20
2.2.2	Word prediction	24
2.2.3	Text auto-completion	26
2.3	Chapter summary	28
3	Ambiguous Input Method	29
3.1	Overview of design	30
3.1.1	Ambiguous keyboard	31

3.1.2	Sentence-level word prediction	32
3.1.3	Word auto-completion	33
3.2	Algorithm	33
3.2.1	Wordlist construction	34
3.2.2	Phraselist construction	36
3.2.3	System behaviour	39
3.3	Configuration	44
3.3.1	Flags	45
3.3.2	Configuration options	48
3.4	Considerations, limitations & issues	51
3.4.1	Response time	52
3.4.2	Memory limitations	53
3.4.3	Impact of corpus size	54
3.5	Chapter summary	59
4	Corpora	61
4.1	Corpus construction	62
4.2	Corpora used	63
4.2.1	enron	64
4.2.2	newsgroup	66
4.2.3	fanfiction	67
4.2.4	ota-bawe	69
4.2.5	wikipedia	70
4.3	Corpus resizing	71
4.4	Chapter summary	72
5	Evaluation	73
5.1	Evaluation methodology	74
5.1.1	Process	76
5.1.2	Default configuration	77
5.2	Experimental design	78
5.2.1	Inter-corpus evaluation	79

5.2.2	Single corpus evaluation	81
5.2.3	Personalized corpora evaluation	82
5.2.4	Keyboard split evaluation	84
5.2.5	Subsystem evaluation	87
5.2.6	Tuning parameters evaluation	89
5.3	Chapter summary	90
6	Results	91
6.1	Data preprocessing	91
6.2	Analysing results	93
6.3	Experimental results	94
6.3.1	Inter-corpus evaluation results	95
6.3.2	Single corpus evaluation results	101
6.3.3	Personalized corpora evaluation results	103
6.3.4	Keyboard split evaluation results	105
6.3.5	Subsystem evaluation results	107
6.3.6	Tuning parameters evaluation results	111
6.4	Chapter summary	113
7	Conclusions	115
7.1	Contributions	116
7.2	Future work	118
7.3	Final thoughts	119

List of Figures

3.1	Design of the assistive input system	31
3.2	A simple example of a wordlist	35
3.3	A simple example of a phraselist	38
3.4	Relationship between memory usage and corpus size	56
3.5	Relationship between average response time per character and corpus size	57
6.1	Pseudo-medians for inter-corpus experiments with confidence intervals	97
6.2	Per-phrase relationship between average KSPC and phrase length (alpha level denotes density)	99
6.3	Pseudo-medians for single corpus experiments with confidence intervals	102
6.4	Pseudo-medians for inter-corpus experiments with confidence intervals in KSPC (the blue bars indicate personalized corpora)	103
6.5	Pseudo-medians for keyboard splits experiments with confidence intervals	106
6.6	Pseudo-medians for subsystem experiments with confidence intervals	108

List of Tables

3.1	Memory usage of varying corpus sizes	55
3.2	Average response time per character of varying corpus sizes	57
5.1	Evaluation matrix for inter-corpus evaluation (the grey cells indicate control experiments)	80
5.2	Categorization of corpora	80
5.3	Keyboard splits investigated	86
6.1	Summary statistics for inter-corpus experiments in KSPC (the grey rows indicate control experiments) .	95
6.2	Summary statistics for single corpus experiments in KSPC	101
6.3	Summary statistics for individualized corpora experiments in KSPC (the grey rows indicate cross-validated personalized experiments)	104
6.4	Summary statistics for keyboard split experiments in KSPC	105
6.5	Summary statistics for subsystem experiments in KSPC	107
6.6	Summary statistics for bad word behaviours in KSPC	111
6.7	Summary statistics for phrase window sizes in KSPC	111
6.8	Summary statistics for minimum phrase frequency thresholds in KSPC	112

6.9	Summary statistics for minimum word frequency thresholds in KSPC	112
-----	---	-----

List of Hypotheses and Their Conclusions

H.1	Corpora pairings which share the same domain will outperform those from different domains	80
C.1	Corpora pairings from the same domain exhibit improved performance, but not to the same extent that the difficulty of the corpus impacts performance	98
H.2	Corpora pairings which exhibit similar writing styles will outperform those with different styles	81
C.2	Corpora pairings which share the same writing style do not exhibit better performance when compared to other corpora pairings . .	98
H.3	A corpus taken from a smaller domain will perform better than one taken from a larger domain . .	82
C.3	The size of the domain from which the corpus is constructed has no impact on the performance of the corpus	102
H.4	A corpus which has been personalized to a user will allow the user to obtain better performance from the system when compared with a general corpus .	84
C.4	A personalized corpus allows a user to enter text at considerably higher rates when compared with a general corpus	105

H.5	The keyboard split of the ambiguous keyboard has a significant impact on the performance of the system	86
C.5	The performance of the system is heavily dependent on the way in which the ambiguous keyboard is split	106
H.6	An optimal unordered keyboard split can achieve better performance than an optimal ordered split . .	86
C.6	An unordered split is able to achieve higher performance when compared to an ordered split	107
H.7	The ambiguous keyboard has a negative impact on the performance of the system	88
C.7	The performance of the system is very negatively impacted by the ambiguous keyboard	109
H.8	Sentence-level word prediction greatly increases the performance of the system	88
C.8	The performance of the system is very positively impacted by sentence-level word prediction	110
H.9	Word auto-completion greatly increases the performance of the system	89
C.9	The performance of the system is positively impacted by word auto-completion, but only when paired with sentence-level word prediction; otherwise, the performance is negatively impacted	111
H.10	Changing the values and behaviours of the different parameters will have an effect on the performance of the system	90

C.10	The tuning parameters have a moderate impact on the performance of the system, but to a lesser extent than the training and testing corpora	113
------	---	-----

Chapter 1

Introduction

Assistive technologies are designed to improve the quality of life of people with disabilities, or are working in situations where normal operation is restricted. Such technologies can make many tasks easier, such as hearing (through the use of hearing aids), reading (through the use of text-to-speech systems) or walking (through the use of a mobility scooter). An assistive technology can vary in complexity from something as simple as a screen magnifier to something far more complicated, such as brain-computer interfaces.

A common category of assistive technologies is assistive input methods—a type of technology for assistive and augmentative communication (AAC)—which enable communication for people who are unable to communicate (or would have a large amount of trouble communicating) otherwise. In the health domain, assistive input methods are designed to aid people with a wide range of disabilities, from full-body muscular diseases (such as amyotrophic lateral sclerosis (ALS) or locked-in syndrome) to speech disabilities (such as verbal dyspraxia). Assistive input methods can be used to aid in entering text into a computer system, which—depending on the disability—can then be used to control the com-

puter system or processed using a speech synthesis engine in order to communicate with others directly through natural language.

While assistive input methods greatly benefit people who would be otherwise unable to communicate, the technologies can also aid able-bodied people in situations where regular typing systems are impractical. A common scenario where these methods are useful is encountered by many people on a daily basis in the form of text entry on a mobile device. Due to the limited space afforded by many of these devices (especially in the pre-smartphone era), a full keyboard is often impractical; instead, many of the smaller phones opt for a reduced keypad containing only 12 keys. Even today, many people struggle with entering text using a full keyboard on a smaller device; software which emulates the 12-key input system of older devices continues to offer an alternative input method for these users. The T9 system [30] found on many of these devices is an example of an assistive input method designed for users of these devices.

Assistive input methods—including those found on mobile devices—often demonstrate much lower performance than a standard full-sized keyboard, and as a result, can be frustrating to use for many users. This thesis aims to design an assistive input method which improves on the performance of existing methods by providing performance comparable to that of a standard keyboard. Furthermore, this thesis will also look at the impact of various tuning parameters and configuration options on the performance of the system, and design a set of experiments to determine the extent of this.

1.1 Types of assistive input

An assistive input method is an assistive technology designed to aid the entry of text in any situation, whether the aim is to enter text faster or easier using either a limited interface (i.e. a limited set of keys or other inputs) or a standard, full keyboard.

Ambiguous keyboards are designed for the former situation by mapping multiple characters to a single key or input, attempting to provide users with a method of typing efficiently with a limited key set.

Sentence-level word prediction is a technique which aims to improve the typing speed of a user with a full keyboard by suggesting words which the user may wish to type before the word has been fully entered. As per the name, the suggested words are generated from the context in which the user is typing—that is, they are based on the words previously entered as part of the current sentence.

Word auto-completion is a technique similar to word prediction, in that it also attempts to predict the word the user desires before he or she completes entering it; however, the primary difference is that the word is automatically entered, allowing the user to accept it with a single keystroke and continue typing the following word. Along with word prediction, word auto-completion is designed to improve the text entry rates of standard keyboards (although both methods can be equally used with limited input keyboards). Several other assistive inputs exist for improving text entry rates, including auto-correct and auto-fill systems; however, these were not considered for this research

1.1.1 Ambiguous keyboard

An ambiguous keyboard is a common assistive input method which allows us to map multiple letters to a single input key, with an internal speller model being used to determine which mapped letter is desired [30]. An ambiguous keyboard allows users to type using the full range of keyboard keys in systems with a limited set of inputs, making it impossible to achieve a one-to-one mapping between letters and inputs. Additionally, the system allows entering a single letter with a single keystroke, with a disambiguation algorithm used to determine the desired character. The fewer inputs available to a system naturally results in a larger number of characters mapped to each input key; where possible, it is desirable to increase the number of inputs into the system, as this will improve the performance of an ambiguous keyboard. It is naturally not possible to increase the number of inputs in situations where the inputs are defined by hardware, but occasionally this number is defined in software as a tunable parameter—for example, in various paradigms in the field of brain-computer interaction (BCI; see Section 1.2.1).

An ambiguous mapping is usually required to be designed in such a way that minimizes the cognitive load imposed upon the user when mentally mapping letter to the input. This is often achieved by requiring that the letters are kept in order across the inputs. This requirement can be relaxed in certain situations where the underlying mapping (and therefore the keyboard ambiguity) is hidden from the user. In these situations, the keyboard can, in fact, be presented naturally (for example, in alphabetical order or in the QWERTY format).

The aforementioned T9 keyboard—often called *predictive text*—allows users to type and select one of several words associated with an input sequence: for example, entering the key sequence

63 (associated with the letter groups (mno) and (def), respectively) on a standard T9 keyboard will suggest both “me” and “of” as matches. The mapping of multiple words to a single key sequence (resulting from mapping multiple letters to a single key) is hence why the input method is referred to as an ambiguous keyboard. The T9 keyboard is provided as an alternative to the standard multi-tap keyboard found on many older mobile phones, which requires several taps to enter in certain characters—despite also mapping multiple letters to a single key, this keyboard is not ambiguous, as the number of key presses determines the desired letter as the word is being typed (rather than presenting a list of matches to the user at the conclusion of word input).

1.1.2 Word prediction

While an ambiguous keyboard aims to offer as close as possible to the same performance as a traditional keyboard with fewer keys, word prediction aims to outperform standard input methods regardless of the number of available inputs [16, 78]. Input methods incorporating word prediction achieve such a performance boost by reducing the number of keystrokes required to input a word by suggesting a list of words the user may be attempting to input before he or she has finished typing the word. For example, if the user enters the character sequence “typ”, the speller may predict and present “typing” as a suggestion. This allows the user to complete the word with fewer keystrokes, resulting in a total cost of four or five (depending on implementation) keystrokes for a six letter word. By continually and accurately suggesting words for the user to insert, the user is potentially able to input text at a much higher rate.

English text tends to contain structures known as *collocations*—

sequences of words which occur frequently together. Sentence-level word prediction is a subclass of word prediction systems which offer predictions based on words previously typed in the current sentence. These systems are able to benefit from the existence of collocations by taking the immediate context into account when sorting suggestions. By considering the previously entered words, the system is able to rank the suggested words based on the frequency at which they follow the entered sequence. Naturally, a predictive speller can only make suggestions based on what the system has seen before, and therefore the usefulness of its suggestions is based entirely on the data originally shown to the system to train it.

1.1.3 Text auto-completion

The technique of text auto-completion refers to automatically inserting text, allowing the user to skip a certain amount of typing and continue naturally on to the following text [69]. Word auto-completion is a specific type of auto-completion which completes a word based on the previously entered letters in the current word. There is no restriction on where candidates for auto-completion are taken: they can be sourced from a list of suggestions provided by a word prediction technique, a list of matches from an ambiguous keyboard, or from some other source entirely. The technique which arguably benefits the most from auto-completion is sentence-level word prediction; in this pairing, the first letters of the next word need not be entered before a suggestion can be auto-completed. With the added benefit of collocations, this allows a user to enter an entire phrase by only entering the first letters of the first word.

Text auto-completion is designed to improve typing speeds;

however, it is difficult to measure the true impact of auto-completion on a user's ability to type, as the input method introduces a new source of cognitive load [31]. For this reason, the raw keystroke savings do not always reflect the true performance increase when used by a human operator, as each keystroke must be more carefully considered before being entered. Despite this, the number of keystrokes saved using auto-completion is high enough to justify continued use of it as an input method, at least in the case of word auto-completion [34, 56].

1.1.4 Other assistive input technologies

Keyboards featuring auto-correct systems [15] have gained in popularity greatly with the rise of the smartphone. Since the release of Apple's iPhone in 2007, the majority of keyboards used on smartphones are able to detect misspellings and correct them without user intervention. This allows the user to type at much higher rates, as it is not necessary to obtain 100% keypress accuracy. However, auto-correct systems are themselves not always accurate, and can erroneously replace words entered by the user if the word is not in the keyboard's internal dictionary; this can be a source of frustration for the user. For this reason, the user of an auto-correcting keyboard must constantly check to ensure the words entered have not been changed by an overzealous auto-correct system, which introduces an additional cognitive load. Auto-correct was not considered for this research due to the difficulty in measuring the performance benefit of the system: the system relies on user error, which is dependent on more variables than simply the text being entered into the system.

Another type of assistive input is the auto-fill functionality of many modern web browsers. Because many text forms on the

Internet require the user to enter his or her personal details, modern web browsers can store this (and other) frequently-entered information for the user and automatically fill forms on other web pages. This prevents the need for users to constantly type the same information. However, auto-fill is not useful in general text input situations, and was therefore not considered for this research.

1.2 Motivation

At the beginning of this chapter, the concept of assistive technologies was introduced as being especially beneficial to people with disabilities. Assistive input systems can benefit such people by providing a means for them to communicate with others—a potentially difficult or even impossible task otherwise, depending on the extent of their disabilities. Assistive and augmentative communication (AAC) systems often provide a specially-designed interface to allow disabled people to communicate—examples of these interfaces include gaze tracking systems, which allow text entry by tracking eye movement as the user looks at targets, and sip-and-puff systems, which gives the user binary control over a selection mechanism. These hardware interfaces severely hamper text entry due to the time required to enter a single character.

The limited rate at which a user can enter text using these systems can be a major source of frustration, especially when it limits the amount of text they are able to comfortably input. By reducing the number of (what are effectively) keystrokes required to enter an arbitrary amount of text, it is possible to increase the amount of text which a user of one of these technologies is able to comfortably enter in a session. It is this reduction in keystrokes that the word prediction and auto-completion aspects of the assistive input system aim to achieve. Furthermore, the performance of

many of these systems are dependent on the number of different characters that can be entered; for example, the gaze tracking system is more accurate the fewer selections it must differentiate between, and the sip-and-puff system is more efficient the fewer characters it must cycle through. To reduce the number of selections, the ambiguous keyboard is used.

Earlier in this chapter, mobile devices were presented as another of the major motivations for this research. The T9 predictive text entry system was earlier introduced as a common text entry method for traditional mobile phones (that is, non-smartphones). These phones are, for the most part, required to use ambiguous keyboards due to their form factor—they are simply not big enough to house a full keyboard. Modern smartphones alleviate this issue thanks to their bigger size, and the majority include a full QWERTY keyboard. Both of these types of mobile entry methods would benefit from the speed improvements provided by word prediction and auto-completion—however, memory limitations could mean that only the more modern devices would be able to include these systems. Along with text entry using a standard computer keyboard, text input on a mobile phone acts as a secondary motivation for this research.

Assistive technologies and mobile devices provided the two major motivations for this work and provided justification for the majority of the decisions made during the engineering of the system; however, they are not the only hardware interfaces which benefit from more efficient text entry. Many text editors available for various operating systems include a primitive form of word prediction, providing a list of words previously entered into the current document. This can often be used to increase the rate at which typing is possible—however, it does not take advantage of language models or sentence-level prediction. Along with simple

text entry, more specific systems could benefit from assistive input methods, such as code completion for software development in an integrated development environment (IDE).

1.2.1 Brain-computer interfaces

Brain-computer interfaces (BCI) are a special class of assistive technology which allows users to provide input into a system through use of a special electroencephalographic (EEG) headset which is able to read electrical activity within the user's brain [59]. BCI systems work with one of several paradigms, including imagined motor movement (by reading mu- and beta-rhythms) and visually evoked potentials (VEP; such as steady-state evoked potentials (SSVEP) and P300 signals).

A common application of BCI is a *visual speller*: a method of entering text by presenting the user with a visual representation of a keyboard (which may consist of a full set of letters [49, 57] or groups of letters [7]). These visual spellers can be controlled by any of the aforementioned paradigms—and while each has their own advantages and disadvantages, all except SSVEP are limited in the number of available system inputs. When compared with text entry on a mobile device with 12 keys, BCI devices have a considerably smaller input set; often a device will have only 4 or 5 inputs available. For this reason, BCI spellers rely heavily on ambiguous keyboards.

Despite increasing interest in the field of BCI—and the rapidly decreasing cost of consumer devices (with some selling for under \$1000 USD)—visual spellers are still not often considered to be viable alternatives to other assistive technologies. The deciding factor in this is likely to be the speed at which text can be entered using them; for example, one of the best performing visual spellers

is the Hex-o-Spell system, with which users have been measured to obtain up to only 7.6 characters per minute (CPM) [12]. This speed is seen as prohibitively slow in almost all use cases. As a result, brain-computer interfaces are often regarded as a last resort for AAC.

That the existing visual speller systems are so slow provided the original primary motivation for this research. As the time required to input text increases, the effect of performance increases exhibited by assistive input methods such as word prediction become much more noticeable. This motivation was the driver behind several of the decisions made during the construction of the assistive input method, including the addition of an ambiguous keyboard (a requirement for many BCI paradigms) and the removal of the requirement for ambiguous keyboard splits to be ordered. The latter decision is based on the fact that the way the internal keyboard split is represented is not tied to the visual display of this split—the keyboard split must be displayed to the user in a sensible order to improve usability, but this disassociation allows any keyboard split to be used, as long as the split is displayed in order in the user interface. The specific factor that makes this possible depends on the paradigm, but—generally speaking—it is based on the fact that the position of elements are irrelevant to the BCI system (the important factor is, for example, the frequency of visual pulsations for SSVEP spellers).

Communication aids—such as brain-computer interfaces and other assistive technologies—and mobile devices both serve as motivations for this thesis, as it is these two classes of input methods which would benefit most from increased input rates. To this end, this thesis looks to design and implement an assistive input system which will provide increased performance, as well as evaluating the parameters which have the biggest impact on perfor-

mance. The specific goals which will achieve this target are described henceforth.

1.3 Goals

The class of assistive technologies known as assistive input methods incorporates several different text entry systems, including ambiguous keyboards, word prediction systems, auto-completing keyboards, and more. Each of these input methods serve a different purpose, whether it be supporting text entry with an incomplete set of inputs, or increasing the speed with which a user can enter text. Assistive input methods generally have several configurable parameters which can be adjusted to obtain optimum performance. This main goal of this research is to determine whether it is possible to create an all-encompassing input method which allows users to input text at a high rate with a limited set of inputs. An additional, secondary goal of this research is to determine the impact of corpus size on the memory usage and response time of the system. By developing and evaluating this system, this thesis intends to answer the following four questions:

1. Can an input method be devised which allows input using a limited entry method at speeds comparable to (or better than) a standard keyboard?
2. What is the impact of various tuning parameters on the performance of the assistive typing system?
3. How does the size, domain and writing style of the corpus used to train the system affect the performance of the combined input method?

4. How is the response time and memory usage of the system affected by the size of the corpus loaded into the system?

In achieving these goals, this thesis contains three major contributions:

1. A novel assistive input method which demonstrates equal (or better) performance than a standard keyboard;
2. An evaluation framework which can be used to investigate effects of the training corpus and tuning parameters; and
3. An analysis of the effects that the training corpus and tuning parameters have on the performance of the system.

1.3.1 Efficiency targets

As previously mentioned, one of the main goals of the research was to determine whether the performance of an assistive input method could improved such that it reaches speeds comparable to that of a standard keyboard. To formalize this, the assistive input method developed for this research attempts to achieve a lower keystroke per character (KSPC; see Section 5.1 for more details on this measure) count than 1.0, which is the KSPC of a standard keyboard. This is the measure that the experiments described in Chapter 5 and the results presented in Chapter 6 will be compared to.

Although the assistive input system was designed to match the efficiency of a standard keyboard, many of the cases in which this category of input method is used have severely limited hardware available when compared with the contemporary computers on which these experiments were performed. While newer mobile phones do not suffer as much from such limitations due to recent

major advances, even they have limited available memory and storage capacities when compared with the potential size of a corpus used to train the system. Additionally, Section 3.4.1 presents the concept of a minimum response time for the system. Section 3.4.3 investigates the relationship between corpus size and memory usage, as well as the relationship between corpus size and system response time with the system running on a modern computer; the results of this investigation determines the ability for the assistive input method to run on systems of varying power.

1.4 Chapter summary

The first chapter of this thesis introduced the concept of an assistive technology, and gave an overview of the many different types of assistive technologies available for text entry. The three main types of assistive input—the three which form the basis of the system used for this research—were described in detail. Following the introduction of these concepts and systems, the motivation for this research was outlined (including the major motivation in the form of brain-computer interfaces). From this motivation, the goals of this research were detailed—it is these goals which the remainder of this thesis will aim to meet.

1.4.1 Layout of thesis

This chapter introduced the concept of an assistive technology, and discusses the various types of assistive input. The motivation for this research was also introduced, along with a discussion of brain-computer interfaces and their relation to assistive input. The goals that this thesis will attempt to achieve were also presented. Chapter 2 investigates the background of assistive tech-

nologies and reviews several works related to this thesis. Chapter 3 details the design of the system used to investigate the properties of assistive input technologies which are being evaluated in this research, and covers the decisions that went into constructing such a system to meet the aforementioned goals. The chapter also includes the experiments used to measure the impact of corpus size on the response time and memory usage of the system. Following this, Chapter 4 details the corpora which were constructed for use in this research, while Chapter 5 designs a set of experiments which use the corpora to evaluate the system. Chapter 6 presents the results obtained from the experiments which were run to evaluate the system, while providing a discussion on these results. The final chapter, Chapter 7, provides several conclusions for the thesis, and outlines directions for future work in the area.

Chapter 2

Background

For many people with disabilities, even the most menial of everyday tasks can be difficult to perform. Assistive technologies aim to make it easier for such people to perform these tasks, by providing aids to assist them. Such tasks are wide ranging, and can be as simple as hearing clearly or reading easier, to something more complex such as interacting with a computer. Assistive technologies are not only useful to those who find certain functions difficult to perform unaided, but also useful for people in situations where they are constrained by their environment—for example, interacting with a computer when their hands are occupied, or entering text on a small device.

Aids for entering text fall under a special class of assistive technology known as augmentative and adaptive communication (AAC): adaptive technologies which are designed to allow users to communicate where they would be otherwise unable to. The remainder of this chapter will investigate the history of assistive technologies in general, as well as providing an in-depth background into AAC systems. The three AAC technologies incorporated into the design of the assistive input method created for this thesis are ambiguous keyboards, word prediction systems and

word auto-completion; as such, background into these three technologies in particular will be presented in the following sections.

2.1 Assistive input systems

Assistive technologies span a wide range of tools and aids, from hearing aids to screen readers. AAC technologies—a subset of assistive technologies—improve the efficiency with which users can communicate using computerized communication aids. These aids require input from the user which meets their requirements set forth by their disability; this input is necessary for selecting the correct symbols used to generate communication signals (whether this is text or speech). Low-tech systems include physical switches to trigger pre-recorded phrases, head/mouth tubes, gaze trackers, or a simple joystick or trackball system [4, 28].

A modern, high-tech type of assistive input system is a brain-computer interface (BCI). BCI systems were designed as a way of interacting with a computer for people with severe motor disabilities. The systems work by reading electrical signals from the user's brain through a headset, which detects surface-level electroencephalographic (EEG) signals. Signals which are used to control the system include visually evoked potentials (VEP) and imagined motor movement, amongst others. The interfaces have been used for many different purposes, including controlling robotic movement (such as an electric wheelchair [44, 58] or a prosthetic arm [36, 66]), other control tasks [52], and text input via a visual speller [17, 23, 57, 75]. As discussed in Section 1.2.1, brain-computer interfaces were one of the primary motivations for this research, as they suffer from almost intolerably slow input rates—by using assistive text entry techniques such as ambiguous keyboards and word prediction, these interfaces could see a great in-

crease in performance.

2.2 Assistive text entry

Assistive technologies are commonly employed to make communication easier for people with physical disabilities which prevent them from otherwise communicating efficiently. These tools enable AAC, a tool-assisted method of enabling people to produce communication signals who would otherwise struggle to communicate with others due to physical restrictions. Part of the early research into these communication aids involved the development of a standard construction of such a tool [55, 63, 64]. The accepted standard for separating communication aids into individual components was proposed in 1981 [9], and considers an assistive input tool to consist of four components:

- An *input device* to translate user's movement into input;
- *Output devices* to display the selected symbols;
- A *selection algorithm* to choose the correct symbol from the input signals; and
- A *prompting device* to show the user which symbols are currently available for selection.

This thesis presents an assistive input method which is designed to allow for text entry using a limited key set at a rate comparable to that provided by a standard keyboard. While there is no restriction on what type of hardware, a simple implementation is a small keyboard with a small number of keys, such as those found on mobile devices. Using this as an example, the *input device* refers to the keys on such a keyboard, which allow not only

text input, but also the selection of suggestions by the ambiguous keyboard and word prediction systems. The *output devices* refer to the screen, which also doubles as the *prompting device*. The *selection algorithm* refers here to two separate constructs—the disambiguation algorithm used by the ambiguous keyboard, and the sorting algorithm used by the sentence-level word prediction system.

The history of the three major classes of assistive text entry—ambiguous keyboards, word prediction and word auto-completion—is presented in this section. Much of the research into AAC has also looked towards leveraging social contexts in an effort to improve the ability of people (especially children) with disabilities such as autism to communicate [8, 61]; however, this thesis will only consider solutions to the mechanical aspect of this problem.

2.2.1 Ambiguous keyboards

The English language contains upwards of 50% redundancy due to the structure of the the written language; for example, graphemes such as “pq” never occur, while “er” occurs very frequently. Ambiguous keyboards take advantage of this fact by mapping multiple input characters to each input, and algorithmically disambiguating based on language models. One of the earliest motivations for ambiguous keyboards was AAC, as a number of characteristics of ambiguous keyboards provide potential benefits for users of AAC. The three goals of this research that resulted in benefits for AAC users were the reduced number of inputs required to use an ambiguous keyboard, the reduced number of keystrokes required to input text, and the familiarity of ambiguous keyboards to users of AAC (thanks to the prevalence of T9 on mobile phones—although this benefit was not made clear until later) [39]. The first commercially-available implementation of

an ambiguous keyboard was the T9 system [30], which was (and is still being) used in many mobile phones. While AAC is often not considered a motivation for modern research into ambiguous keyboards [39], this was not the case for the T9 system—the material regarding the development of such explicitly listing it as a major motivation during the development of the system [37, 42].

The concept of an ambiguous keyboard was originally proposed in two separate papers [27, 38]. Unlike modern ambiguous keyboard systems, the systems proposed in these studies relied on manual disambiguation: that is, they required the user to manually select which letter was to be entered at each keypress. This was done by requiring the user to perform two selections to enter a single letter: the first to select the key containing the character group, and the second to select the character to enter. While this reduced the reliance on disambiguation algorithms to successfully predict the user’s desired character (which, in turn, avoided the need for language models or training corpora), the performance (in raw keystroke cost) is approximately half that of later “predictive text” style ambiguous keyboards. In the mobile phone domain, these older ambiguous keyboards are more similar to multi-tap input methods—that is, a keyboard which requires the user to tap a key multiple times in quick succession to select the desired character—than predictive text systems such as the modern T9 keyboard [30].

Several years after the advent of the original systems, research began looking at ambiguous keyboards using statistical models to perform character-based disambiguation. Levine et al [47] proposed a technique based on requiring the user to disambiguate each character as entered—this differs from the aforementioned manual disambiguation as the character disambiguations were sorted according to character-level statistics. This technique relies

on a statistical model based on a language model to determine which characters are mostly likely to follow the previous n characters of the current word (where n is a tunable parameter). This system was found to improve on the manual disambiguation keyboards, but the performance was much worse than subsequent automatic word-level ambiguous keyboards. One of the main reasons for focusing on character-based disambiguation (as opposed to the later word-based disambiguation techniques) was due to limited memory of the devices for which ambiguous keyboards were being developed. However, as hardware improved, these limitations were relaxed, and research began focusing on word-level disambiguation [29, 70]. While word-based disambiguation allows for much higher potential input rates (nearly one keystroke per character [1] compared with 1.15 keystrokes per character [50] for character-level disambiguation), it is reliant on a suitable internal dictionary model (see the following section for details).

Optimizing disambiguation performance

One of the major issues with fully-automatic ambiguous keyboards is the reliance of the system on the language model or internal dictionary. This reliance is such that it is generally impossible to enter words with the system if they do not appear in this source. There have been attempts to disambiguate these words through the use of statistical methods [45], but this has generally been unsuccessful. The focus of the majority of research has been instead on constructing a corpus—whether manually or automatically—which allows the user to type any words that they desire [70, 71].

In Section 1.1.1, it was stated that the performance of an ambiguous keyboard is highly dependent on the way in which the keyboard is split—that is, the way the characters are arranged on the ambiguous inputs. Existing research has attempted to dis-

cover a universally-optimal ambiguous keyboard character arrangement using a number of methods such as keystroke evaluation [45] and particle swarm optimization [80]. However, the target system for previous research has been almost exclusively mobile phones—namely, mobile phones with 9 keys. As such, the optimal character arrangements discussed in these papers are for 9 ambiguous classes, making them incompatible with the system developed for this thesis. The benefits of optimizing the keyboard layout is high, however, with research suggesting a text entry rate approaching one keystroke per letter for reduced keyboards [1]; the implication of this being that the performance of an ambiguous keyboard can approach the performance of a full keyboard given an optimal character arrangement. The impact of this on a severely-reduced keyboard will be investigated as part of the research performed for this thesis—the experiments which corroborate these findings are outlined in Section 5.2.4.

Along with the character arrangement, the performance of an ambiguous keyboard can be further improved by ordering the results of the disambiguation process. A common (and somewhat naïve) approach to this order is simply ordering the matches by frequency of their occurrence in the English language [20], which can be determined by constructing a language model from large amounts of text. Further research has attempted to leverage sentence-level context information when sorting disambiguations [31, 35], which presents users with ambiguous keyboard matches in an order which reduces the number of keystrokes required to select the correct disambiguation; this research has shown improvement over the word-frequency approach.

Optimizing the performance of the ambiguous keyboard was not a goal of this thesis in itself, but a side-effect of investigating the effect that different parameters had on the performance of the

input method. For this reason, the techniques for finding an optimal character arrangement were deemed to fall outside the scope of this thesis, and a simple method of discovering a strongly-performing (but not necessarily optimal) arrangement was used in its stead. This process (described in Section 5.2.4) was used to approximate the techniques described in previous research. Similarly, the implementation of the disambiguation algorithm of the ambiguous keyboard does not use the state-of-the-art techniques for ordering matches, but instead makes use of the sentence-level word prediction engine to improve suggestions (the impact of which is measured by experiments in which the word prediction subsystem is disabled). The implementation details of the ambiguous keyboard used in this thesis are described further in Chapter 3.

2.2.2 Word prediction

One of the original purposes of word prediction systems was as a communication aid for disabled users [16, 78]. Word prediction systems allow users to input text at much higher rates than usual by predicting which characters or words they wish to write next, which the user can then accept (generally with a single keystroke). The technique relies on storing an internal word model which acts as a lookup table; whether these lookup tables are character-based or word-based has been the subject of much research. Other research has been undertaken to investigate the performance benefits of using part-of-speech tagging to improve performance, and restricting the size of the internal word model. Other approaches in existing research have included using machine learning techniques to offer word suggestions [6].

Word prediction systems contain either a character or word lookup table in order to suggest words based on previously en-

tered text [25, 32, 72]. Early research determined that English is approximately 75% redundant, and only a few characters are required for accurate prediction [67]. From a statistical standpoint, the ability to predict the following words is not significantly improved beyond eight to ten characters [14, 68]. This character-based language model provides the basis for some word prediction systems (e.g. [74]). However, these character-based systems do not contain any information about the context of the current sentence, which would allow for more domain-specific efficiency increases. Using a word-based lookup allows the word prediction system to leverage such domain-specific information in order to suggest more appropriate words, and has been attempted in prior research to some success [70]. However, word-based word prediction systems require much more memory (compare a set of constructs containing two or three words versus a set of constructs eight to ten characters long); regardless, this trade-off is expected to be beneficial, and a word-based approach is used in this thesis.

Part-of-speech tagging is a technique which refers to—as the name suggests—tagging words in a sentence with their part-of-speech (function) within that sentence. This has been suggested for use in word prediction systems as a way of ensuring that only words which are grammatically correct in the current context are suggested by the system [51]. However, other research has determined that the benefit provided by part-of-speech tagging is negligible [74]. One possible reason for this is the fact that part-of-speech tagging is inherently captured by word-based language models.

A large consideration with word-based word prediction systems is the n -gram size—that is, the number of previous words which are used when looking up word suggestions. It has been suggested that the optimal size of n -grams is 2, due to the fact

that the number of possible n-grams increases exponentially with every additional word considered [46], which increases the number of potential suggestions. Additionally, the memory requirements increase linearly with the number of phrases in the language model, which restricts the number of devices on which such a system can be used. This thesis, however, attempts to use a much larger n-gram size (up to 5) in an attempt to quantify the effects that the n-gram size (herein referred to as the *phrase window size*), as high memory devices are becoming much more commonplace.

Another method that previous research has considered for reducing the memory footprint of a word prediction system is reducing the number of words stored in the internal word model. One way of achieving this is by only storing phrases which meet certain criteria (such as frequency of occurrence) [56]. This approach was investigated as part of this thesis in the form of “bad” words and phrases. These words and phrases were not added to the internal language model in an attempt to both reduce memory usage and increase the relevance of the contents of the model.

2.2.3 Text auto-completion

Like word prediction, auto-completion was originally investigated as an AAC technique used to increase text entry rates. Early systems were based on a simple wordlist approach which auto-completed high frequency words when entered by the user [69], and were found to be beneficial for users of AAC technologies [40]. Commonly, auto-completion techniques are coupled with either ambiguous keyboards or word prediction systems, as the same wordlist models used to power these systems also provide the basis for auto-completion (e.g. [19, 20, 56]).

After initial studies found word auto-completion to improve typing speeds, research was performed to look into the effect of various additions, such as leveraging probabilities of the appearance of successive words [3], grammatical properties of words [79], and part-of-speech tagging [11]. Various additional studies have looked into combining these additions [24, 51]. Similar to part-of-speech tagging for word prediction (described in Section 2.2.2), the technique was also not found to give large performance increases when coupled with statistical methods, due to capturing information which had already been reflected in the model [24].

Research into text auto-completion has attempted to uncover the average cognitive cost of auto-completing text. This cost may be high for the user in cases where the auto-completion system is overzealous, auto-completing words which the user is not attempting to type [2]. This is especially true in the case of an auto-completion system paired with an ambiguous keyboard, as the auto-complete system may use a completely different disambiguation, thereby confusing the user (for example, using a mobile phone keypad, entering the key sequence 349 may auto-complete to “EGYPT”, despite the user attempting to type “FIXED”) [20].

At which point the auto-completion system completes the current word has a major impact on the performance of the system. The latest point at which an auto-complete system will complete a word is at a word’s *unicity point*—the point at which one and only one word can possibly be completed from the currently entered key sequence. However, this point generally only occurs after the majority of the characters have been already entered, and the system therefore does not benefit the user greatly. By looking not only at the currently entered letters, but also characters entered as part of the previous words (i.e. leveraging a simple form of sentence-level word prediction), the point at which the word can

be auto-completed can be brought forward without greatly impacting cognitive load [74]. However, if the unicity point occurs too early, an auto-complete system can cause text input rates to decrease, in some cases over 50% when compared to standard text entry [20]. Because the system developed for this research was targeted towards input systems with extremely low input rates, the additional cognitive cost was considered to be negligible; therefore, the assistive input method described in this thesis will always attempt to auto-complete at the earliest possible point.

2.3 Chapter summary

This chapter introduced the history of assistive input systems and—more specifically—tools for augmentative and adaptive communication (AAC). The literature surrounding the three major types of AAC tools—ambiguous keyboards, word prediction and word auto-completion—was then discussed in the context of this thesis. Chapter 3 follows, and discusses the high-level design and lower-level implementation of the system. The following chapters will evaluate the system in the context of a number of hypotheses, all of which will aim to meet the goals set out in Section 1.3.

Chapter 3

Ambiguous Input Method

The assistive input system is a typing aid which is designed to improve a user's ability to input text in situations where the use of a standard keyboard is impossible or infeasible. The system is made up of three major subsystems: an *ambiguous keyboard*, a *sentence-level word prediction* engine and a *word auto-completion* engine. These three features combine to improve the input rates of alternative input methods (such as augmentative and adaptive communication aids, or mobile devices) which may have a limited input set, be mentally taxing to use, or are incredibly slow to type with. Additionally, the implementation of the system includes a component for constructing and storing the wordlist and phraselist data structures.

The version of the assistive input system created for this research does not include an interactive element—that is, a system to allow a user to enter text—but merely an automated system for evaluation. However, the system works by emulating an interactive mode. This was done in order to evaluate the system in a meaningful way; that is, evaluating the system will give results which reflect the true nature in which a user would interact with such an input method. Additionally, because the automated

testing is performed in a separate layer to the underlying subsystems layer, an interactive version of the system can be created by replacing the automated testing façade with an interactive one.

This chapter will detail the design of the assistive input system, and discuss some of the considerations that went into the design process. After a high-level overview of the design, the system algorithm is described in detail. Following this is a section devoted to the different configuration flags and options found within the system. The following section discusses the limitations and issues that were considered during the construction of the system; this section also includes a pair of experiments which measure the effects of these limitations. Finally, the chapter concludes with a summary.

3.1 Overview of design

The assistive input system consists of three distinct subsystems. The first subsystem is the ambiguous keyboard, a component which allows entry of a set of characters larger in size than the set of characters which are able to be entered—that is, it allows a full set of characters to be entered with a smaller set of inputs available. The second component is the sentence-level word prediction subsystem, which takes advantage of collocations to allow users to enter phrases at a faster rate by suggesting words before the user has finished (and in some cases, started) typing them. The third and final subsystem increases the rate of typing by automatically completing the user's word based on the two internal models—the wordlist and the phraselist—preventing the need to type the words in full. These three subsystems, combined with their various configuration options, make up the assistive input system.

Figure 3.1: Design of the assistive input system

3.1.1 Ambiguous keyboard

The ambiguous keyboard subsystem allows users to enter text into the assistive input system with fewer than the number of inputs usually required to enter text (that is, one per letter). To do so, each letter in the target language (in this case, English) is mapped to an input key. Because there are fewer input keys than letters to be typed, the input keys will have more than one letter associated with this. This one-to-many association introduces a source of ambiguity when entering text. The ambiguous keyboard subsystem is required to disambiguate this text entry by attempting to predict which of the possible letters the user desired when pressing the input key.

In the automated testing framework created for this research, a standard English letter is input into the system. This letter is then

converted to an alphanumeric key class representing an ambiguous input key. As a word is entered, the assistive input system's internal wordlist is queried to determine which words match the ambiguous input keys. The wordlist must then be presented to the user in some way to allow the user to select a different match if required. The order that this list is presented can be varied, and it is the order that defines the performance of the subsystem (as the higher in the list the user's desired word is located, the fewer the keystrokes required to select it).

3.1.2 Sentence-level word prediction

Sentence-level word prediction provides the user with a way to quickly select words which are likely to follow the words previously entered into the sentence. This component is based around a feature of the English language known as collocations. These are structures made up of several words which are commonly used together in English text and speech (and are described in further detail in Section 1.1.2). This component works by querying the internal model's phraselist for phrases which are prefixed by the user's previously entered words.

For any given prefix, there are likely to be many potential words which begin with the same set of characters. Because of the high number of prefix matches, it is required for the system to present these in a sensible order to maximize efficiency for the user (as the further down the list the word appears, the less likely the prediction is for the user). This sorting is done for similar reasons as sorting the ambiguous keyboard matches: there is little efficiency to be gained by requiring the user to scroll a large list in order to save two or three keystrokes. In the case of the implementation of the assistive input system, words outside the top five suggestions

are not presented to the user, making it even more important to rank the suggestions appropriately (five suggestions was chosen as the default value for this thesis as the system uses five inputs—four character classes plus the space key—with which to select a suggestion). While it is possible to rank the suggested words by frequency, this is a fairly naïve approach.

3.1.3 Word auto-completion

Word auto-completion allows a user to skip typing their current word and continuing on to the next, allowing the system to insert the remainder of their current word on their behalf. The implementation of word auto-completion in this assistive typing system involves automatically inserting the top suggestion from the sentence-level word prediction engine. The inserted auto-completed word can be accepted by the user by inserting either a space or a punctuation mark, or can be rejected by simply ignoring the word and continuing to type. In the case where a word has been auto-completed but the user desired a word pertaining to the input key sequence (for example, the word “type” was auto-completed to “typed”), the user can clear the auto-completed suggestion with a single press of the backspace key (with further presses of the backspace key deleting characters as per usual). The word auto-completion subsystem is closely linked to the sentence-level word prediction by virtue of the auto-completion being taken as the first entry in the word prediction suggestions.

3.2 Algorithm

Before it is able to be used, the system must first be initialized through the construction of both a word- and a phraselist. These

data structures are constructed from two corpus files: the first file is a list of words and frequencies, and the second is a list of phrases. The format of these files is described in further detail in Section 4.1.

Once this initialization phase is complete, the system is able to accept input in the form of characters and meta keys (for speller control, such as selecting from a list). As a user types, the system tracks three key pieces of information: one from each of the three main subsystems. The first is the word (or letters) the system thinks the user is typing according to the ambiguous keyboard mapping—a user is able to cycle through a list of alternatives for the current word at any point. For an interactive input method, this information would be displayed under the user’s cursor. The second is a list of suggestions provided by the word prediction engine to complete the current word. From this list, the user is able to select a completion of the current word with a single keystroke. Finally, the system presents the auto-completed word as a suffix of the currently entered sequence of characters, allowing the user to either accept the auto-completion (by entering a space) or reject it (by continuing to type the word or, in the case where no additional letters are to be typed, by pressing backspace).

3.2.1 Wordlist construction

The first stage of the initialization of the assistive typing system is to load in the wordlist of the training corpus, and generate the word frequencies data structure. As mentioned in Section 3.1.1, the wordlist is stored as a trie, with each node uniquely labelled with the key class of the corresponding letter. Each node in the trie has up to n children (where n is the number of classes in the ambiguous keyboard; each child represents a single class) and a set

of WORD objects (which are simply {word, frequency} pairs with a natural sort order) associated with the key sequence from the root node to it. A word is inserted into the data structure by first generating the ambiguous key sequence for the word, traversing the trie (adding nodes as necessary) for the key sequence, and inserting the word and its frequency at the resulting node.

Once the entire wordlist has been constructed, the wordlist is then sorted—this ensures that sorting is not required whenever the wordlist is accessed, which is an expensive operation. Sorting the wordlist involves only sorting each individual node; it is not required to sort the wordlist proper. This is due to the fact that a trie is sorted by definition—additionally, nodes in the wordlist are queried in isolation; it is never required to combine the words from multiple nodes. The individual nodes are sorted by ordering the list of WORD objects at that node by descending frequency.

Figure 3.2: A simple example of a wordlist

As an example of adding a word to the trie, we will consider the word “too”. We first need to generate the key sequence for the word—depending on the key mapping, an example of this sequence may be *caa*. We then traverse the trie by starting at the

root node, and traversing down the trie, using each of the key classes in the key sequence as child nodes (that is, we first traverse to the *c* node, then its child node *a*, followed by its child *a* node). Assuming we have previously already inserted the word “to”, the *ca* path will already exist; however, the *a* node may not exist as a child, so this node will be created. Once we have a reference to the appropriate node, we can insert the word into the that node’s list of WORD objects—for example, the pairing may be {too, 54}.

3.2.2 Phraselist construction

At this point of the initialization, we have constructed a data structure combining the ambiguous word mappings and the word frequency table. This enables simple word frequency-based prediction. In order to achieve sentence-level word prediction, we require a data structure which allows us to determine which words are likely to follow a given phrase (this is another frequency-based measure). A simple tree structure is used for this purpose. Each node of the phraselist is labelled with a word and contains a frequency count (which represents how often the word associated with that node appears after the phrase constructed by following the path to the node).

After constructing this phraselist, the entire tree is recursively sorted; this is done by traversing the phraselist tree, sorting the children of each node. The children are sorted in descending order of frequency. The result of this sort is that the children of each node can be used as sentence-level word predictions in the order that they are retrieved. Like the wordlist construction, sorting in the construction phase prevents the need to sort (at high cost) at each query. This is possible due to the fact that—as per Section 3.2.3—the sentence-level word suggestions are presented in order

of the size of the current context first, and frequency of the word occurring in that context second. In other words, we are not required to combine the children of multiple nodes; they are simply presented in descending order of depth.

It is during the construction of the phraselist that the first of the system's configuration options is used. When constructing the phraselist, phrases from the corpus are split into sub-phrases; the phrase window size configuration option determines the length of the sub-phrases which are extracted from the full sentences and stored in the phraselist. The phrases are split into sub-phrases due to the fact that sentence context is often very localized, and will generally not exceed a range of a few words [56]. The phrase window size therefore determines the context size of the extracted phrases. A moving window is employed to extract the sub-phrases (if the variable phrase length flag is disabled, then the phrase is simply split instead), and each sub-phrase is added to the phraselist, incrementing the frequency of each node as the tree is traversed. For example, if the sub-phrase "what will you do" is added to the phraselist, the frequency of the *will* child of the *what* node, the *you* child of the *will* node, and finally the *do* child of the *you* node will all be incremented by one (note that the frequency of *what* node is therefore simply the frequency as taken from the wordlist).

Consider the tree in Figure 3.3, which demonstrates how the phraselist is structured as a tree of {word, frequency} pairings (note that variable window size has been disabled and each phrase from which sub-phrases were extracted was 5 or fewer words long—this was done to simplify the example). The frequency count at each node determines how often that word appears at the end of a phrase beginning with the nodes in the path from the root to the parent. For example, the word "you" appears 8 times

after the phrase “what will”, and the word “is” occurred 4 times after the phrase “hello my name”. Naturally, no node should have a higher frequency value than any node in the path to it from the root—for example, the phrase “what will you” cannot occur fewer times than the phrase “what will you say”.

Figure 3.3: A simple example of a phraselist

A special consideration during phraselist construction is the behaviour of adding phrases which contain “bad” words. Bad words are considered here to be either words which appear with very low frequency or words which are exceedingly long (both of these cases are able to be toggled using the configuration flags detailed in Section 3.3.1). There are four defined special behaviours, of which one can be applied for the phraselist construction process. The behaviours exist for practical reasons: because some of the corpora can be very large, the phraselist can be too large to be used efficiently, and so the phraselist should be trimmed. Phrases which contain bad words—whether they are because of low frequency (which, by association, makes the phrases also low frequency) or unusual length (which are potentially not real words—for example, sentences which are missing spaces between words) are the prime candidates for pruning the phraselist.

Further pruning of the phraselist is possible to improve both

the memory usage of the data structure and the usefulness of the sentence-level word prediction (as a poor suggestion is often worse than making no suggestion at all due to added cognitive load [56]). One suggestion made by Nandi et al [56] is the use of a phrase frequency threshold. Along with the four bad word behaviours, the phrase frequency threshold option is detailed in Section 3.3.2.

3.2.3 System behaviour

After the initial construction of the word- and phraselist, the user is free to begin typing. At any time while using the system, the user is able to perform one of five actions. The first is to enter a character, which will either continue a previous word or begin a new one. The second action—entering a space or punctuation character—has the effect of both inserting the character and, where applicable, accepting the auto-completed word. Similarly, pressing the backspace key will either clear the currently suggested auto-completion or delete the previously entered character or punctuation mark. The other two actions involve selecting suggestions made by the speller: the user can scroll through the list of words matching the currently entered ambiguous key sequence, or select one of the predicted word suggestions.

Character entered

Entering a character into the system is the most common—and the most computationally complicated—action that the user undertakes while typing with an assistive typing system. Entering a character invokes all three subsystems: in order of invocation, these are the ambiguous keyboard, the sentence-level word prediction and the word auto-completion systems. Immediately af-

ter entering a character (where a character is one of the ambiguous key classes), the speller first queries the wordlist tree to determine which words map to the entered character sequence. Because the wordlist is stored as a trie with pre-sorted nodes, the lookup cost for retrieving the list of words matching the input sequence is constant.

After retrieving the list of matches for the ambiguous key sequence—but before it displays the topmost word from the list as a match—the system retrieves a list of sentence-level word completion suggestions for the currently entered phrase. It does this in order to sort the words from the ambiguous keyboard by the context size determined by the sentence-level word prediction to present the list in a more efficient order for the user. To do this, the system visits the node corresponding to the end of the currently entered phrase and retrieves all words prefixed with the currently entered ambiguous key sequence. If the phrase window size is greater than 1 (that is, in most cases), there may be more than one node from which to retrieve word predictions. For example, if the user entered the phrase “what is” with the phrase window size set to 2, prefix matches for the currently entered key sequence would be extracted from the nodes at $(root) \rightarrow \textit{what} \rightarrow \textit{is}$ and $(root) \rightarrow \textit{is}$. In these cases, the nodes are visited deepest to shallowest (i.e. longest to shortest phrase match), which results in a word suggestion list which is sorted first by how many of the previous words in the phrase have been known to precede the match, and secondly by raw frequency.

At this point, both the wordlist and the phraselist have been queried for the current phrase and ambiguous key sequence. The wordlist can now be sorted according to the phraselist prefix matches (as above) and display the top match as the currently entered word. The sentence-level word prediction list is also populated

here. The word prediction list is populated with, starting from the second entry, the number of word prediction suggestions as determined by the sentence-level word prediction suggestion list size from the phraselist query results. This word prediction list can now be displayed to the user. At this point, the word auto-completion system (if enabled) will auto-complete the current word with the top suggestion from the word prediction suggestion list.

Space/punctuation entered

When either a space or a punctuation mark is entered, the currently entered word is considered complete. If the current word has been auto-completed, then this auto-completion is accepted. If a sentence-ending punctuation mark is entered (for example, a fullstop or an exclamation mark), the current phrase is terminated; this means that any further words which are entered (in the following sentence) will not cross back over this phrase termination boundary when using a phrase for sentence-level word prediction. In other words, at most only words since the last punctuation mark will be used as a phrase prefix for the sentence-level word prediction.

On entering a space during the normal operation of the assistive input method, the sentence-level word prediction and word auto-completion processes take place as they would when a letter is entered, using a null value as a prefix for the current word. That is, the words suggested to the user are the most commonly entered words following the phrase prefix (the at most n previous words entered since the previous punctuation mark). This allows the user to be presented immediately with a suggestion for the next word upon finishing their current word and, furthermore, auto-complete it for them. This allows the user to potentially—in the best case scenario—complete entire phrases by continually

accepting the suggested words as they are auto-completed.

Backspace pressed

The backspace key performs two functions in the assistive typing system: if the word which is currently being entered has an auto-completion pending (i.e. the auto-completion is being displayed to the user but has not yet been accepted), the backspace key will simply clear this auto-completion and does not delete characters. On the other hand, if there is no auto-completion pending, the backspace key will simply delete the previously entered character. The dual nature of the backspace key allows users to reject the word auto-completion when the word has been fully entered, yet the auto-complete functionality has suggested a longer word. For example, if the user wishes to type the word “check”, the system may auto-complete to “checked” after the user has finished typing the word in full. In these cases, inserting a space will accept the auto-completion, and therefore inserting a backspace character is required to first clear the auto-completion. Note that this is one of the primary cases where typing the word with auto-completion enabled will be slower than typing with the system disabled. Also note that due to the nature of the ambiguous keyboard, the auto-completed word will not necessarily have the same matched prefix: the word “check” may in fact auto-complete to an entirely different word.

Scrolling matches

As suggested by the name, the ambiguous keyboard will more often than not match more than one word for any given key sequence input. For this reason, the user requires a means to scroll through the matches. Because the wordlist is sorted following its

construction, entering a letter simply needs to obtain the list of words which matches the key sequence. This means that in order to select an alternative match of the ambiguous key sequence, we only need to select a word from the already-obtained list. This prevents the need for a costly tree lookup and sort for each ambiguous matches scroll operation.

In interactive mode, scrolling through the matches will generally be mapped to a pair of inputs, such as up and down arrow keys; hence, when evaluating the system, the keystroke cost of selecting a word from this list is equal to the position of the word in the list minus one (that is, a cost of one to select the second word in the list, two to select the third, and so on). An important consideration here relates to the auto-completion system: if the currently entered key sequence has an auto-completion pending, the user will not know what the top match from the ambiguous keyboard is (given that it will be masked by the auto-completion). While this does not affect the automatic evaluation, this would need to be alleviated for a real user using the system; in this case, if there is an auto-completion pending, the first scroll of the ambiguous matches will instead simply clear the pending auto-completion (and therefore adding a cost of 1, mirroring the functionality of the backspace) and show the top ambiguous keyboard match. An additional consideration when evaluating the system is the cost of entering a word which does not exist in the wordlist of the input method—the cost of this is determined to be the length of the list of word matches, plus the length of the word being entered, plus one. This cost represents the user scrolling the list of matches in an attempt to find the desired word, pressing a key to enter a free type mode, and typing the word manually using a standard keyboard. Note that this cost includes cases where there are no matches for a given key sequence—in this instance, the cost of

scrolling the list of matches is 0 (as the list is empty).

Selecting suggestion

Selecting a sentence-level word prediction requires two keypresses: one to enter a suggestion selection mode, and a second to select the desired word suggestion. This is to simulate the limitation of having a small number of input keys, which is typical of the average input device suited to assistive typing system; for example, BCI spellers will generally not support enough inputs to allow unique selection from a list in the standard typing mode, and instead require a separate keypress to enter the selection mode. Like scrolling through ambiguous matches, the word prediction suggestions have already been obtained at this point, and selecting a suggestion for input will simply pull the correct index out of the list. When the suggestion has been selected, the cursor is then placed after the inserted word, allowing the user to either continue typing the word or finish the word with a space or punctuation mark. However, note that auto-completion is disabled for the selected word; this prevents situations where a suggestion has been inserted, but a press of the backspace is required to clear the auto-completion before inserting a space or punctuation mark.

3.3 Configuration

The system has been designed in such a way that every feature is able to be switched on or off, in order to determine the effect that each may have on the overall performance. On top of this, many of the features have values which can also be configured. For example, if we wish to determine the effects of the ambiguous keyboard on its own, we can simply disable the two remaining

subsystems—the sentence-level word prediction and word auto-completion. Additionally, we can also measure the impact of the phrase length, for example, by comparing different values of the appropriate configuration option. This section lists each of the configuration flags and options which allow us to investigate different aspects of the system.

3.3.1 Flags

The configuration flags available in the system serve to enable or disable various features or options. By explicitly choosing which subsystems and features are enabled and which are disabled, it is possible to evaluate the effects of each in isolation—imperative in determining the optimal configuration for the system. The flags available cover aspects of the system ranging from enabling and disabling the three major subsystems to whether to give special treatment to certain words and phrases. This section details each of the flags available in the system.

Enable/disable the ambiguous keyboard

This flag allows the ambiguous keyboard to be disabled. The ambiguous keyboard is the one subsystem which is **not** designed to increase raw typing speed, but instead to allow text entry with a limited set of inputs. By disabling this, we can determine the raw keystroke savings provided by the two subsystems which are designed to improve efficiency. Disabling this results in the system assigning each input letter with its own unique class (rather than assigning multiple letters with the same class).

Enable/disable sentence-level word prediction

This flag enables or disables the sentence-level word prediction engine; disabling this flag allows for the evaluation of the two remaining subsystems. Note, however, that the word auto-completion subsystem relies on this subsystem being active to work; by disabling the word prediction subsystem, the word auto-completion system will never save keystrokes (in fact, in some cases it will require more keystrokes than a standard keyboard).

Enable/disable word auto-completion

Similar to the previous two flags, this flag disables the word auto-completion system, allowing the other two subsystems to be tested in isolation. As mentioned prior, the auto-completion subsystem relies heavily on the word prediction subsystem—although these flags are not linked, disabling the word prediction component should always be paired with disabling the word auto-completion.

Enable/disable variable phrase window size

The variable phrase window size flag determines whether or not phrases of varying size should be inserted into the phraselist. When reading the phraselist into the system, this flag determines whether sub-phrases with a length between 2 and $p - 1$ (where p is the phrase window size) will be extracted along with sub-phrases of size p ; if the flag is not enabled, only sub-phrases of size p will be extracted. For example, if the flag is enabled, the sub-phrases “what are” and “what are you” will be extracted from the phrase “what are you doing” (along with the full phrase). If the flag is not enabled, only the full phrase will be extracted (assuming a phrase window size p of 4). Note that this only affects sub-phrases lo-

cated at the beginning and the end of each phrase in the corpus file, as a sub-phrase s which has a length less than p will be added to the phraselist during extraction of the phrase of length p which begins with the sub-phrase s . As an example of this, the sub-phrase “what will” (s) is added to the phraselist when the phrase “what will you do” (p ; $s \in p$, $|s| < |p|$) is extracted, and does not need to be added separately.

Note that the flag does not affect the retrieval of phrases from the phraselist in the sentence-level word prediction subsystem: the size of the phrase prefix is always variable to ensure that there will be an appropriate number of suggestions to show the user to reach a minimum level of usefulness.

Enable/disable phrase frequency threshold

The phrase frequency threshold refers to the minimum number of times a phrase must appear in the corpus in order for it to be kept in the phraselist—if this flag is enabled and a sentence does not appear with the minimum frequency τ , the phrase will be removed from the phraselist. The justification for this is that low-frequency phrases are likely to be specific to the context from which they were extracted, and therefore should not be included in a generic corpus phraselist. The minimum occurrence threshold (which is a tunable parameter) ensures that the corpus only includes phrases which are useful in multiple contexts.

Phrases which do not meet the minimum threshold requirements are removed as part of phraselist pruning, which occurs after its construction; disabling this flag essentially disables this pruning.

Low frequency words are bad words

The concept of a bad word was introduced in Section 3.2.2 as a word which is not to be added to the wordlist. This flag determines whether or not a word should be labelled as a bad word depending on its frequency of occurrence. By removing words which appear with a frequency below the threshold (thereby removing words which are highly unlikely to be entered by a user), the wordlist can reside in less memory, and accessing the wordlist can be done more efficiently. Additionally, removing low frequency words reduces the chance that the ambiguous keyboard matches will contain words which the user does not desire.

Very long words are bad words

The decision to treat very long words as bad words is a purely pragmatic one: due to system limitations, very long words can cause the system to run out of memory, or even break entirely. Because there is a maximum reasonable length of a word in English, it is possible to simply prune any words which exceed this length. By lowering this threshold, it is possible to prune words which may exist in English, but are used very infrequently (especially in everyday language)—by removing these words, it is possible for the performance to be increased, as these words would only serve to clutter the word prediction suggestions and auto-completion.

3.3.2 Configuration options

The following set of configuration options allow various aspects of the system to be tuned in order to investigate the impact each has on performance. In addition to this investigation, by finding the optimal combination of parameter values and settings the performance of the assistive input system can be maximized, thereby

meeting the other major goal of this research.

Key mappings for the ambiguous keyboard

The ambiguous keyboard subsystem maps the entire set of English letters to a smaller set of input keys. This mapping—including the number of input keys—can be changed using this option. By varying the mapping, it is possible to gain further performance from the ambiguous keyboard, by reducing the ambiguity of each input key sequence. Various key mappings were investigated as part of the experiments outlined in Section 5.2.4. Note that the keyboard mapping is normally required to be ordered in some way to reduce cognitive load—this effect is further detailed in Section 1.2.1.

Phrase window size

The phrase window size refers to the length of phrases which are extracted from the training corpus; in the case of the variable phrase window flag being enabled (described in Section 3.3.1), this option refers to the maximum phrase window size. As discussed in Section 3.2.2, the phrase window size correlates with the size of the context being extracted during phraselist construction. By varying this option, we can extend or reduce the context size, which will impact the performance of the sentence-level word prediction subsystem. This option also affects the memory usage of the system; by tuning this parameter, the size of the phraselist (and therefore the memory cost) changes exponentially.

Phrase frequency threshold

Section 3.3.1 describes the phrase frequency threshold as removing infrequently occurring phrases in order to keep the phrase-

list from containing phrases which are only useful in specific contexts. This parameter simply sets the threshold under which these phrases are removed. By increasing this threshold, the number of phrases pruned from the phraselist is increased, resulting in a phraselist which is more useful in more contexts.

Sentence-level word prediction suggestion list size

The sentence-level word prediction system constructs and presents to the user a list of suggested words to follow the already-entered words in the current sentence. Scanning this list introduces a cognitive load which is a direct function of the number of words in the list. By reducing this number, the cognitive load can also be reduced. This option determines the size of the list which is presented to the user as part of the word prediction subsystem.

Minimum word frequency threshold

The minimum word frequency threshold is described in Section 3.3.1 as a minimum number of times a word must appear for it to be included in the system's wordlist. Uncommon words are purged from the wordlist for previously-stated efficiency and performance reasons; the minimum frequency that these words must appear is determined by this parameter.

Maximum word length threshold

Similar to the minimum word frequency, the maximum word length threshold defines the maximum word length of "good" words. The reasons for this setting are described in Section 3.3.1, which can also be summarized as being for efficiency and performance reasons.

Bad words behaviour in phraselist construction

The bad words flags and options determine which words are to be pruned from the wordlist—however, these words will still exist in the phraselist (as the wordlist was originally constructed from the same corpus). Because the words no longer exist in the wordlist, it is impossible to type these phrases; for this reason, these bad words should also be removed from any phrases which may contain them. How these phrases are altered may affect the performance of the system, however, so four different behaviours for dealing with phrases containing bad words were created. When adding a phrase containing a bad word to the phraselist, one of the following four behaviours is exhibited:

- **Drop children:** insert phrase up to and including the bad word.
- **Drop self:** insert phrase up to (but don't include) the bad word.
- **Drop start:** drop the entire phrase if it starts with bad word.
- **Drop all:** drop the entire phrase if it includes the bad word at all.

3.4 Considerations, limitations & issues

When engineering the assistive input system for this research, two issues needed to be considered regarding the low-level performance of the system. The first of these issues was ensuring that the system responded to each keypress within an acceptable amount of time (that is, the minimum response time of the system). The second issue was that of the memory limitations: due to the nature

of the system, a large amount of memory is required to load in the larger corpora—an amount which at times exceeds the available memory. This section discusses these issues and possible solutions, as well as an experiment to determine the effect of corpus size on both response time and memory consumption.

3.4.1 Response time

When using an assistive input method such as this, the system must be able to respond quickly to user input. If the system fails to respond to a keystroke within 100ms, cognitive dissonance may occur [10, 53], potentially causing disorientation for the user. A modern machine is able to handle this requirement with relative ease for a small corpus, but a per-character delay of 100ms is impractical for automatic evaluation, due to the large amounts of text which would otherwise be able to be evaluated in a very short amount of time. For this reason, the system must be designed in such a way to absolutely minimize this delay. The two main considerations for minimizing the impact of a slow response time during corpus evaluation are a multi-threaded evaluation process and minimizing unnecessary word- and phraselist sorting.

While it is not feasible to parallelize word- or phraselist queries (and therefore the interactive speller), an automatic evaluation system is ripe for multi-threading. As the evaluation process involves evaluating a very large collection of independent phrases, the evaluation of each of these phrases can be done in parallel. On a system which can execute up to 8 threads in parallel, this can result in the speed of the system increasing substantially. This point was carefully considered during the implementation of the system, as it required designing the system in a stateless, thread-safe manner.

In an earlier version of the assistive typing system, the most costly operation was found to be the sorting of both word- and phraselist sub-lists. In order to alleviate this bottleneck, the final implementation sorts both the entire wordlist and the entire phraselist during the construction phase. In doing so, the need for frequent, expensive sort operations is reduced. The removal of the repeated sorts was possible due to the decision to order word suggestions by phrase depth first and frequency second (as opposed to just frequency)—a decision which effectively makes each leaf in the phraselist tree completely independent.

3.4.2 Memory limitations

The assistive typing system used in this research is unique amongst similar input methods due to its use of large data structures. These large data structures are a result of using large corpora to investigate the performance benefit this may or may not provide. The size of the data structures is determined by a function of the corpus size and the phrase window size, with the function varying depending on whether or not the variable phrase length flag is enabled. Constructing the phraselist requires an entire corpus to be loaded into memory, and it is this requirement that causes the corpus size to be limited by memory availability. Enabling the variable phrase length flag results in considerably higher memory usage due to the duplication of phrases (as in this case, the phrase length essentially represents the amount of phrasal overlap). For these reasons, it is often impossible to build an entire phraselist tree in memory.

Large corpora

While the assistive input method was able to load in reasonably large corpora (up to approximately 2 million phrases or 50 megabytes, although this can be further increased by reducing the phrase length and disabling variable phrase length), it is impossible in the current design to evaluate huge corpora. In a later chapter (Chapter 6), we discuss the impact of corpora of various sizes on the ability of the assistive input method to improve typing performance. However, our claims are weakened by the inability of the system to load much larger corpora. To alleviate this, it may be necessary to consider alternative data structures to allow such corpora to be evaluated.

One type of data structure which would allow larger corpora to be evaluated is a non-memory resident data structure; that is, data structures which reside on disk rather than in memory. The most common example of this would be a database. Storing the phraselist data structure especially in a database would allow for huge corpora to be evaluated, with any phraselist queries being performed through standard database queries. Naturally, this would have implications for the input method as far as access speed is concerned (as the database access may exceed the 100ms cognitive limit); however, this is not an issue when determining the theoretical improvement provided by such corpora. The issue of the handling of enormous corpora (along with issues in real-world application thereof) is outside the scope of this research.

3.4.3 Impact of corpus size

In order to measure the impact of the corpus size with regards to the aforementioned response time and memory limitations, a pair of simple experiments were performed: the first to measure the

impact of the corpus size on memory usage, and the second to measure how the corpus size affects the response time of the system (bearing in mind the 100ms limit described in Section 3.4.1). The first set of experiments (described in Section 3.4.3) measure the relationship between the memory usage and the size of the corpus. The second experiments are described in Section 3.4.3 and aim to determine the average time required to enter a letter, word and phrase for various corpus sizes.

The corpora used for this set of experiments were generated from the `wikipedia` corpus (described in Chapter 4) by using the first n megabytes as a separate corpus. For each experiment, the full wordlist from the `wikipedia` corpus was used. Five corpora were generated in total, ranging from 10 to 50 megabytes in size (in 10 megabyte increments). Each of the experiments involved training the system on the given corpus, and evaluating the same corpus; however, the results themselves were discarded after the experiments.

Memory–corpus size relationship

Corpus size	Memory usage
10 MB	2256 MB
20 MB	4239 MB
30 MB	5168 MB
40 MB	5985 MB
50 MB	7077 MB

Table 3.1: Memory usage of varying corpus sizes

The memory usage of each corpus was calculated by determining the amount of memory used by the system every 15 seconds during the evaluation of the given corpus. The overall memory usage for each given corpus in Table 3.1 is the maximum memory usage reported during the running of the corresponding experi-

Figure 3.4: Relationship between memory usage and corpus size

ment. The maximum memory was used (rather than the average) due to the fact that a device without this memory figure would be unable to allocate enough memory to enable the use of the appropriate corpus.

The results shown in Table 3.1 and Figure 3.4 demonstrate a linear relationship between corpus size and memory usage. This is expected, due to the linear growth of the corpus. It is possible that the memory growth would begin to level once a certain threshold is reached due to a repetition of phrases. However, due to the number of possible word combinations in English (especially with a phrase length of 5, the default phrase window size as per Section 5.1.2), this is highly unlikely to occur within any reasonable memory limit, and was certainly not demonstrated in the range of corpus sizes investigated in these experiments.

Query time–corpus size relationship

Figure 3.5: Relationship between average response time per character and corpus size

Corpus size	Response time
10 MB	0.2554 ms
20 MB	0.6385 ms
30 MB	0.6132 ms
40 MB	0.7356 ms
50 MB	0.8084 ms

Table 3.2: Average response time per character of varying corpus sizes

The average response time per character refers to the average amount of time required between a character being entered and the system being ready to accept further input (including selecting an item from a word suggestion list). This figure was calculated

for each corpus by first calculating the total time required to enter a phrase: this was done by taking the current system time prior to, and at the conclusion of, entering each phrase. This time was then divided by the length of the phrase to give an average response time per character for each phrase, which was then averaged over the entire corpus.

The results shown in Table 3.2 and Figure 3.5 show a roughly linear relationship between corpus size and average response time per character (a spike in the 20 MB corpus is assumed to be an artifact caused by the machines on which the experiments were run). The most important thing from these results, however, is the fact that the response time never exceeded 1ms—considerably lower than the 100ms limit presented in Section 3.4.1. Bearing the linear relationship in mind, this limit can be reasonably disregarded for all corpora up to one gigabyte or larger in size.

Effects of corpus size on the system

From the previous two sections, it can be concluded that both the memory usage and response time of the system are linearly correlated with the size of the corpus used to train the system. This was an expected conclusion; however, the magnitude of the impact of these two relationships was slightly more surprising, as it was expected that the response time of the system would be more greatly impacted by the corpus size. However, this result means that the response time of the system is a concern which can be somewhat neglected for any corpus which is able to fit in memory given the current state of memory limits.

3.5 Chapter summary

This chapter detailed the design of the assistive input system, which included an in-depth look at the algorithm used by the system to retrieve word matches, suggestions and auto-completions. The chapter also listed the configuration flags and options available in the system, and described each in detail. Finally, the chapter includes a discussion on several of the limitations and considerations that went into the design of the system, and measures the impact of these issues. The following chapters discuss the corpora which were used by the system as training corpora and as testing corpora during the experiments.

Chapter 4

Corpora

The performance of an assistive input method is highly dependent on the corpus used as input data to train the system, to the extent that the suggestions made by all aspects of the system—including ambiguous keyboard and sentence-level word prediction—are entirely bound to the words and phrases in the corpus. For this reason, there are likely to be (potentially large) discrepancies in performance between each corpus; it is therefore important to determine the impact of different corpus qualities (such as corpus size and domain) on the performance of the system. Because of the strong connection between the system and the corpora used as training data, it is important to determine whether a given corpus is able to be used generally or not—that is, whether or not the speller allows the user to type words which may not be associated with the domain on which the input corpus is based. In order to evaluate these effects, several corpora of varying size, domain and generality have been constructed from various sources for use with the assistive typing system. By evaluating these corpora, we can determine the effects of, for example, using a large corpus versus a small.

The remainder of this chapter will describe the five corpora

used in order to evaluate the assistive input system developed for this thesis. Each corpus is given an in-depth description, along with how it was constructed, its unique qualities and a justification for its inclusion in this thesis. Before describing each corpus individually, the general process used to construct the word- and phraselist for the corpora is detailed. The chapter concludes with a summary.

4.1 Corpus construction

Before a corpus is able to be used by the assistive input system, it must first be parsed and converted to a format which the system expects. This format consists of two files: a wordlist file and a phraselist file, from which the wordlist and phraselist data structures are constructed respectively (for details on the purpose of these two data structures, refer to Sections 3.2.1 and 3.2.2).

The first file to be constructed for each corpus was the phraselist file. To convert the file into the appropriate format, the original data set was first combined into a single file, with any markup or other extraneous data removed (see Section 4.2 for per-corpus details). This file was then processed using a set of simple rules to create the phraselist. These rules exist for two reasons: firstly, the assistive input method is limited to alphabetical entry, and cannot insert numbers or punctuation. Secondly, the corpus must be formatted in such a way that each phrase appears on its own line. The specific rules are as follows:

1. All punctuation which does not denote the termination of a sentence is replaced with a space—this includes commas, colons and parentheses.
2. All apostrophes are removed.

3. All ampersands are replaced with the word *and*.
4. All punctuation which terminates a sentence is replaced with a newline character—this includes full stops, exclamation marks and question marks. Additionally, multiple occurrences of newline characters are replaced with a single newline character.
5. All other remaining non-letter characters are replaced with a space.
6. All occurrences of multiple spaces are replaced with a single space.
7. All upper case characters are converted to lower case.
8. Finally, all phrases containing special Unicode characters were removed (as these characters do not exist in English, and their appearance generally denotes non-English text).

After parsing the file with these rules, the remaining file is a collection of phrases containing only letters, with each phrase appearing on its own line and all extraneous whitespace removed. From this phraselist, the wordlist file can then be constructed. To do so, each word from the newly-created phraselist is counted, and the resulting list of words and their counts is then sorted (in descending order by number of occurrences) and stored in a separate file.

4.2 Corpora used

Each of the five corpora constructed for the purposes of this thesis have unique qualities which are evaluated in these experiments.

The `enron` and `newsgroup` corpora both represent written communication over the Internet—the `enron` corpus representing professional communication, and the `newsgroup` corpus representing a wide variety of subjects (generally written in a more casual, conversational style). The `fanfiction` corpus consists of fiction contributed by the users of a website, and incorporates a wide range of writing quality and formality; additionally, the corpus is an example of a huge corpus. The `ota-bawe` and `wikipedia` corpora are both examples of more research-oriented writing—the `ota-bawe` corpus containing published academic writing, while the `wikipedia` corpus contains user-contributed text.

4.2.1 `enron`

The `enron` corpus was constructed from the Enron email data set [13]. The Enron email data set is a collection of emails taken from the email sever of Enron Corporation which was released into the public domain by the Federal Energy Regulatory Commission (FERC) as part of its investigation into the company’s collapse and subsequent bankruptcy. The emails originate from 150 individual users, most of whom were part of senior management at the company. The data set is separated into individual user directories, each of which is further separated into a directory structure defined by the server and/or the user (for example, subdirectories of various users included “inbox”, “calendar” and “texas”). In each subdirectory was a collection of files, each of which containing a single email. Before being processed (as per Section 4.1), all the files located in each of the subdirectories were combined to create a single, large file.

Due to the nature of the data set, there are natural privacy concerns with regards to the content of the emails. While the emails

were released as part of the aforementioned FERC investigation, the version of the data set used in this thesis has been curated by researchers at Massachusetts Institute of Technology (MIT) and SRI International, who have removed several emails at the request of their authors. Because the emails included in the data set were not stripped of metadata (such as the sender and recipient(s) of the messages), the content of the emails are directly linked to their authors. However, this privacy concern is somewhat mitigated when constructing this corpus from the original data set, as the content of the messages are shuffled together in the phraselist, making it impossible to connect phrases to their author. Furthermore, there is no concept of phrase order in the phraselist, making it impossible to deduce the author of individual phrases.

The `enron` corpus is sized at 824 megabytes, and includes approximately 32.4 million lines, making it one of the larger corpora used in this thesis. The corpus was constructed to represent (mostly) professional emails sent between members of a corporate body. Furthermore, the data set was used to construct smaller sub-corpora from collections of individual users' emails, allowing the impact of using a personalized corpus compared with a general one to be measured. Many of the emails included were very short, reflecting the nature of communication via email; it is expected that this corpus would be most useful when training an assistive input method for forms of written communication over the Internet. Additionally, the email metadata included in the corpus may assist users in writing messages in this format.

Individual user corpora

A set of personalized corpora were created from the writing samples of various individuals within the `enron` data set in order to determine the impacted of training the assistive input system

with a corpus tailored to its user. In order for a corpus to be accurately representative of a user's writing style, the corpus must consist of a reasonably large amount of text. The amount of text in the corpus for a given user is represented by the file size of each user's text. Therefore, this file size was used to determine which users would be appropriate for the construction of personalized corpora. I chose users with at least 20 megabytes of text, which resulted in seven individual, personalized corpora—these corpora were then constructed by simply combining each of the files in the user's directory and running the standard scripts for corpus construction (as per Section 4.1).

4.2.2 **newsgroup**

The `newsgroup` corpus contains extracts from was constructed from the *20 Newsgroups* data set [62], a collection of around 20,000 postings to 20 different public newsgroups. The data set was originally constructed in 1995 by Ken Lang [43]. The data set contains postings from over 8,000 users, on topics ranging from sport, technology and science, to religion, automobiles and classified advertisements. Like the `enron` data set, `newsgroup` is split into sub-directories based on the newsgroup each message was posted to, with each message occupying its own file; therefore, it was also required to combine these into a single file before processing it (as per Section 4.1).

The average message consists of short header, which includes the sender's name and email address, followed by a subject line. Following this header, the message body begins. Messages which were replies to other messages often begin with the original message quoted, thus increasing the number of occurrences of the original text. Following the quoted text is the sender's message.

There are likely to be some quirks owing to this structure—for example, phrases which are included in a subject line (or even an email address) are likely to be repeated many times, especially for posts which accumulated many replies. On top of messages being quoted in replies, this may skew the n -grams extracted from the corpus due to their perceived popularity.

Due to the nature of newsgroup postings, this corpus is sized at a very small 28 megabytes. Similar to the `enron` corpus, the `newsgroup` corpus represents written Internet communication, which represents a fairly unique form of the English language. The `newsgroup` corpus differs from `enron`, however, in that the context is more casual and the subject matter generally has a more recreational nature.

4.2.3 `fanfiction`

The term “fan fiction” refers to any piece of work containing characters and/or settings of another piece of work, which has been authored by someone other than the original creator. The Internet provides a popular means of distributing these works to other fans, and as such, there are many large databases of fan fiction available on the Web. One of the more popular repositories is FanFiction.net [48], a website containing more than 5 million stories written by more than 2 million registered users¹. This data set was constructed by automatically scraping the text from a large portion of stories found on the website. Because of the huge amount of text obtained from FanFiction.net, it was not possible to combine this into a single large corpus file.

The `fanfiction` corpus was constructed due to the expected similarity between the average writing style and everyday En-

¹As of 28th January 2014

glish. It is expected that the quality of the fan fiction writing will be similar to that of the average user of an assistive typing technology, due in part to the fact that the authorship of the majority of the fan fiction is amateur. This results in the use of a much smaller vocabulary when compared to professional writing—this is appropriate as people generally use a far smaller vocabulary when compared to written works [5, 20]. The range of topics is also likely to be very large, given the number of different source materials from whence the corpus is constructed. Finally, fan fiction tends to involve a lot of speech between characters (similar to—if not more so than—published fiction), which lends itself well to any assistive input method used for communication. However, the major downfall is the occurrence of unusual names and themes which are prevalent amongst works of fantasy or science fiction. With all this in mind, it is likely that the `fanfiction` corpus will perform best with the `enron` and `newsgroup` corpora; however, the fictional nature of the corpus could result in unusual suggestions from the sentence-level word prediction system.

The `fanfiction` corpus is the largest corpus created for this thesis—too large, in fact, to be used with the assistive input system regardless of configuration. The entire corpus is 66 gigabytes large; at this size, it would be impossible to load the entire corpus into the memory of a regular workstation. In order to use this corpus in its entirety, it would be necessary to use a data structure such as those outlined in Section 3.4.2. In order to evaluate the corpus without requiring a major re-engineering of the system, a small sample of the corpus was taken. This sample was constructed by taking a single line at random from each file in the original data set. This resulted in a much smaller corpus of only 529 megabytes.

4.2.4 ota-bawe

The ota-bawe corpus [73] was originally created for use in research into the genres of student academic writing [26]. The corpus was made available to the public, and consists of over 2,700 pieces of student writing, each of which contains between 500 and 5000 words. The work was obtained from students studying subjects across one of four disciplines—Arts and Humanities, Social Sciences, Life Sciences and Physical Sciences—at either undergraduate or (taught) Masters level.

Along with the wikipedia corpus, this corpus contains extracts of academic- or research-style writing. However, unlike that corpus, the ota-bawe corpus covers a smaller selection of disciplines and topics. On the other hand, it is expected that this corpus will have a more consistent writing style and quality, as each source material was chosen as an example of “proficient” student writing (as opposed to the wikipedia corpus, which is unregulated and is therefore of uncertain quality). It is likely that this corpus will show the best performance when compared with the wikipedia corpus, and much poorer performance with corpora constructed from conversational language, such as the newsgroup corpus.

At only 34 megabytes, the ota-bawe corpus is one of the smaller corpora used in this thesis. However, unlike the similarly small newsgroup corpus, this corpus contains a much higher proportion of body text (whereas the newsgroup corpus contains a large amount of metadata also). Each piece of work in the original data set was located in its own file, which contained additional markup denoting headings and figures, amongst other information. Before the data set was parsed for use with the assistive input system (as per Section 4.1), this markup data was removed, and the files were combined into a single source file.

4.2.5 wikipedia

Wikipedia is commonly regarded as one of the largest available sources of human knowledge, with more than 4.4 million English articles available². This makes it far larger than contemporary encyclopaedias—by comparison, the *Encyclopedia Britannica* is smaller by several orders of magnitude, consisting of fewer than 50,000 articles [22]. The aim of Wikipedia is to provide a source of information far more detailed and varied than that of a traditional Encyclopedia; to achieve this, all the content on the site is contributed by its users. While the benefit is clear from the sheer scale of information to be found on Wikipedia, using exclusively user-contributed information introduces concerns regarding factual accuracy and the quality of writing—especially in “fringe” articles, which are rarely viewed or edited.

That all Wikipedia articles have been contributed by a large range of users, combined with the huge variety of information on the website, makes it an interesting corpus. The differences in quality of writing means it should be able to cover a wider variety of user writing styles, while the overall more formal style lends itself well to users wishing to write in a similar style. The wide range of information contained in Wikipedia also allows for a wide variety of concepts to be expressed by users—however, the topics in the corpus may not necessarily lend themselves well to conversational English. Combined with the formal writing style, it is expected that the Wikipedia corpus may be less useful in real-world applications where the input method is used in an everyday setting.

The Wikipedia corpus is 381 megabytes in size, totalling approximately 3.5 million lines. The corpus was constructed using Wikipedia Miner [54], a toolkit developed by the University

²As of January 2014 [77].

of Waikato for querying a snapshot of Wikipedia from a specific date³. The corpus consists of a single sentence from almost every article in the Wikipedia snapshot. Sentences were required to meet a set of simple criteria (specifically, the sentences had to be from body text and not headings, and had to be a minimum of 20 characters in length); for this reason, several articles were devoid of appropriate sentences. Nevertheless, the vast majority of articles are represented by the corpus, making it a varied collection of English concepts and vocabulary. However, only retrieving a single sentence from each article could have the unwanted effect of constructing a somewhat diluted corpus, therefore resulting in domain-specific terminology occurring infrequently and appearing at the end of both the word- and phraselists.

4.3 Corpus resizing

Due to the memory issues mentioned in Section 3.4.2, many of the corpora were too large to be loaded in by the system. As a result of this, smaller versions of the corpora were constructed as needed. These were constructed by taking a subset of the phrases (either at random or by some other determined order) and constructing a new phraselist—the existing wordlist was used, as the high memory usage was largely caused by the phraselist. The details of how the smaller versions of each corpus was created (and the size of these smaller corpora) is detailed in this section.

The smaller versions of each corpus were created by shrinking the phraselists to approximately 50 megabytes or smaller in order to comply with the aforementioned memory limitations. The phraselist was the target of shrinking moreso than the wordlist due to the higher memory usage of that particular data structure.

³The snapshot used to create this corpus was dated 22nd July, 2011.

The performed method of shrinking each corpus was to shuffle the phraselist, and keep the top n phrases such that the resulting phraselist was around the desired target size of 50 megabytes. The details of each of the corpora are as follows:

- `enron`: 51 megabytes, approximately 2 million phrases.
- `newsgroup`: 28 megabytes, approximately 900,000 phrases (not shrunk).
- `fanfiction`: 49 megabytes, 900,000 phrases (shrunk from the smaller 528 megabyte corpus; see Section 4.2.3).
- `ota-bawe`: 34 megabytes, approximately 250,000 phrases (not shrunk).
- `wikipedia`: 53 megabytes, 500,000 phrases.

4.4 Chapter summary

This chapter described the process of obtaining, parsing and constructing the corpora used for the purposes of this thesis. The benefits and unique features of each corpus were also detailed, along with a brief comparison between similar corpora. These comparisons also include predictions regarding the expected performance of the assistive input method when trained with and tested against various corpora. Chapter 5 contains a formalization of these expectations in the form of hypotheses, followed by a description of the experiments which were performed in order to confirm or reject these hypotheses (amongst others). Subsequent chapters will also discuss the results of these experiments and the relationship between corpora and system performance.

Chapter 5

Evaluation

The assistive input system developed as part of this research was designed to allow the investigation of effects of various parts which make up the system. By investigating these aspects of the system, we can determine how they impact the theoretical maximum typing speed of the system and, by extension, how these aspects can be combined to create an optimal assistive input method. In order to investigate these characteristics of the system, an evaluation framework was constructed around the input method. This was built in lieu of an interactive input system (which could theoretically be added to the system and run in parallel with the evaluation framework). The evaluation framework allows us to empirically measure the impact of the different parts of the system by evaluating variations in configuration, which allows us to achieve one of the major goals of this research.

In Section 1.3, the goals of this research were outlined. Two of these goals refer to evaluating the impact of the corpus used to train the assistive input system, and the impact of all other configuration options. The first of these goals was based on the conjecture that the training corpus has a potentially large impact on its performance when used to input text, especially in other domains;

the second goal aims to quantify the impact of various other configuration options.

This chapter will discuss the process used to evaluate the assistive input method with a given training and testing corpus. Following this is an in-depth description of each of the experiments performed during the undertaking of this research, where we formalize the expectations of the system in the form of hypotheses.

5.1 Evaluation methodology

In order to evaluate the assistive input method, we use a simple measure of keystroke savings. The keystroke savings is calculated by subtracting the number of keystrokes required to type a sentence using the system from the number of keystrokes required to input the phrase using a standard keyboard. This measure has been used in prior studies [34, 46, 51] and represents a simple, yet accurate, performance measure for assistive input methods. Because the keystroke savings is correlated with the length of the phrases, however, the performance of the system can also be measured using a common metric known as *keystrokes per character* (KSPC). This metric is used in some of the previous literature [45], and can be used to compare performance independent of the varying phrase lengths. A major advantage of the KSPC measure is that it is intuitive—punctuation and capitalization notwithstanding, a standard keyboard has an average KSPC of 1.0, as every alphanumeric character occupies its own key. Therefore, by treating a KSPC of 1.0 as a baseline, the performance of an input method can be directly compared to that of a standard keyboard. This measure will be referred to heavily in Chapter 6 when analysing the results.

Evaluating the system involves two additional input data struc-

tures (on top of the training corpus—the wordlist and the phraselist—and the ambiguous keyboard split) required to perform an experiment: the testing corpus and a configuration file. The testing corpus is the phraselist of one of the corpora described in Chapter 4, the entry of the phrases from which are simulated in order to determine the keystroke savings provided by the assistive input method when used to type the phrase in question. The configuration file determines which subsystems are to be enabled during evaluation, and specifies the state of all other configuration flags and options (described in Section 3.3). The testing corpus provided to the system during these experiments is designed to be representative of a certain style, tone or domain of writing; by varying the testing corpus presented to the system, we can collect system performance data in various domains. At the conclusion of the evaluation of a testing corpus, the performance of the speller configuration—and training and testing corpus—is reported as the percentage of keystrokes saved.

There are many aspects besides raw keystroke savings which can be taken into account when evaluating the performance of an assistive input method, such as the cognitive cost of auto-completion, or the cost breaking the flow of typing by selecting a word from a suggestion list. However, many of these aspects are not considered during the evaluation of this system. This was done to simplify the evaluation of the system. While the performance of the system as reported by the evaluation may be slightly less meaningful due to failing to consider these aspects, because the evaluation compares features and corpora (and not necessarily the impact of the assistive typing system in a real-world scenario), this was considered to be an acceptable limitation.

5.1.1 Process

The evaluation of phrases is enabled by the evaluation framework created as part of the assistive input method. This evaluation framework measures the keystroke savings by simulating a user typing using both the assistive input method and a standard keyboard. To be more specific, the system steps along each character of the input phrase and keeps track of the number of keystrokes entered at each point. For simulating a standard keyboard, this is simplified by simply counting the number of characters in the sentence; the assistive input system, however, needs to step over the phrase one character at a time. This is necessary due to the system needing to present a list of ambiguous word completions, word suggestions and word auto-completions at each step, which each have a different cost based on a number of variables (including the characters entered to that point).

How the system responds to user input (in this case, simulated user input) was covered in detail in Section 3.2.3. This Section also details the cost associated with each action the user can take during typing with the system. These costs are summarized as follows:

- **Entering a character** is performed at a cost of 1 keystroke.
- To **accept an auto-completion** the keystroke cost is 1; this also enters a space or sentence-terminating punctuation mark.
- **Rejecting a character** is performed at a cost of 1 keystroke—this must be done before entering a space or sentence-terminating punctuation mark where an undesired word has been auto-completed.
- The cost of **selecting a different ambiguous word completion** is based on the position of the desired word in the am-

ambiguous key sequence matches list (for example, if the desired word is at position 3 in the list, the cost is 2 keystrokes). If the word does not exist in the list, the cost is $n + w + 1$, where n is the number of items in the list and w is the length of the word being entered.

- **Selecting a word prediction suggestion** costs 2 keystrokes: the first to enter word selection mode, the second to select the desired word.

For each completed corpus evaluation, the output is a CSV file containing the results of evaluating each phrase in the testing data set. The format of each line of the file is as follows:

```
keystroke-cost-raw , keystroke-cost-predictive ,
keystroke-savings , evaluation-time , phrase
```

5.1.2 Default configuration

Many of the experiments performed as part of this research and described in this chapter vary only partially, and are otherwise similar. For example, the experiments comparing the effects of using different training corpora use the same values for the tuning parameters in order to make any comparisons between corpora valid. For this reason, the concept of a *default configuration* is hereby defined as follows (refer to Section 3.3 for an in-depth explanation of each of these options):

- All subsystems—ambiguous keyboard, sentence-level word prediction, word auto-completion—are enabled.
- The keyboard is split into the following groups: SNWZXOF, AUCJEVB, YIDPKL, QHGRMT.

- Variable phrase windows are enabled and set to size 5.
- Infrequent phrases are removed (threshold is 5).
- Infrequent words are **not** considered bad words.
- Long words are considered bad words (maximum word length is 20).
- The bad word behaviour is set to **drop contains**.

The keyboard split used in the default configuration was the best performing of the original naïve search for keyboard splits described in 5.2.4. A default phrase window size of 5 ensures the experiments performed are novel, as previous studies often limited the extracted phrases to bigrams or trigrams. Infrequent phrases were removed in order to cut down the memory usage of the phraselist, and long words were also removed due to them often not being real words beyond a large number of characters. Finally, the bad word behaviour was chosen as **drop contains**, as this is the most straightforward—as well as memory-efficient—behaviour.

5.2 Experimental design

The remainder of this chapter will design a set of experiments necessary to determine the effect the various aspects of the system have on the overall performance of the system in a variety of contexts. In doing so, we will achieve the goals set out in Section 1.3 and understand what is required to create an assistive input method which demonstrates optimal performance in the majority of situations.

The performed experiments need to several aspects of the system, each of which will have varying effects on the performance

of the assistive input system. The first two sets of experiments aim to determine the effect of the corpus used to train the system, with the first looking at the performance differences with various combinations of training and testing corpus, while the second set considers the effects of using the same corpus for training and testing. The third collection of experiments consider the effect of the keyboard split on the performance of the system. The fourth experiments consider the impact of each of the three major sub-systems, while the fifth set of experiments looks into the effects of the other tuning parameters.

5.2.1 Inter-corpus evaluation

Inter-corpus evaluation refers to the process of evaluating a training corpus by measuring its performance when typing phrases from a second corpus. This is repeated for every pair of corpora detailed in Chapter 4. By performing this set of evaluations, we are able to make inferences regarding the impact of domain and writing style on the performance of a system independent of other system parameters.

This evaluation is performed by constructing an “evaluation matrix” of all of the corpora listed in Chapter 4. The evaluation matrix covers every unique pairing of corpora; by completing the matrix, we will have a complete comparison of every corpora created for this research. In order to complete the matrix, each cell is filled with the results of an experiment performed with one corpus used to train the system and the other used to evaluate it. Each of these experiments uses the default configuration, with only the corpora used to train and evaluate the system used as variables.

	enron	fanfic.	news.	ota-bawe	wiki.
enron					
fanfic.					
news.					
ota-bawe					
wiki.					

Table 5.1: Evaluation matrix for inter-corpus evaluation (the grey cells indicate control experiments)

Hypotheses

In Chapter 4, several predictions were provided regarding the results of these experiments, which were related to the domain and writing style of the corpora. The predictions were made by essentially categorizing the corpora according to these two aspects. This categorization can be simplified as follows:

Corpus	Domain	Style
enron	Communication	Formal
fanfiction	Conversational	Casual
newsgroup	Communication	Casual
ota-bawe	Academic	Formal
wikipedia	Academic	Semi-formal

Table 5.2: Categorization of corpora

From Table 5.2, we can determine which corpora are expected to perform well as training/testing corpus pairs. One such expectation is for data sets originating from the same domain to exhibit better performance. This is formalized as the following hypothesis:

Hypothesis 1 *Corpora pairings which share the same domain will outperform those from different domains.*

Similarly, we also expect that corpora which share similar writing styles to also demonstrate good performance:

Hypothesis 2 *Corpora pairings which exhibit similar writing styles will outperform those with different styles.*

From Hypotheses 1 and 2, we expect that the `ota-bawe` and `wikipedia` corpora will demonstrate the best performance of any pair. Other potential high-performance matches include the `enron` and `newsgroup` pairing (due to the domain match) and the `enron` and `ota-bawe` pairing (due to the similar writing styles).

5.2.2 Single corpus evaluation

While the experiments described in the previous section are designed to measure the performance of evaluating each corpus against every other corpus to determine the effect of domain and style, the single corpus experiments measure the performance of each corpus independently. Measuring the ability of a corpus to effectively type phrases from itself reveals information about the size of the domain contained within (i.e. how specific the domain is)—for example, a corpus which demonstrates high performance when typing its own phrases would be expected to perform well in situations where phrases from this domain is to be entered by a user. Furthermore, a high-performing corpus is likely to require a smaller training corpus in order to cover a much higher percentage of possible phrases typed by a user. The results of these experiments will inform us which corpora are most useful in domain-specific settings, and which are likely to be more useful as general corpora. Combined with the knowledge about the intended target domain of the user, we could also use the results of these experiments to determine which corpus is likely to be most useful for a given user.

These experiments are similar to those described in Section 5.2.1, with the only difference being the reuse of the training cor-

pus as the testing corpus in all instances. However, in order to alleviate bias, a form of cross-validation was used in the experiments [41]. This 10-fold cross-validation involved dividing the phraselist of each corpus at random into 10 equal parts. At this point, 10 separate experiments were performed on each corpus, with each experiment using a unique part as the testing corpus a combination of the remaining 9 corpus parts as the training corpus. The final step of the cross-validation process was to average the results of the 10 experiments; this average is then used as the performance measure of the corpus being evaluated. Finally, like the inter-corpus evaluations, the default configuration is used for other flags and options.

Hypothesis

It is expected that corpora with a smaller, more specific domain are likely to perform better in a single-corpus experiment than those corpora which are cover a much wider range of topics, as larger domains are likely to cover a larger set of common phrases. Therefore, corpora constructed from academic writing is likely to see a lesser benefit from collocations than a more specific domain such as conversational English or communication:

Hypothesis 3 *A corpus taken from a smaller domain will perform better than one taken from a larger domain.*

Based on Hypothesis 3, we can expect that the performance of the `fanfiction` corpus will be higher than that of the `wikipedia` corpus in these experiments.

5.2.3 Personalized corpora evaluation

One potentially interesting question which can be asked of the assistive typing system is whether training the system with a data

set consisting of samples of the user's typing will result in better performance for that user when compared with training the system using text written by others. In other words, these experiments investigate the impact of using a personalized corpus versus using a generalized corpus.

Because every person exhibits a unique personal writing style, it is natural to assume that training the system using a corpus constructed from the user's own typing will be demonstrate higher performance when compared with other corpora. This is due to the fact that people tend to reuse the same phrases, which increases the impact of collocations (words that occur together frequently; see Section 1.1.2 for more details). It is therefore expected that the performance of the system will be greatly improved by using a corpus which is personalized to the user's exact writing style.

These experiments make use of the personalized corpora which were constructed from the `enron` corpus described in Section 4.2.1. Two sets of experiments were performed which roughly correspond to the experiments performed with general corpora and described in Sections 5.2.1 and 5.2.2. The first set of experiments involved measuring the performance of each user's personal corpus against a combination of the remaining personalized corpora (that is, the combined corpus of every user *except* that which was used to train the system). This was done in order to establish a baseline measure for the system trained using a personalized corpus. The second set experiments measured the impact of testing the system with the same user's corpus used for training. Similar to the single corpus evaluation, a form of cross-validation was used (however, due to the size of the individual user corpora, only five divisions were used—that is, 5-fold cross validation); for details on this process, see Section 5.2.2. After performing these ex-

periments, the results of the two were directly compared to measure the impact of using a corpus personalized for a given user.

However, one downfall of using the `enron` corpus as a source of personalized corpora is due to the way replies to other users are handled within the corpus. Because messages that were sent by an individual in reply to another user often included the original message, part of the corpus personalized to a user will be diluted somewhat with text that was written by another user. This is expected to somewhat reduce the performance difference between the system trained and tested with a single user's corpus and the system trained and tested with two different users' corpora. However, this dilution is minimal enough that we should still be able to clearly see a performance benefit in most cases.

Hypothesis

Personalized corpora are beneficial to the assistive input method as they allow the user to type their most oft-repeated phrases with greater efficiency by promoting only phrases and words which the user themselves has previously typed. In so doing, it is therefore intuitive to expect that through using a corpus provided by samples of the users text we can maximize the performance of retrieval from the word- and phraselists. Therefore:

Hypothesis 4 *A corpus which has been personalized to a user will allow the user to obtain better performance from the system when compared with a general corpus.*

5.2.4 Keyboard split evaluation

The ambiguous keyboard subsystem is closely linked to—and highly dependent on—the way the keyboard is split or grouped. This keyboard split refers to how English letters are mapped to the

abstract input keys of the ambiguous keyboard (see Section 3.3.2 for a more detailed explanation). For this reason, its performance (and by association, the performance of the entire system) can be impacted greatly by this split.

Evaluating the impact of the keyboard split was done by first creating several candidate keyboard splits. This was done by running a naïve search over random combinations of letters split into 4 equally-sized groups, and measuring the performance of these splits using a simple estimated performance measure. This measure was taken as a combination of two performance measures: an ambiguity measure and a range measure. The ambiguity measure is simply the sum of the number of ambiguous keyboard matches for each word in a given a collection of sample text, while the range measure is the squared average of the range of sizes of the ambiguous matches lists for each input key sequence length (i.e. the difference between the most ambiguous input sequence and the least ambiguous input sequence for each sequence length). Several of the best performing sequences from these initial experiments were used in the final keyboard split experiments. Note that the top performing keyboard split from these initial tests was used as the keyboard split in the default configuration from Section 5.1.2.

The list of candidate splits was appended with three splits which are ordered in some way: an alphabetical split, an alphabetical split with vowels separated, and a split with the characters ordered according to a QWERTY keyboard. These splits are included so that the impact the order of ambiguous keyboard groups can be measured. By measuring this impact, we can determine whether being able to use an unordered split (such is the case for BCI spellers, and described in Section 1.2.1) is advantageous. The candidate splits are listed in Table 5.3.

Group 1	Group 2	Group 3	Group 4
SNWZXOF	AUCJEVB	YIDPKL	QHGRMT
CEGLSUZ	HMNKWAX	FJVYIT	PRODQB
WPFERVY	GDATEKB	JCHNLO	ZMUSQI
LOCVGDQ	HMRIYZK	FETJBU	APWNSX
URIQPF	CSXOJKH	GYVTAM	NWLEBZ
NWZJEBH	GMXAKUT	CDRPOV	SLQIFY
QWERTYU	IOPASDF	GHJKLZ	XCVBNM
ABCDEFGH	HIJKLMN	OPQRST	UVWXYZ
AEIOUY	BCDFGHJ	KLMNPQR	STVWXZ

Table 5.3: Keyboard splits investigated

After constructing the list of candidate splits, the experiments involved simply swapping out the keyboard split for another. Each keyboard split was tested for each corpus pairing (as per Section 5.2.1), with the performance of each split reported as an average of the measured performance of each corpus pairing—that is, the performance of a single keyboard split is the average of the performance from every corpus pairing using the keyboard split.

Hypotheses

From the initial tests performed in order to construct the list of candidate splits, it was noticed that the way the keyboard was split has the potential to noticeably affect the performance of the ambiguous keyboard. Furthermore, it is expected that the performance of the ambiguous keyboard using unordered splits will be much higher than when using an ordered split. These hypotheses are as follows:

Hypothesis 5 *The keyboard split of the ambiguous keyboard has a significant impact on the performance of the system.*

Hypothesis 6 *An optimal unordered keyboard split can achieve better performance than an optimal ordered split.*

5.2.5 Subsystem evaluation

The assistive input system is made up of three separate subsystems: the ambiguous keyboard, sentence-level word prediction and word auto-completion. This means that the performance of the system is dependent on these three individual subsystems and the interaction between them. These three components have somewhat competing effects on the maximum theoretical typing speed using the assistive system, however; while the word prediction and auto-completion subsystems both aim to increase the speed at which a user can type, the ambiguous keyboard instead attempts to reduce the number of physical keys required to type using the system whilst minimizing the negative affect on typing speed. By analysing the performance each of the subsystems in isolation, we can determine the effects that each have on the performance of the system. This information can also assist in situations where it may be beneficial to remove a subsystem to maximize performance of the overall input system—for example, in situations where a full keyboard is able to be used, it may be better to remove the ambiguous keyboard subsystem. Due to the reliance of the word auto-completion system on the prediction engine (it does not function without this system enabled), this component should not be enabled without sentence-level word prediction.

In order to measure the effects that each subsystem has on the overall system, it is necessary to simply disable the required components using the flags defined in Section 3.3.1. The experiments performed involved every permutation of the three subsystems, excepting all subsystems enabled (a duplicate of the experiments in Section 5.2.1), all subsystems disabled (simply a standard keyboard), and situations where the auto-completion component was enabled without the word prediction subsystem. This resulted

in four sets of experiments—three permutations with word auto-completion disabled, and one with only the ambiguous keyboard disabled. The performed experiments were otherwise identical to the inter-corpus experiments of Section 5.2.1, which involved measuring the performance of every unique corpus pairing. By performing these experiments in such a way that is closely analogous to the inter-corpus ones, we are able to compare the results in order to determine how the performance of each permutation compares to the default set-up (that is, all three subsystems enabled).

Hypotheses

As previously mentioned, the three components of the assistive input system are not tailored to achieve a common goal. In fact, the absolute best case scenario for the ambiguous keyboard is to achieve the same performance as a standard keyboard—it will never be able to outperform it (in the majority of cases, even this is an unreasonable goal). Therefore, the first hypothesis of these experiments is as follows:

Hypothesis 7 *The ambiguous keyboard has a negative impact on the performance of the system.*

The other two components of the system—sentence-level word prediction and word auto-completion—both work to reduce the number of keystrokes required to input text into the system. Therefore, both of these subsystems are expected to benefit the performance of the system:

Hypothesis 8 *Sentence-level word prediction greatly increases the performance of the system.*

Hypothesis 9 *Word auto-completion greatly increases the performance of the system.*

From the above Hypotheses 7, 8 and 9, we can expect that configurations which include the ambiguous keyboard will exhibit worse performance than the corresponding configuration with the ambiguous keyboard disabled, and that configurations with either sentence-level word prediction or word auto-completion will perform better than those with these components disabled. On top of this, we expect that the benefit provided by these two systems is accumulative, and therefore that a configuration with both of these components enabled will outperform configurations with one or the other enabled.

5.2.6 Tuning parameters evaluation

During the construction of the assistive typing system, several tunable parameters (detailed in Section 3.3.2) were integrated with the system so that their effects could be measured. By determining the effects of various tuning parameters on the performance of the system, we can establish the optimal configuration for this input method. By comparing the results of the tuning parameters for different corpora, we can additionally gain an understanding of whether some parameters should be tuned for specific use cases, or whether a global optimum can be achieved.

The experiments designed to measure each parameter is again similar to that of Section 5.2.1, with the default configuration of Section 5.1.2 being used for every parameter other than that which is being tuned. The performance of each parameter is reported per-corpora pairing; in doing so, we can achieve the above goal of determining whether the parameters can be tuned globally or whether they should be tuned on a case-by-case basis. The fol-

lowing is a list of tuning parameters which were covered in these experiments:

- Variable phrase length
- Bad word behaviour
- Bad word and phrase thresholds
- Size of word prediction suggestion list

Hypothesis

The parameters were included as part of the design of the system to investigate the effects of changing these parameters—the effects of each parameter is not known, but it is expected that they will all have some impact on the performance of the system. Therefore, the hypothesis for these experiments is as follows:

Hypothesis 10 *Changing the values and behaviours of the different parameters will have an effect on the performance of the system.*

5.3 Chapter summary

This chapter detailed the experiments which were carried out in order to determine the effects of various aspects of the system, thereby achieving the goals described in Section 1.3. This chapter also detailed the process used to perform each experiment, and the default configuration used to ensure consistency between experiments. The results of each experiment are reported and analysed in Chapter 6, which is followed in the same chapter with a discussion. The findings of this research is summarized and the discussion concluded in Chapter 7.

Chapter 6

Results

The experiments outlined in the previous chapter were designed to evaluate the performance that various configuration options have on the performance of the assistive input system. By evaluating the results of these experiments, we can draw conclusions for the hypotheses which were also detailed in Chapter 5. Through these conclusions, the goals of this research can be achieved. This chapter will first explain how the results were analysed, and how the results should be interpreted. Section 6.3 presents the entire set of results from the experiments presented in the previous chapter. Each of the hypotheses that were presented in Chapter 5 are also given conclusions in this section—the numbers of the conclusions directly correspond to the numbers of the hypotheses. The chapter will conclude with a brief summary, and the thesis concluded in the following chapter.

6.1 Data preprocessing

Prior to analysing the results of each set of experiments, the data was first pre-processed. By doing so, the data was normalized in such a way that the results obtained from each test corpus can be

compared. Without normalizing the results, corpus aberrations such as abnormal sentence lengths could skew the results of one test corpus compared to the others. The full set of pre-processing operations is described as follows.

The first processing step that was performed was limiting the length of the phrases within the test corpus. This involved both removing phrases which were considered to be too short or too long to contribute to the performance of the system in a real world situation. This limitation was introduced in response to the `fanfiction-small` corpus, the low-quality writing of which contains phrases of lengths which are not representative of standard use of an assistive input system. The accepted phrases were allowed fairly liberal lengths, with valid phrases varying from 1 to 1,000 characters in length. Empty phrases were removed, as these potentially skew the results without reflecting true use of the system; no other phrases with few characters were removed, however, as even single characters have the potential to be entered in real-world contexts. On the other hand, phrases beyond a certain length would be considered to be grammatically incorrect, hence the removal of such phrases from the result data; the high character limit is so as to not artificially penalize long (but valid) phrases.

The second processing step involved the removal of duplicate phrases. Duplicate phrases were removed in order to evenly weight every phrase which is being evaluated. Ensuring that every phrase is unique ensures that no phrase is over-represented within a given test corpus. Despite uniqueness of phrases not being a requirement of entering text in a real-world situation, this requirement during the evaluation of the system ensures that the performance is fairly representative of the overall average performance of the system.

One processing step which is worth mentioning on account of

it not being performed is the removal of outliers. While removing phrases which exhibited abnormal behaviour in the context of the rest of the corpus would be a reasonable action to take from a statistical point of view, this has the potential to somewhat falsify the results, and result in reporting incorrect performance figures. From a performance perspective, outliers are generally observed as a result of phrases which contain words which are not in the training corpus; this is especially true of shorter words, as the cost of entering a word is much higher with a larger ambiguous word match list (see Section 3.2.3 for more details). Because these situations are possible through real-world use, outliers are instead treated as per usual when analysing the results.

6.2 Analysing results

Analysis of the results was performed with scripts written using the statistical programming language R [60], and the libraries `data.table` [18], `ggplot2` [76] and `lattice` [65]. After loading in the data, the scripts first processed the results as described in Section 6.1; as such, the scripts make up a crucial part of the experimental framework. The figures presented in this chapter were also generated using these scripts.

Throughout this chapter, the pseudo-median of each experiment will be presented as an estimator of the population mean, which was generated using the Wilcoxon test across the sets of experiments. The pseudo-median statistic reported for each result is the Hodges-Lehmann estimator [33], which is closely related to the population mean in non-symmetric data such as is contained within these results. For each set of experiments, the pseudo-median was calculated with a 95% confidence interval, which was generated over the entire set of experiments using a confidence of

$1 - \frac{1 - \text{confidence}}{n}$, where n is the number of experiments. After generating these intervals, Bonferroni correction [21] was applied over the set to give a 95% confidence interval for each individual experiment. For example, a confidence of 99.8% was used over the set of 25 inter-corpus experiments, which resulted in a per-experiment confidence of 95%.

For each set of data, the results are reported as the average keystroke per character (KSPC) of the given corpus or configuration, as KSPC is used as a performance measure. The average KSPC is compared to a KSPC of 1.0, which is the KSPC rate of typing with a standard keyboard (at least in the case of lower case letters; the average KSPC is increased when considering the SHIFT key). Therefore, the lower KSPC the better the performance of the system; a KSPC lower than 1.0 is a target of the majority of the experiments. The aforementioned pseudo-median therefore refers to the median of the average KSPC across all the corpora and/or configurations. The other reported figures are the standard deviation (s.d.) and the previously-explained confidence intervals (CIs).

6.3 Experimental results

This section discusses the results of the experiments designed in Chapter 5. For each experiment, the raw data will be presented along with a summary of the most important data. The data will also be investigated with regards to the hypotheses that were detailed within the previous chapter, so that these can be accepted or rejected.

Train.	Test	P-median	s.d.	CIs
enron	enron	1.0335	0.9414	[1.0314,1.0355]
enron	fanfic.	1.3519	0.6301	[1.3504,1.3533]
enron	news.	1.3881	1.6414	[1.3848,1.3915]
enron	ota-bawe	1.0999	0.2995	[1.0985,1.1014]
enron	wiki.	1.2549	0.4907	[1.2536,1.2562]
fanfic.	enron	1.5933	6.6388	[1.5867,1.5999]
fanfic.	fanfic.	0.8806	0.3471	[0.8800,0.8812]
fanfic.	news.	1.3549	8.8267	[1.3482,1.3617]
fanfic.	ota-bawe	1.0084	0.7738	[1.0074,1.0093]
fanfic.	wiki.	1.1473	2.0935	[1.1454,1.1492]
news.	enron	1.4160	0.8620	[1.4135,1.4185]
news.	fanfic.	1.2716	0.5121	[1.2704,1.2729]
news.	news.	1.2000	1.2584	[1.1972,1.2029]
news.	ota-bawe	1.1073	0.2581	[1.1060,1.1086]
news.	wiki.	1.2413	0.4086	[1.2401,1.2425]
ota-bawe	enron	1.3362	1.1269	[1.3332,1.3391]
ota-bawe	fanfic.	1.2166	0.6122	[1.2155,1.2178]
ota-bawe	news.	1.2781	1.5432	[1.2744,1.2818]
ota-bawe	ota-bawe	0.7682	0.1645	[0.7675,0.7689]
ota-bawe	wiki.	1.0280	0.4760	[1.0268,1.0292]
wiki.	enron	1.3305	2.8618	[1.3268,1.3343]
wiki.	fanfic.	1.1819	1.3571	[1.1807,1.1831]
wiki.	news.	1.2120	4.2766	[1.2083,1.2158]
wiki.	ota-bawe	0.8566	0.3270	[0.8559,0.8574]
wiki.	wiki.	0.8649	0.5125	[0.8641,0.8657]

Table 6.1: Summary statistics for inter-corpus experiments in KSPC (the grey rows indicate control experiments)

6.3.1 Inter-corpus evaluation results

Before looking at the inter-corpus performance, it is worth looking at the performance of the system during the “control” experiments—that is, the performance of the system in perfect conditions. For these experiments, this is defined as the performance of the system which has been trained and tested using the same corpus. By analysing this data, we can determine how successful the system is in the ideal situation of having previously seen the text which the system is being used to type. From the grey cells in Table 6.1 it can be seen that four of the five corpora perform above or at the

performance of a standard keyboard (which has a KSPC of 1.0) during the control experiments. This is confirmation that, even with a limited set of inputs, it is theoretically possible in most cases to enter text using the system with at least the same efficiency as a standard keyboard—and in many cases, even exceed it.

The performance outlined by the aforementioned control experiments also suggest the relative difficulty of each corpus: if a perfectly-trained system exhibits poor performance, then it can be established that this corpus is difficult to type using the system. From Table 6.1, we can determine that the most difficult corpus to type is `newsgroups`, whilst the easiest is `ota-bawe`. Figure 6.1 shows the average KSPC for each pair of training and test corpora (along with the corresponding confidence intervals, the lack of overlap of which demonstrates the statistical significance of these results). By examining this figure, it can be seen that the pseudo-median KSPC for experiments where `newsgroups` was used as the test corpus tend towards the top-right corner of the graph (denoting a high average KSPC, with a mean of 1.2867), while the pseudo-medians for `ota-bawe` as the test corpus tend towards the lower-left corner (denoting a low average KSPC, with a mean of 0.9681). The positions of these pseudo-medians visually confirms the relative difficulty of each corpus; however, the average standard deviation of the `newsgroup` test set is extremely high (3.5092) and therefore this conclusion is possibly of little notability.

In Section 5.2.1, two hypotheses were presented pertaining to the performance of the system with certain domains and writing styles. Hypothesis 1 stated that train/test corpora pairings originating from the same domain will outperform other pairings. Two pairs of corpora share the same domain (as described in Table

Figure 6.1: Pseudo-medians for inter-corpus experiments with confidence intervals

5.2): `enron` and `newsgroup`, which are both of the Communication domain, and `ota-bawe` and `wikipedia`, which are both of the Academic domain. From the data in Table 6.1 and Figure 6.1, it can be seen that the system demonstrates the best performance out of all the non-control experiments when trained on the `wikipedia` corpus and tested on the `ota-bawe` ($KSPC = 0.8566 \pm 0.3270$), and third-best performance in the reverse configuration ($KSPC = 1.0280 \pm 0.4760$). Conversely, both pairings of the `enron` and `newsgroup` corpora demonstrate the second- and third-worst performances ($KSPC = 1.3881 \pm 1.6414$, 1.4160 ± 0.8620); however, as previously mentioned, both of these corpora appear to be difficult corpora to type using the system, as the average KSPC for experiments where these corpora were used as the test corpus is high. Therefore, on the strength of the results from the Academic domain, the hypothesis is weakly accepted.

Conclusion 1 *Corpora pairings from the same domain exhibit improved performance, but not to the same extent that the difficulty of the corpus impacts performance.*

Hypothesis 2 presented the theory that corpora which exhibited the same writing style would outperform those that are written in differing styles. The two styles under consideration are a Formal style (the `enron` and `ota-bawe` corpora) and the Casual style (the `newsgroup` and `fanfiction-small` corpora). However, given the distribution of average KSPCs across all relevant corpora pairings ($KSPC = 1.0999 \pm 0.2995$, 1.3362 ± 1.1269 , 1.3549 ± 8.8267 , 1.2716 ± 0.5121) between all other average KSPCs, it appears that the writing style has little impact on performance. Therefore, the hypothesis is rejected.

Conclusion 2 *Corpora pairings which share the same writing style do not exhibit better performance when compared to other corpora pairings.*

Figure 6.2: Per-phrase relationship between average KSPC and phrase length (alpha level denotes density)

The scatter plots in Figure 6.2 show the relationship between phrase length and average KSPC from each of the inter-corpus experiments. An interesting trend is that all corpora contain a number of phrases which were very short (under 20 characters in length), and yet exhibited a very high average KSPC. This could be caused by a combination of two factors. The first factor is that most (if not all) of the corpora contain several short phrases which are simply one or more acronyms; this is notable as acronyms are context-dependent, and will generally not be contained in more than one corpus. This resulting in a frequent inability to predict acronyms. The second (and potentially related) issue is that if a very short word is not in the system's wordlist, there is a very high penalty associated with attempting to enter the word due to the likely high number of ambiguous matches (as there are many more ambiguous combinations for very short input sequences; see Section 5.1.1 for an explanation of this penalty). While an input sequence of six or seven characters may only match three or four words (and therefore have a very low no-match penalty), input sequences of two or three characters could match upwards of 20 words. Combine this with a very short phrase, and it is understandable that some very short phrases containing unknown words may have an average KSPC in the double digits.

From Figure 6.2, two other interesting pieces of information can be deduced. The first is that there is no symmetry as far as training and testing corpora go. That is, high performance with one train/test pair does not imply high performance reversing the roles of the corpora. This is to be expected, as the difficulty of a corpus is unrelated to its suitability as a training corpus. The second piece of information is that the variance in average KSPC between testing corpora is different for each training corpus. For example, the range of average KSPC where the system has been

trained with `fanfiction-small` is 0.7126, while the range for `newsgroup` is only 0.3087. Furthermore, the `enron` corpus shows a range similar to `newsgroup` of 0.3546, suggesting that perhaps the domain from which the corpus was constructed impacts the variance of average KSPC.

6.3.2 Single corpus evaluation results

Corpus	Pseudo-median	s.d.	CIs
<code>enron</code>	1.0379	1.1012	[1.0363,1.0395]
<code>fanfic.</code>	0.9104	0.8890	[0.9098,0.9110]
<code>news.</code>	1.2179	1.3586	[1.2158,1.2200]
<code>ota-bawe</code>	0.7849	0.1785	[0.7843,0.7855]
<code>wiki.</code>	0.8826	0.6689	[0.8819,0.8833]

Table 6.2: Summary statistics for single corpus experiments in KSPC

By comparing Table 6.2 to Table 6.1, it can be seen that results of these single corpus experiments are very similar to that of the control experiments in Section 6.3.1, with the KSPC values differing by an average of 0.02. This suggests that attempting to write large bodies of text which the system has already been trained on is not easier to type than if the system had not been trained with that specific text (but had been trained with the remainder of the corpus in question). Whilst this observation does not impact the performance of the system, it does suggest that the evaluation methodology involving cross-validation is somewhat unnecessary.

In Section 5.2.2, it was suggested that corpora from smaller, more specific domains are likely to perform better than corpora from larger domains (Hypothesis 3). From this, it was suggested that `fanfiction` will perform better due to the fact that most of the text considers day-to-day situations, whereas `wikipedia`

contains text pertaining to the majority of human knowledge. From the pseudo-median KSPCs listed in Table 6.2, it appears that this behaviour is not observed. In fact, the `ota-bawe` and `wikipedia` corpora demonstrated the best performance out of all the corpora ($KSPC = 0.7849 \pm 0.1785, 0.8826 \pm 0.6689$ respectively). The `enron` corpus was taken from what was likely to be the smallest domain—business emails related to a single corporation—demonstrated the second-worst performance ($KSPC = 1.0379 \pm 1.1012$), albeit with a high standard deviation. Based on this analysis, Hypothesis 3 is rejected out-of-hand.

Conclusion 3 *The size of the domain from which the corpus is constructed has no impact on the performance of the corpus.*

Figure 6.3: Pseudo-medians for single corpus experiments with confidence intervals

In Section 5.2.3 it was proposed that training the system with a corpus which contains samples of the user’s text would allow this user to type with improved performance when compared with a general corpus; this is the hypothesis presented in Hypothesis 4. Table 6.3 shows the results of the experiments performed to determine whether this was indeed the case, with the rows in grey denoting experiments which were cross-validated (Section 5.2.3 explains the experimental process in detail). Figure 6.4 demonstrates that in every case where the training corpus and the testing corpus were constructed from samples of text written by the same user, the performance of the system exceeds cases where different

Train.	Test	P.-median	s.d.	CIs
jd	jd	0.8740	0.8913	[0.8716,0.8764]
jd	km	1.0456	0.8319	[1.0406,1.0505]
jd	mb	0.9496	0.6269	[0.9468,0.9524]
jd	ss	1.0356	0.8460	[1.0302,1.0409]
jd	sk	0.9572	0.8239	[0.9519,0.9624]
jd	tj	1.0184	0.8345	[1.0131,1.0237]
jd	vk	1.0555	1.0070	[1.0501,1.0609]
km	km	0.9227	0.9020	[0.9191,0.9264]
km	jd	1.0735	0.7869	[1.0700,1.0771]
km	mb	1.0909	0.5947	[1.0885,1.0932]
km	ss	1.0159	0.8532	[1.0104,1.0214]
km	sk	1.0392	0.8007	[1.0345,1.0439]
km	tj	0.9976	0.8398	[0.9924,1.0029]
km	vk	1.0795	1.0012	[1.0745,1.0845]
mb	mb	0.8323	0.7249	[0.8307,0.8338]
mb	jd	1.0138	0.8125	[1.0097,1.0179]
mb	km	1.1020	0.8214	[1.0972,1.1068]
mb	ss	1.1008	0.8327	[1.0958,1.1059]
mb	sk	0.9970	0.8145	[0.9923,1.0017]
mb	tj	1.0914	0.8231	[1.0861,1.0966]
mb	vk	1.1047	0.9960	[1.0999,1.1095]
ss	ss	0.8931	0.9662	[0.8893,0.8968]
ss	jd	1.0880	0.7848	[1.0844,1.0916]
ss	km	1.0402	0.8389	[1.0350,1.0454]
ss	mb	1.1062	0.5929	[1.1039,1.1086]
ss	sk	1.0511	0.7995	[1.0465,1.0557]
ss	tj	0.9485	0.8566	[0.9426,0.9545]
ss	vk	1.0791	1.0035	[1.0740,1.0842]
sk	sk	0.8696	0.8718	[0.8666,0.8726]
sk	jd	0.9855	0.8139	[0.9814,0.9895]
sk	km	1.0501	0.8320	[1.0451,1.0552]
sk	mb	0.9663	0.6188	[0.9637,0.9690]
sk	ss	1.0371	0.8463	[1.0319,1.0424]
sk	tj	1.0212	0.8352	[1.0158,1.0267]
sk	vk	1.0479	1.0111	[1.0426,1.0533]
tj	tj	0.8380	0.9493	[0.8338,0.8422]
tj	jd	1.0979	0.7817	[1.0944,1.1013]
tj	km	1.0574	0.8330	[1.0524,1.0624]
tj	mb	1.1139	0.5912	[1.1115,1.1162]
tj	ss	1.0023	0.8598	[0.9970,1.0076]
tj	sk	1.0620	0.7968	[1.0573,1.0666]
tj	vk	1.0942	0.9989	[1.0891,1.0994]
vk	vk	0.9483	1.1309	[0.9447,0.9520]
vk	jd	1.0568	0.7924	[1.0531,1.0606]
vk	km	1.0514	0.8309	[1.0464,1.0564]
vk	mb	1.0587	0.6018	[1.0562,1.0612]
vk	ss	1.0267	0.8479	[1.0213,1.0321]
vk	sk	1.0069	0.8103	[1.0016,1.0121]
vk	tj	1.0130	0.8358	[1.0074,1.0185]

Table 6.3: Summary statistics for individualized corpora experiments in KSPC (the grey rows indicate cross-validated personalized experiments)

users were used for the training and testing corpora. Furthermore, by comparing the results of these experiments to the results displayed in Table 6.2, it can be seen that the average KSPC for personalized corpora is considerably lower than the average KSPC for the single corpus evaluation of the `enron` corpus. From this we can conclude that using a personalized corpus allows a user to enter text at much higher rates than if the system was trained with a general corpus.

Conclusion 4 *A personalized corpus allows a user to enter text at considerably higher rates when compared with a general corpus.*

6.3.4 Keyboard split evaluation results

Split	Pseudo-median	s.d.	CIs
ABCDEF ^G	1.2842	1.4880	[1.2838,1.2847]
AEIOUY	1.5561	2.0664	[1.5555,1.5567]
CEGLSUZ	1.2244	1.2971	[1.2240,1.2248]
LOCVGDQ	1.2024	1.1934	[1.2020,1.2028]
NWZJEBH	1.2090	1.2283	[1.2085,1.2094]
QWERTYU	1.3409	1.5452	[1.3404,1.3414]
SNWZXOF	1.2328	1.2817	[1.2324,1.2332]
URIQPDF	1.2016	1.2073	[1.2012,1.2020]
WPFERVY	1.2062	1.1993	[1.2058,1.2065]

Table 6.4: Summary statistics for keyboard split experiments in KSPC

The data presented in Table 6.4 demonstrates the impact of the keyboard split on the performance of the ambiguous keyboard and, by association, the entire system. Section 5.2.4 detailed two hypotheses related to the keyboard split. The first hypothesis stated that the keyboard split would have a significant impact on the rate at which a user is able to input text into the system. Given the range between the best- and worst-performing split

(0.3545), it is clear that this hypothesis should be accepted. However, one caveat of these results is that range of all the candidate splits (the process of the extraction of which is described in Section 5.2.4) is much lower than the range between the best- and worst-performing splits (only 0.03121). This implies that there is little room for improvement over the existing splits. Despite this, there is still a clear correlation between system performance and keyboard split.

Conclusion 5 *The performance of the system is heavily dependent on the way in which the ambiguous keyboard is split.*

Figure 6.5: Pseudo-medians for keyboard splits experiments with confidence intervals

As mentioned, the performance of the set of candidate splits is somewhat fixed, and the range between the best- and worst-

performing splits is very small. However, the impact of the keyboard split is a lot more obvious when looking at the ordered splits. Looking at Figure 6.5, it is clear that the ordered keyboard splits all perform with considerably worse performance. The implication of this is that uses of a system which does not require the characters of the keyboard split to appear in some sort of natural order (such as a BCI system) will benefit from one of the candidate splits, and has the potential to improve on a system which cannot use these splits for reasons of cognitive load. This result means that we can accept Hypothesis 6 (Figure 6.5 shows that this conclusion is statistically significant).

Conclusion 6 *An unordered split is able to achieve higher performance when compared to an ordered split.*

6.3.5 Subsystem evaluation results

Subsystems	P-median	s.d.	CI
default	1.2315	3.0398	[1.2310,1.2319]
no-ambiguous	0.9116	0.3185	[0.9114,0.9119]
no-autocomplete	1.2379	1.2261	[1.2375,1.2382]
no-prediction	1.3778	1.2444	[1.3775,1.3782]
only-ambiguous	1.3078	1.2065	[1.3075,1.3081]
only-autocomplete	1.1237	0.2341	[1.1235,1.1239]
only-prediction	0.9942	0.2787	[0.9940,0.9944]

Table 6.5: Summary statistics for subsystem experiments in KSPC

The assistive input method is made up of three distinct subsystems: the ambiguous keyboard, the sentence-level word prediction system and the word auto-completion system. In Section 5.2.5, the purpose of each of these three systems was discussed, and it was pointed out that they are not all necessarily working towards a common goal—specifically, the ambiguous keyboard

acts to adapt the system to limited input methods, which can impede the efficiency of the system. The results presented in Table 6.5 demonstrate the performance impact that each subsystem has on the overall performance of the system by showing every combination of subsystems. These combinations include all three subsystems enabled (default), one subsystem disabled (e.g. no-ambiguous) and two subsystems disabled (e.g. only-auto-complete).

Figure 6.6: Pseudo-medians for subsystem experiments with confidence intervals

As previously mentioned, the ambiguous keyboard subsystem has the potential to make it slower to enter text using the assistive input system (but allows users to input text with fewer physical inputs). Hypothesis 7 formalizes this by stating that the ambiguous keyboard will have a negative impact on the performance of the system. Of the seven tested configurations, three have the am-

ambiguous keyboard disabled. These are `no-ambiguous`, `only-autocomplete` and `only-prediction`. As expected, Figure 6.6 shows that these three configurations all strongly outperform all other configurations when the KSPC is averaged across all corpora. The `no-ambiguous` configuration, which keeps sentence-level word prediction and word auto-completion enabled, leverage the benefits of these by providing the best performance; by comparison, only enabling the word auto-completion system is slower for entering text than a standard keyboard. This behaviour is to be expected, as the auto-completion system relies on the word prediction system to provide suggestions to present as an auto-completion—with this system disabled, the suggestions will generally be somewhat random and will serve only to require an additional backspace in many situations.

Conclusion 7 *The performance of the system is very negatively impacted by the ambiguous keyboard.*

The sentence-level word prediction engine is one of the two subsystems designed to increase the rate at which a user can type. Hypothesis 8 states that the performance of the system will be greatly improved through the presence of the word prediction subsystem. To determine the effect that the word prediction engine has on the system, we will do a pairwise comparison of the configurations. In the case where both the other subsystems are enabled (`default` and `no-prediction`), the system performs significantly better with the word prediction subsystem enabled ($KSPC = 1.2315$ and 1.3778 respectively). Comparing the effect of the word prediction engine in cases where only auto-complete is enabled (`no-ambiguous` and `only-autocomplete`) and where only the ambiguous keyboard is enabled (`no-autocomplete` and `only-ambiguous`), the configuration where the subsystem is en-

abled performs significantly better than in cases where the subsystem is disabled. Finally, the `only-prediction` configuration performs better than most other configurations, and outperforms a standard keyboard also. Based on these findings, the hypothesis is accepted.

Conclusion 8 *The performance of the system is very positively impacted by sentence-level word prediction.*

The third and final subsystem is the word auto-completion engine, which is another system designed to increase text entry rates. Hypothesis 9 states that, like the sentence-level word prediction subsystem, the auto-completion system will have a great positive impact on the performance of the system. By performing another pairwise comparison of the configurations, we can determine the impact that this subsystem has on the overall performance of the system. Where both the other subsystems are enabled (`default` and `no-autocomplete`), there is very little difference—disabling the word auto-completion subsystem provides a KSPC of 1.2379 compared to the 1.2315 where the subsystem is enabled. Furthermore, this result is not statistically significant. When only the ambiguous keyboard is enabled (`no-prediction` and `only-ambiguous`), having the auto-completion system enabled is actually detrimental—this gives further strength to the argument that the auto-completion system will complete more-or-less random suggestions when the sentence-level prediction engine is disabled. However, when the ambiguous keyboard is disabled and only the word prediction engine is enabled (`no-ambiguous` and `only-prediction`), the presence of the auto-completion system brings notable improvements ($KSPC = 0.9116$ and 0.9942 respectively). Therefore, the hypothesis is conditionally accepted.

Conclusion 9 *The performance of the system is positively impacted by word auto-completion, but only when paired with sentence-level word prediction; otherwise, the performance is negatively impacted.*

6.3.6 Tuning parameters evaluation results

Behaviours	P-median	s.d.	CI
bad-drop-all	1.2315	3.0398	[1.2311,1.2318]
bad-drop-children	1.2272	1.2823	[1.2268,1.2276]
bad-drop-start	1.2272	1.2823	[1.2268,1.2275]
bad-drop-self	1.2308	1.2819	[1.2304,1.2312]

Table 6.6: Summary statistics for bad word behaviours in KSPC

The four bad word behaviours (described in Section 3.3.2) were evaluated in isolation in order to determine whether changing this behaviour would result in a notable performance difference. However, as can be seen from the data in Table 6.6, the enabled behaviour tended to have no impact; the differences in average KSPC were statistically insignificant. This is likely due to the fact that there are comparatively very few “bad words” in the corpora, and varying the behaviour of how the phrases containing these words are treated is relatively insignificant.

Window sizes	P-median	s.d.	CI
size-2	1.2366	1.2799	[1.2364,1.2369]
size-3	1.2329	1.2817	[1.2325,1.2332]
size-4	1.2349	1.2814	[1.2345,1.2353]
size-2-variable	1.2366	1.2799	[1.2362,1.2370]
size-3-variable	1.2322	1.2817	[1.2318,1.2326]
size-4-variable	1.2324	1.2818	[1.2321,1.2328]

Table 6.7: Summary statistics for phrase window sizes in KSPC

Table 6.7 details the impact of varying the phrase window size, a configuration flag which is described in detail in Section 3.3.1. It was expected that increasing the size of the moving window used

to extract phrases from the corpora would have a positive impact, as it would allow for more context to be captured in the phraselist. However, the results demonstrate that there is no statistical significance in varying this number; the same is also true of the variable phrase window size. Due to these results, it is therefore preferable for the system to use the smaller phrase window size, especially considering that this will result in considerably lower memory consumption, allowing for much larger corpora to be used. Given that the corpus used to train the system has a considerable effect on its performance (as shown in Section 6.3.1), this is a significant benefit.

Thresholds	P-median	s.d.	CIs
phrase-freq-off	1.2202	1.2815	[1.2198,1.2206]
phrase-freq-10	1.2428	1.2792	[1.2424,1.2432]
phrase-freq-20	1.2529	1.2766	[1.2526,1.2533]
phrase-freq-30	1.2591	1.2749	[1.2587,1.2595]
phrase-freq-50	1.2674	1.2726	[1.2671,1.2678]

Table 6.8: Summary statistics for minimum phrase frequency thresholds in KSPC

Thresholds	P-median	s.d.	CIs
word-freq-5	1.2850	0.8670	[1.2198,1.2206]
word-freq-10	1.3246	0.7498	[1.2424,1.2432]
word-freq-20	1.3727	0.6590	[1.2526,1.2533]
word-freq-30	1.4021	0.6115	[1.2587,1.2595]
word-freq-50	1.4430	0.5617	[1.2671,1.2678]

Table 6.9: Summary statistics for minimum word frequency thresholds in KSPC

Section 3.3.2 describes both the minimum phrase frequency and minimum word frequency thresholds. The minimum phrase frequency threshold determines the frequency at which a phrase must appear within a corpus for it to be included in the phraselist of the system, while the minimum word frequency threshold

determines the minimum number of times a word must appear for it to be counted as a “good” word. Tables 6.8 and 6.9 present the performance of the system with these two options varied respectively. While it was thought that these two thresholds could be increased in order to remove words and phrases which were specific to the contexts and domains in which they were used, the data from these tables demonstrate that removing phrases and words according to these thresholds had a uniformly negative effect, with the average KSPC increasing in all cases. From this it can be determined that there is no cause for removing such words and phrases, as the memory consumption is not drastically reduced through their removal.

From the above analysis, Hypothesis 10 can be given the following conclusion:

Conclusion 10 *The tuning parameters have a moderate impact on the performance of the system, but to a lesser extent than the training and testing corpora.*

6.4 Chapter summary

In this chapter, the results of the experiments from Chapter 5 were presented. From these results, it was possible to draw conclusions regarding the hypotheses set forth in the same chapter. This chapter also detailed the process taken to analyse the results. Chapter 7 concludes the thesis with the findings and how these relate to the goals of the research, whilst also providing a suggestion for future work to be undertaken.

Chapter 7

Conclusions

Assistive input methods are used by many users to enter text in situations where it is not possible to use a full keyboard, whether this is due to a user's physical disability, the usage scenario or limitations of the hardware. These assistive methods have existed for several decades, but have traditionally demonstrated low speeds for the input text, thereby limiting their usefulness—this is especially true in fields such as brain computer interaction, where anything over seven characters per minute is considered to be a good input rate. The primary aim of this thesis was to develop a system for efficiently entering text using a limited hardware interface, whilst also investigating the effects that various tuning parameters have on its performance. These goals were achieved through the design and implementation of a system, as well as a corresponding evaluation framework which was used to successfully evaluate the system with a variety of configurations and corpora.

This chapter will discuss the conclusions presented in Chapter 6, and revisit the goals of the research based on these findings. Following this, possibilities for future work based on the results of this research is presented. Finally, the thesis is concluded with a brief summary and some final thoughts.

7.1 Contributions

The contributions of this thesis are threefold. The first contribution is the assistive input method which combines three assistive technologies—an ambiguous keyboard, a word prediction system and a word auto-completion system—which provides performance comparable to that of a standard full-sized keyboard. The second contribution is an evaluation framework, which allowed for measuring the performance as well as—the third contribution—the impact of various tuning and configuration parameters. Section 1.3 listed the four major goals of this research, and these goals will be revisited here in the context of the results presented in Chapter 6.

The first of these goals was to attempt to design an assistive input method which allows users to enter text at a comparable rate to a standard keyboard, despite having a severely limited number of inputs. While many situations in which an ambiguous keyboard is used have more inputs available (for example, mobile phones with nine alphanumeric keys), the ambiguous keyboard designed for this research was restricted to having four inputs. This number was chosen to reflect a domain which is even more restrictive in the number of unique inputs, such as brain-computer interaction. Despite this restriction, the system presented in this thesis was often able to give similar performance to a standard keyboard—especially in favourable (but, as far as real-world applications go, realistic) configurations, such as when the system has been trained with a similar corpus to the target domain of the system, or when it has been trained using a corpus personalized for the user.

The second goal of this thesis was to determine the effect that various tuning parameters have on the performance of the sys-

tem. From the results presented in Section 6.3.6, it was concluded in Conclusion 10 that although the tuning parameters do affect the performance of the system, it is the corpus used to train the system which has a much larger performance impact. The parameter which had the most impact on performance other than the training corpus was the way in which the keyboard was split. The performance of the system was found to be much improved when the restriction on keyboard order was lifted. As a result of this, hardware interfaces for this system which do not require the user to explicitly select an input class for the ambiguous keyboard—for example, a brain-computer interface which presents the keyboard in full as a mask for the underlying ambiguous interface (as per Section 1.2.1)—will benefit greatly from this finding.

The third goal of the research was to determine the effect of the corpus size, domain and writing style on the performance of the system. Sections 6.3.1 through 6.3.3—along with Conclusions 1 through 4—presented evidence that the performance of the system is almost entirely dependent on the corpus used to train the system. From these results, it can be deduced that the best way to optimize the assistive input system is to train it using a corpus which is specific to the user's needs, whether this is a sample of his or her own writing, or whether it is simply text from the same domain (writing style was not found to impact performance). Because the system is able to load a corpus in a reasonably short amount of time, it is not an unreasonable suggestion to load a new, specific corpus for each new typing task that the user needs to perform.

The fourth and final goal of this thesis was to determine the relationships between the corpus size used to train the system and both its memory usage and its response time. It was concluded that both of these relationships were linear in nature, although

their impacts were very different. On the one hand, the memory requirements increase greatly with the corpus size, with a 50 MB corpus using more than 4 GB extra memory over a 10 MB corpus. Conversely, the response time of the system was well under 1ms for each of the tested corpora. For this reason, the response time is not likely to be a concern for any corpus able to fit into a reasonable amount of memory—at least by today’s standards.

7.2 Future work

One of the contributions of this research was to determine the impact that the size of the corpus used to train the system has on its response time and memory usage. An important related question is what the relationship is between the corpus size and the performance of the system. By answering this question, it would be possible to determine the optimal trade-off between corpus size and system performance for devices with much tighter memory and storage restrictions; therefore, this is an important direction for future research to take.

This research has determined that a large performance increase can be obtained by using an ambiguous keyboard with an unordered keyboard split. However, removing this restriction is impossible in many circumstances, as removing this order would increase the cognitive load required to use the system, thereby negating any performance improvement gained from using an unordered split. For this reason, assistive input methods such as the one created in this thesis would benefit from research into methods of removing this limitation.

7.3 Final thoughts

Assistive typing techniques have been around for many decades, and exist both to improve communication for people with physical disabilities, and to improve communication ability for people in situations where it is otherwise difficult or impossible to communicate (due to hardware or social constraints). This thesis has looked at existing text input methods for assistive and augmentative communication (AAC), and created a novel system which combines several of these techniques—including ambiguous keyboards, word prediction and word auto-completion—to rival the efficiency of a standard keyboard. To achieve this, the thesis looked at the various configurations and tuning parameters for such a system and determined that such an input method can be tuned to a user's needs to achieve better performance. By investigating the relationship between the system configuration and the running requirements, this thesis has also determined how best to configure the system for limited hardware configurations. By continuing work in this area, it will be possible to further optimize an assistive input system to better suit the needs of users who need it most.

Bibliography

- [1] ARNOTT, J., AND JAVED, M. Probabilistic character disambiguation for reduced keyboards using small text samples. *Augmentative and Alternative Communication* 8, 3 (1992), 215–223.
- [2] BAILEY, B. P., KONSTAN, J. A., AND CARLIS, J. V. The effects of interruptions on task performance, annoyance, and anxiety in the user interface. In *Proceedings of INTERACT* (2001), vol. 1, pp. 593–601.
- [3] BENTRUP, J. A. Exploiting word frequencies and their sequential dependencies. In *Proceedings, 10th Annual Conf. on Rehabilitation Technology* (1987), pp. 121–123.
- [4] BEUKELMAN, D., AND MIRENDA, P. Augmentative and alternative communication: Supporting children and adults with complex communication needs.
- [5] BIBER, D. *Variation across speech and writing*. Cambridge University Press, 1991.
- [6] BICKEL, S., HAIDER, P., AND SCHEFFER, T. Learning to complete sentences. In *Machine Learning: ECML 2005*. Springer, 2005, pp. 497–504.

- [7] BLANKERTZ, B., DORNHEGE, G., KRAULEDAT, M., SCHRÖDER, M., WILLIAMSON, J., MURRAY-SMITH, R., AND MÜLLER, K.-R. The Berlin Brain-Computer Interface presents the novel mental typewriter Hex-o-Spell.
- [8] BRYEN, D. N., AND JOYCE, D. G. Language intervention with the severely handicapped: A decade of research. *The Journal of Special Education* 19, 1 (1985), 7–39.
- [9] BUHR, P., AND HOLTE, R. Some considerations in the design of communication aids for the severely physically disabled. *Medical and Biological Engineering and Computing* 19, 6 (1981), 725–733.
- [10] CARD, S. K., ROBERTSON, G. G., AND MACKINLAY, J. D. The information visualizer, an information workspace. In *Proceedings of the SIGCHI conference on human factors in computing systems* (1991), ACM, pp. 181–186.
- [11] CARLBERGER, A., CARLBERGER, J., MAGNUSON, T., HUNNICUTT, M. S., PALAZUELOS-CAGIGAS, S. E., AND NAVARRO, S. A. Profet, a new generation of word prediction: An evaluation study. In *Proceedings, ACL Workshop on Natural language processing for communication aids* (1997), pp. 23–28.
- [12] CECOTTI, H. Spelling with Brain-Computer Interfaces: Current trends and prospects. In *In Proc. of the 5th French Conference on Computational Neuroscience (Neurocomp 2010)* (2010), pp. 215–220.
- [13] COHEN, W. W. Enron Email Dataset. <https://www.cs.cmu.edu/~enron/>, Aug. 2009. Accessed 2013-10-09.

- [14] COVER, T. M., AND KING, R. A convergent gambling estimate of the entropy of English. *Information Theory, IEEE Transactions on* 24, 4 (1978), 413–421.
- [15] DAMERAU, F. J. A technique for computer detection and correction of spelling errors. *Communications of the ACM* 7, 3 (1964), 171–176.
- [16] DARRAGH, J. J., AND WITTEN, I. H. Adaptive predictive text generation and the reactive keyboard. *Interacting with Computers* 3, 1 (1991), 27–50.
- [17] DONCHIN, E., SPENCER, K. M., AND WIJESINGHE, R. The mental prosthesis: assessing the speed of a P300-based brain-computer interface. *Rehabilitation Engineering, IEEE Transactions on* 8, 2 (2000), 174–179.
- [18] DOWLE, M., SHORT, T., SRINIVASAN, S. L. W. C. F. A., AND SAPORTA, R. *data.table: Extension of data.frame for fast indexing, fast ordered joins, fast assignment, fast grouping and list columns.*, 2013. R package version 1.8.10.
- [19] DUNLOP, M., AND CROSSAN, A. Dictionary based text entry method for mobile phones. In *Second Workshop on Human-Computer Interaction with Mobile Devices* (1999), pp. 5–7.
- [20] DUNLOP, M. D., AND CROSSAN, A. Predictive text entry methods for mobile phones. *Personal Technologies* 4, 2-3 (2000), 134–143.
- [21] DUNN, O. J. Confidence intervals for the means of dependent, normally distributed variables. *Journal of the American Statistical Association* 54, 287 (1959), 613–621.

- [22] ENCYCLOPEDIA BRITANNICA. Britannica Global Edition - The Britannica Store. <http://store.britannica.com/collections/books/products/043009100>, 2013. Accessed 2014-01-23.
- [23] FARWELL, L. A., AND DONCHIN, E. Talking off the top of your head: toward a mental prosthesis utilizing event-related brain potentials. *Electroencephalography and clinical Neurophysiology* 70, 6 (1988), 510–523.
- [24] FAZLY, A., AND HIRST, G. Testing the efficacy of part-of-speech information in word completion. In *Proceedings of the 2003 EACL Workshop on Language Modeling for Text Entry Methods* (2003), Association for Computational Linguistics, pp. 9–16.
- [25] FOULDS, R. A., BALETSA, G., AND CROCHETIERE, W. The effectiveness of language redundancy in non-verbal communication. In *Proceedings of the Conference on Devices and Systems for the Disabled* (1975), pp. 82–86.
- [26] GARDNER, S., AND POWELL, L. An investigation of genres of assessed writing in British higher education: a Warwick-Reading-Oxford Brookes project. In *Research, scholarship and practice in the area of Academic Literacies, University of Westminster* (2006), vol. 30.
- [27] GLASER, R. E. A telephone communication aid for the deaf. *Proceedings of the Johns Hopkins First National Search for Applications of Personal Computers to Aid the Handicapped* (1981).
- [28] GLENNEN, S., AND DECOSTE, D. C. *The handbook of augmentative and alternative communication*. Cengage Learning, 1997.

- [29] GONG, J., AND TARASEWICH, P. Alphabetically constrained keypad designs for text entry on mobile devices. In *Proceedings of the SIGCHI conference on Human factors in computing systems* (2005), ACM, pp. 211–220.
- [30] GROVER, D., KING, M., AND KUSHLER, C. Reduced keyboard disambiguating computer, Oct. 1998. US Patent 5,818,437.
- [31] HARBUSCH, K., AND KÜHN, M. Towards an adaptive communication aid with text input from ambiguous keyboards. In *Proceedings of the tenth conference on European chapter of the Association for Computational Linguistics-Volume 2* (2003), Association for Computational Linguistics, pp. 207–210.
- [32] HECKATHORNE, C. W., AND CHILDRESS, D. S. Applying anticipatory text selection in a writing aid for people with severe motor impairment. *IEEE Micro* 3, 3 (1983), 17–23.
- [33] HODGES JR, J. L., AND LEHMANN, E. L. Estimates of location based on rank tests. *The Annals of Mathematical Statistics* (1963), 598–611.
- [34] HOW, Y., AND KAN, M.-Y. Optimizing predictive text entry for short message service on mobile phones. In *Proceedings of HCI* (2005), vol. 5.
- [35] INVERSO, S., HAWES, N., KELLEHER, J., ALLEN, R., AND HAASE, K. Think and Spell: Context-Sensitive Predictive Text for an Ambiguous Keyboard Brain-Computer Interface Speller. *Biomedizinische Technik* 49 (2004), 53–4.
- [36] JACKSON, A., MORITZ, C. T., MAVOORI, J., LUCAS, T. H., AND FETZ, E. E. The Neurochip BCI: towards a neural pros-

- thesis for upper limb function. *Neural Systems and Rehabilitation Engineering, IEEE Transactions on* 14, 2 (2006), 187–190.
- [37] JAMES, C., AND LONGÉ, M. Bringing text input beyond the desktop. In *CHI'00 extended abstracts on Human factors in computing systems* (2000), ACM, pp. 49–50.
- [38] JOHNSON, A. B., AND HAGSTAD, R. F. DTMF telecommunications for the deaf and speech impaired,". *Proceedings of the Johns Hopkins First National Search for Applications of Personal Computers to Aid the Handicapped* (1981).
- [39] JUDGE, S., AND FRIDAY, M. Ambiguous keyboards for AAC. *Journal of Assistive Technologies* 5, 4 (2011), 249–256.
- [40] KOESTER, H. H., AND LEVINE, S. Effect of a word prediction feature on user performance. *Augmentative and alternative communication* 12, 3 (1996), 155–168.
- [41] KOHAVI, R., ET AL. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *IJCAI* (1995), vol. 14, pp. 1137–1145.
- [42] KUSHLER, C. AAC using a reduced keyboard.
- [43] LANG, K. Newsweeder: Learning to filter netnews. In *Proceedings of the Twelfth International Conference on Machine Learning* (1995), Citeseer.
- [44] LEEB, R., FRIEDMAN, D., MÜLLER-PUTZ, G. R., SCHERER, R., SLATER, M., AND PFURTSCHELLER, G. Self-paced (asynchronous) BCI control of a wheelchair in virtual environments: a case study with a tetraplegic. *Computational intelligence and neuroscience 2007* (2007).

- [45] LESHER, G. W., MOULTON, B. J., AND HIGGINBOTHAM, D. J. Optimal character arrangements for ambiguous keyboards. *Rehabilitation Engineering, IEEE Transactions on* 6, 4 (1998), 415–423.
- [46] LESHER, G. W., MOULTON, B. J., HIGGINBOTHAM, D. J., ET AL. Effects of ngram order and training text size on word prediction. In *Proceedings of the RESNA'99 Annual Conference* (1999), pp. 52–54.
- [47] LEVINE, S. H., TREPAGNIER, G. C., GETSCHOW, C., AND MINNEMAN, S. Multi-character key text entry using computer disambiguation. In *RESNA 10th Annual Conference, San Jose, California* (1987), pp. 177–178.
- [48] LI, X. FanFiction. <http://www.fanfiction.net>, Oct. 2013. Accessed 2013-10-18.
- [49] LU, S., GUAN, C., AND ZHANG, H. Unsupervised brain computer interface based on intersubject information and on-line adaptation. *Neural Systems and Rehabilitation Engineering, IEEE Transactions on* 17, 2 (2009), 135–145.
- [50] MACKENZIE, I. S., KOBER, H., SMITH, D., JONES, T., AND SKEPNER, E. LetterWise: prefix-based disambiguation for mobile text input. In *Proceedings of the 14th annual ACM symposium on User interface software and technology* (2001), ACM, pp. 111–120.
- [51] MATIASSEK, J., BARONI, M., AND TROST, H. FASTY: A multilingual approach to text prediction. In *Computers Helping People with Special Needs*. Springer, 2002, pp. 243–250.
- [52] MENG, L., JIN, J., AND WANG, X. A comparison of navigation system based on P300 BCI and SSVEP BCI. In *Con-*

- trol and Decision Conference (CCDC), 2012 24th Chinese (2012), IEEE, pp. 3703–3708.*
- [53] MILLER, R. B. Response time in man-computer conversational transactions. In *Proceedings of the December 9-11, 1968, fall joint computer conference, part I* (1968), ACM, pp. 267–277.
- [54] MILNE, D., AND WITTEN, I. H. An open-source toolkit for mining Wikipedia. *Artificial Intelligence* (2012).
- [55] MORASSO, P., PENSO, M., SUETTA, G., AND TAGLIASCO, V. Towards standardisation of communication and control systems for motor impaired people. *Medical and Biological Engineering and Computing* 17, 4 (1979), 481–488.
- [56] NANDI, A., AND JAGADISH, H. Effective phrase prediction. In *Proceedings of the 33rd international conference on Very large data bases* (2007), VLDB Endowment, pp. 219–230.
- [57] NIJBOER, F., SELLERS, E., MELLINGER, J., JORDAN, M., MATUZ, T., FURDEA, A., HALDER, S., MOCHTY, U., KRUSIENSKI, D., VAUGHAN, T., ET AL. A P300-based brain-computer interface for people with amyotrophic lateral sclerosis. *Clinical neurophysiology* 119, 8 (2008), 1909–1916.
- [58] PIRES, G., CASTELO-BRANCO, M., AND NUNES, U. Visual P300-based BCI to steer a wheelchair: A Bayesian approach. In *Engineering in Medicine and Biology Society, 2008. EMBS 2008. 30th Annual International Conference of the IEEE* (2008), IEEE, pp. 658–661.
- [59] PRITCHARD, W. S. Psychophysiology of P300. *Psychological bulletin* 89, 3 (1981), 506.

- [60] R CORE TEAM. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2013.
- [61] REICHLE, J. Communication intervention with persons who have severe disabilities. *The Journal of Special Education* 31, 1 (1997), 110–134.
- [62] RENNIE, J. Home Page for 20 Newsgroups Data Set. <http://qwone.com/~jason/20Newsgroups/>, Jan. 2008. Accessed 2013-10-29.
- [63] RING, N. Communication aids for the speech impaired. *The use of technology in the care of the elderly and the disabled*. Francis Pinter, London (1980), 79–82.
- [64] ROSEN, M. J., AND GOODENOUGH-TREPAGNIER, C. Communication Systems for the Non-Vocal Motor Handicapped: Practice & Prospects. *Engineering in Medicine and Biology Magazine, IEEE* 1, 4 (1982), 31–35.
- [65] SARKAR, D. *Lattice: Multivariate Data Visualization with R*. Springer, New York, 2008. ISBN 978-0-387-75968-5.
- [66] SCHWARTZ, A. B., CUI, X. T., WEBER, D. J., AND MORAN, D. W. Brain-controlled interfaces: movement restoration with neural prosthetics. *Neuron* 52, 1 (2006), 205–220.
- [67] SHANNON, C. E. Prediction and entropy of printed English. *Bell system technical journal* 30, 1 (1951), 50–64.
- [68] SUEN, C. Y. N-gram statistics for natural language understanding and text processing. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 2 (1979), 164–172.

- [69] SWIFFIN, A. L., PICKERING, J. A., ARNOTT, J. L., AND NEWELL, A. F. PAL: An effort efficient portable communication aid and keyboard emulator. In *Proceedings of the 8th Annual Conference on Rehabilitation Technology* (1985), pp. 197–199.
- [70] TANAKA-ISHII, K. Word-based predictive text entry using adaptive language models. *Natural Language Engineering* 13, 1 (2007), 51–74.
- [71] TANAKA-ISHII, K., INUTSUKA, Y., AND TAKEICHI, M. Entering text with a four-button device. In *Proceedings of the 19th international conference on Computational linguistics-Volume 1* (2002), Association for Computational Linguistics, pp. 1–7.
- [72] THOMAS, A. Communication devices for the nonvocal disabled. *Computer* 14, 1 (1981), 25–30.
- [73] UNIVERSITY OF OXFORD. [OTA] British Academic Written Corpus. <http://ota.ahds.ac.uk/desc/2539>, May 2013. Accessed 2013-05-25.
- [74] VAN DEN BOSCH, A. Effects of context and recency in scaled word completion. *Computational Linguistics in the Netherlands Journal* 1 (2011), 79–94.
- [75] VOLOSYAK, I., CECOTTI, H., VALBUENA, D., AND GRASER, A. Evaluation of the Bremen SSVEP based BCI in real world conditions. In *Rehabilitation Robotics, 2009. ICORR 2009. IEEE International Conference on* (2009), IEEE, pp. 322–331.
- [76] WICKHAM, H. *ggplot2: elegant graphics for data analysis*. Springer New York, 2009.
- [77] WIKIMEDIA FOUNDATION. Wikipedia. <http://www.wikipedia.org/>, Nov. 2013. Accessed 2014-01-23.

- [78] WITTEN, I. H. An interactive computer terminal interface which predicts user entries.
- [79] WOOD, M. E. J. *Syntactic pre-processing in single-word prediction for disabled people*. PhD thesis, Citeseer, 1996.
- [80] YIN, P.-Y., AND SU, E.-P. Optimal character arrangement for ambiguous keyboards using a PSO-based algorithm. In *Natural Computation (ICNC), 2011 Seventh International Conference on* (2011), vol. 4, IEEE, pp. 2194–2198.