DEVELOPMENT OF TEXT ENTRY MECHANISMS FOR INDIC LANGUAGES

Sayan Sarcar

DEVELOPMENT OF TEXT ENTRY MECHANISMS FOR INDIC LANGUAGES

Thesis submitted to the
Indian Institute of Technology Kharagpur
for award of the degree

of

Doctor of Philosophy (Ph.D.)

by

Sayan Sarcar

Under the guidance of

Dr. Debasis Samanta

School of Information Technology
Indian Institute of Technology Kharagpur
Kharagpur - 721 302, India
January 2015

CERTIFICATE

This is to certify that the thesis entitled **Development of Text Entry Mechanisms for Indic Languages**, submitted by **Sayan Sarcar** to Indian Institute of Technology, Kharagpur, is a record of bona fide research work under my supervision and we consider it worthy of consideration for the award of the degree of Doctor of Philosophy of the Institute.

Date: 28/01/2015

Dr. Debasis Samanta
Associate Professor
School of Information Technology
Indian Institute of Technology Kharagpur
Kharagpur - 721302, India

DECLARATION

I certify that

a. The work contained in the thesis is original and has been done by myself under the general supervision of my supervisor.

b. The work has not been submitted to any other Institute for any degree or diploma.

c. I have followed the guidelines provided by the Institute in writing the thesis.

d. I have conformed to the norms and guidelines given in the Ethical Code of Conduct of the Institute.

e. Whenever I have used materials (data, theoretical analysis, and text) from other sources, I have given due credit to them by citing them in the text of the thesis and giving their details in the references.

f. Whenever I have quoted written materials from other sources, I have put them under quotation marks and given due credit to the sources by citing them and giving required details in the references.

Sayan Sarcar

BIOGRAPHY

Sayan Sarcar is currently a Ph.D. research scholar in the School of Information Technology at Indian Institute of Technology Kharagpur, India. In his research, he is focusing on development of text entry methods in Indian languages which support mouse, touch and gaze based interaction in desktop and mobile devices. He has received the B.Tech. degree in Computer Science & Engineering from West Bengal University of Technology, India in 2005 and the M.Tech degree in Multimedia & Software Systems from the same university in 2007.

His current research interests include human-computer interaction, user interface design, usability testing, mobile application and machine learning. He is a student member of ACM and IEEE.

Dedicated to

My beloved Mother

ACKNOWLEDGMENT

Since the day I first set out on the journey of my doctoral study, to this day when I am handing my thesis over, there have been innumerable people who have helped me at each and every step. No words in the English language and beyond can do justice in expressing my gratitude for them, but I would make a frail attempt, nonetheless. This journey has been no less than a roller coaster ride. I have had my share of ups, followed by many slip downs, and so on in a similar fashion. These ups were enjoyable because I had people to multiply my happiness by sharing it, and these very people provided for the preventive cushion while I hit the all time lows.

At the very outset, I would like to thank my supervisor, *Dr. Debasis Samanta*, for his perpetual support and guidance. He has been a constant companion in all the difficulties I faced, battling them like his own. He motivated me when I was feeling low, and steered my efforts in the right direction. But for his guidance, I would have been lost right from the very onset. He is a person who went beyond professional walls, and his genuine concern for my personal life has always been heartwarming. Had it not been for him, this day couldn't have been realized. It gives me immense pleasure to thank my doctoral scrutiny committee members Prof. Anupam Basu, Prof. Sudeshna Sarkar, Prof. Goutam Saha, Dr. S. Misra for their valuable suggestions during my research tenure. My sincere thanks to the heads of the department Prof. I. Sengupta, Prof. J. Mukhopadhyay and Prof. R. Mall for the world class infrastructure provided in the department to the research students. I also thank all the faculty members of the School of Information Technology for their many helpful comments and constant encouragement. I owe my deepest gratitude to Dr. Monalisa Sarma for her continuous support and encouragement during my doctoral study. I sincerely remember the support of office staffs Mithunda, Somadi, Malayda, Vinodda, Pratap, Das babu and others.

Some people play such a part in our lives that we feel like they have been God sent, just to fuel our lives forward. These were the people who patiently stood by me when I was difficult, criticized me to hone me, infused me with inspiration when I saw no hope, and were an inifinite store of never ending encouragement. They were the ones who came to my aid even before I called out for help, and expected nothing in return. I would like to thank Manoj, Pradipta, Tuhin and Sankar for being there for me no matter what.

I would like to acknowledge the financial support of Department of Information Technology, New Delhi, India towards my doctoral study. I am indebted to Prateek, Tuhin and Sourov for their pricelss help in the development of some algorithms, contributing to my research.

There have been people who have provided for the support I could fall back on anytime. I would like to thank such support in the form of my old friends and seniors Sujatadi, Debasishda, Ranjanda, Somnathda, Ranjanda, Soumalya, Soumya, Sudhamay, Arkadeep, Chandanda, Barikda, Pushpitadi, Dibyendu, Suman, Tuhin, Anjan, Uttam, Tapan, Jayeeta, Indira and many more. I would also like to thank my SIT friends Nirnay, Manas, Shrutilipi, Barsha, Goutamda,Sai Praveen, Pushpendu, Nabiul, Arindamda, Chandan, Manasa, Ayan, Arijit, Dipanjan, Barun, Pratik, Gaurang, Tamoghna, Samaresh, Arup, Parakrant, Arijul, Prasenjit, Narendra, Judhisthir for giving me memories that I will take with me forever. I am grateful to Partha, Kanchan, Krishnendu, Saptarshi,Subhomoy, Soumyajitda, Krishnadulal, Debapriya and many more who have been awesome friends and messmates. I would also like to make a special mention for the cook Amit, who put in his best efforts to make the food as homely as possible.

I wouldn't be where I am but for the mere presence of some people in my life. Just knowing that they are a part of my life gives me the utmost sense of security and the strength to fight any battle destiny has in store for me. They have been the constant source of support in any which form, and the only source of my recharge when I am drained. I would like to thank my mother for proving that angels do exist, and standing testimony to the saying "God couldn't be everywhere, so he created mothers." I would also like to thank all my family members for their moral support, love and affection. Finally, I would like to thank a special friend, *Sumana*, for being the voice across the phone that could cheer me up in any situation, for the unending encouragement, support and care.

Sayan Sarcar

List of Symbols and Abbreviations

List of Symbols

β	Input string
η	Number of words
N	Order of ngram
w_i	Current word under composition
w_{i-1}	Previous word
w_{i-2}	Second previous word
$\succ$	Preference operator
ψ	Size of the prediction window
V	Vocabulary
$CSID_{seq}$	Sequence of Character Set IDentifier
$\widehat{\beta}$	Normalized version of β
final_list	Final ranked list
α	Word present in the *vocabulary*
α_i	i^{th} word in the *vocabulary*
λ	Probability mass distribution
$\widehat{P}$	Backoff probability
γ	1^{st} character in β
C	Number of counts for seen word sequence
γ_2	CSID(1^{st} character in α)

C^*	Discounted count
δ	CSID(1^{st} character in β)
$\widetilde{P}$	Discounting probability
$iTrans$	iTrans-encoded text
w_{i-2}^{i}	$w_{i-2}w_{i-1}w_i$
B_{Score}	Absolute value of logarithmic (base 10) value of a backoff probability
ED_{Equal}	The minimum number of edit operations needed to transform a typed word into a part of target word

List of Abbreviations

ANOVA	ANalysis Of VAriance
ATOMIK	Alphabetically Tuned and Optimized Mobile Interface Keyboard
AVIN	Assisted Visual Interface Notepad
BTC	Bayesian Touch Criterion
BTD	Bayesian Touch Distance
C-DAC	Center for Development of Advanced Computing
CMU-SLM	Carnegie Mellon Statistical Language Modeling
CSID	Character Set Identifier
FOCL	Fluctuating Optimal Character Layout
GA	Genetic Algorithm
HCI	Human Computer Interaction
HP	Hewlett Packard
HR	Hit Rate
IBM	international Business Machines
ICT	Information and Communication Technology

IF	Incorrect but Fixed
INF	Incorrect and not Fixed
InScript	Indian Script
KS	Percentage of Keystroke Savings
KSPC	KeyStrokes per Character
MT	Mouse Movement Time
MSD	Minimum String Distance
OPTI	Optimized Virtual Keyboard
PDA	Personal Digital Assistant
PKS	Percentage of Potential Keystroke Savings
SHK	Fingerprint Verification Competition
SVM	Support Vector Machine
SHARK	Shorthand-Aided Rapid Keyboarding
SPSS	Statistical Package for the Social Sciences
WoP	Without Prediction
WP	With Prediction
WPM	Words Per Minute
ZWJ	Zero-Width Joiner
ZWNJ	Zero-Width Non Joiner

Abstract

Of late, rapid advancement in ICT offers text-based interaction between people through digital devices in a large scale. Text input mechanisms in users' own languages are necessary for bringing the ICT advantages to the English illiterates. QWERTY keyboard, which has been designed for text entry in English, however, is not so suitable for text composition in other languages. Also, performance of the existing virtual keyboard based text composition is inferior for Indian language text entry.

In this work, we develop mechanisms to design text entry interfaces suitable for mouse, finger and eye gaze based interactions in large and small display devices for Indic languages. The mechanisms include development of virtual keyboards, word-level prediction method and eye-gaze based text entry interface.

We propose design principles to design virtual keyboard for text composition in Indic languages suitable for large and small display devices. Our design principle addresses Indic language related issues such as large character set, difficulties in composing texts due to presence of glyph and conjuncts etc., and interaction related issues like mouse movement minimization during text entry, inaccuracy due to fat finger problem etc.

To enhance the text entry rate, we propose word-level prediction mechanism to be augmented with the virtual keyboard interfaces in Indic languages toward enhancing the text composition rate significantly. It has been observed through user study that users commit many errors while composing text in Indic languages. Our approach predicts the correct words in presence of errors in the initial input. To rank the predicted words in the prediction list, we have implemented multivariate ranking algorithm. We also consider compressing language model files through arithmetic coding concept to run the proposed prediction mechanism in memory and processing constrained mobile devices.

We develop an efficient gaze-based text entry interface for English. The compact on-screen keyboard holds optimal size and space between keys which yields optimal eye movement during text entry task. To increase the gaze-based text entry speed and accuracy, we incorporate two strategies: a) providing visual feedback after selecting a key through highlighting next probable characters to reduce visual search time and saccadic eye movement and b) minimizing as well as diminishing dwell time with controlled eye gesture. As neighbouring wrong key selection error is common during text entry, we consider adjacent keys as an alternative to the selected character when suggesting next words. We further extend the mechanism to design Indic language interfaces.

To test the efficacy of the proposed approaches, we have implemented each approach in three Indian languages namely Bengali, Hindi and Telugu. The experiments with users reveal that the proposed text entry interfaces in Indic languages achieve fast and accurate text entry than the existing approaches. Further, the proposed mechanisms can be extended to other languages in the world.

Keywords: Text entry methods, virtual keyboards, eye-gaze based text entry method, text entry rate enhancement

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The rapid advancement of ICT instantiates the scenario that people all around the world communicate among themselves using hand-held digital devices like mobile, PDA etc. beside usual desktop PC and laptop. Recently, text entry has been evolved as an important way of communication as the rate of composition of e-mails, chats, blogs through different digital devices has significantly increased worldwide. Unlike English, text entry task in Indian languages is difficult due to linguistic complexities and interaction barriers with different size of digital devices. In this dissertation, we have investigated for different text entry methodologies in desktop and small display devices with three mode of interactions namely mouse, touch and gaze targetting majority of Indic languages.

The rest of the chapter is organized as follows. In Section 1.1, we review the current scenario of text entry tasks. Various text entry devices are discussed in Section 1.2. Section 1.3 includes different approaches to text entry method. The issues and challenges with text entry methods are discussed in Section 1.4. The scenario of text entry in Indic language is described in Section 1.5. Section 1.6 describes the scope and objectives of our work. Finally, the outline of the thesis is presented in Section 1.7.

1.1 Text entry tasks

Recent statistics reveal the prosperity of text-based communication. The number of worldwide email accounts continues to grow from over 4.1 billion in 2014 to over 5.2 billion by the end of 2018. Also, total number of worldwide email users, including both business and consumer users, is expected to increase from over 2.5 billion in 2014 to over 2.8 billion in 2018 [73]. Moreover, users send and receive on the average 196.3 billion

emails a day worldwide in 2014, and this is expected to grow to 227.7 billion emails a day by 2018 [73]. Delloitte reports that 21 billion messages were sent per day worldwide via SMS [47]. eMarketer expects 4.55 billion people worldwide to use a mobile phone in 2014 [52]. Mobile adoption is slowing, but new users in the developing regions of Asia-Pacific, the Middle East and Africa will drive further. Between 2013 and 2017, mobile phone penetration will rise from 61.1% to 69.4% of the global population [52]. The global smartphone audience surpassed the 1 billion mark in 2012 and will total 1.75 billion in 2014 [52]. Worldwide over 350 billion text messages are exchanged across the mobile networks every month [71].

Simultaneously, India contributes in the global growth of texting as the outgoing SMS per subscriber in India has been increased to 30 in March 2014 [198]. The mobile device market in India is rapidly emerging and people have started to use mobile phones in a larger way. Nielsen statistics [155] reported that the smartphone users in India were using their phones for more than 2.5 hours per day in 2014, of which communication (calls, SMS, emails) accounts for 28% of usage. TRAI reported that in India, outgoing SMS per subscriber became 30 in March 2014 [198]. India holds 17.5% population of the world [219] and 8.33% of its Internet users [188].

1.2 Text entry devices

There were significant changes observed in writing technology over 19^{th} and 20^{th} centuries [128]. In its first era, typing technology had been evolved with Shole-Gidden typewriter [178] as successful replacement of handwritten input technologies. A typewriter is a mechanical or electromechanical machine (Fig. 1.1(a)) for composing texts by means of keyboard-operated typing onto the paper. Typically, one character is printed per key press. The machine prints characters by making ink impressions of type elements similar to the sorts used in movable type letterpress printing. QWERTY keyboard was among the first few text entry methods, proposed by Sholes et al. in 1868 [178]. This layout was later copied by Remington and other typewriter manufacturers [128]. Such as keyboard became the dominant text entry method with respect to other alternate methods like index typewriter [233], double keyboards [128], Dvorak [51] layout because of several factors like quick learning rate, comfort in ten finger typing due to key arrangement and no requirement of layout switching etc.

Then typewriter was slowly replaced by desktop and personal computers (early in 1980s) when the price became affordable to the people, as it serves many purposes other than the sole text entry task in typewriters [128]. In desktop computers,

(a) Typewriter keyboard [178]

(b) QWERTY keyboard [178]

(c) Mobile 12-key keyboard [128]

(d) Smartphone [155]

Figure 1.1: Different text entry devices

hardware keyboard was maintained as the command input tool to the operating system (Fig. 1.1(b)). The QWERTY is widely selected as de facto layout among keyboard manufacturers for accommodating keys in the character area. The reason of its success is due to an effective typewriting layout. There exist alternate ways through which text entry can be possible in a desktop computer like speech recognition [154], chording keyboards [202] etc.

Though telephone was invented in the 19^{th} century, first mobile phone was appeared in market in 1980s [128]. Initially, landline phones were deployed in the market with speaking facility only. In the first generation of mobile device, 12-key keyboard was

evolved as a popular text input hardware where calling and messaging both can be done with a single keypad. As telephone keypad was followed unanimously among mobile phone manufacturers, 12-key keyboard (Fig. 1.1(c)) outperformed other alternate text entry methods like Fastap [118], BlackBerry reduced QWERTY keyboard [128] etc.

Handheld computing devices, concept envisioned by Alan Kay in 1968 [128], became popular in late 1990s allowing users to have computer anytime anywhere. In 1996, a smaller, simpler and more affordable pen-operated device, named as *Pilot* developed by *Palm Computing* company (currently acquired by *HP* [1]), introduced in the market. Such a device supports stylus and touch-based interaction, virtual keyboard and gesture-based text entry methods. In this genre, the first mobile phone to incorporate PDA features was an IBM prototype developed in 1992 [221].

Ericsson Mobile Communications released *Ericsson R*380 [62] in early 2000 which was the first device marketed as a "smartphone" (Fig. 1.1(d)) [130]. Combining the functionalities of mobile phone and PDA, smartphone supported web browsing in a resistive touchscreen by a stylus. In 1999, the Japanese company *NTT Docomo* released the first smartphones for public use within a country [165]. In the context of smartphone operating system, *Symbian* was the most popular in Europe during the mid- and late 2000s [155]. Entertainment-focused smartphones were first popularized by *Nokia* with the *Nseries* from 2006. In 2007, *Apple Inc.* introduced the *iPhone*, one of the first mobile phones where multi-touch interface was used. The *iPhone* became popular as it accommodates a large touchscreen for direct finger input instead of a stylus, keyboard, or keypad based input of smartphones [20]. In 2008, first Android based phone namely *HTC Dream* (also known as the *T-Mobile G1*) released [146, 203]. Android is an open-source platform founded by Andy Rubin and backed by *Google* [2, 36]. which gained widespread popularity in 2010 and now dominates the market. In the current scenario of smartphone devices, screen size and thumb or finger based user interaction along with virtual keyboards remain as an effective text entry method.

Towards further advancement, in 1999, *Samsung* launched the world's first watch phone [87]. From then, a new era of small devices was evolved. Smart-watches provide users the platform to access many applications on smartphones directly from their wrists, without the need to touch their smartphone. Many companies like *Samsung*, *Sony*, *Pebble*, *Seiko*, *LG*, *Motorola* etc. develop smart watches in current time with different size and shapes. Recent versions of smart watches have supported many digital tasks, including text entry [106]. Current attempts are known to incorporate on-screen keyboards into smart watches having ultra-small display.

[1] www.hp.com

The trend of quick change in text input devices with perspective of mobility and shrinking device size propels the fast and accurate text entry mechanism in coming years.

1.3 Text entry methods

Since the invention of desktop computer, hardware keyboard maintained its superiority over other methods in the context of text composition. As the time progressed, the size of computing devices got decreased causing significant change in interaction while performing text entry task. There exist an enormous amount of methods proposed in literature, this section listed a subset of them. Based on the basic classification of text entry methods made by P. Isokoski [93], we categorize the methods in six groups namely a) keyboards, b) handwriting recognition, c) unistrokes, d) speech recognition, e) gesture recognition and f) eye movement and fixation (see Fig. 1.2).

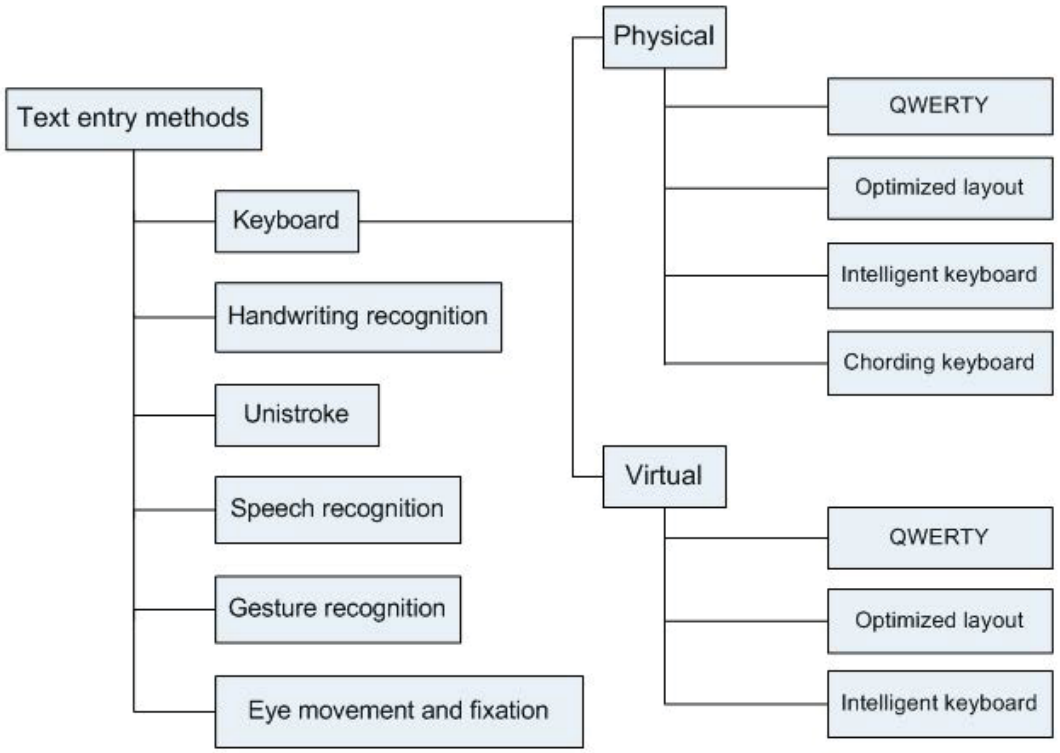


Figure 1.2: Different text entry methods

1.3.1 Keyboards

Typewriter keyboards became popular at early stage and developed a large body of skilled users already before the computers entered the offices. The QWERTY-layout, developed and patented by Sholes at al. in 1868 [178] is still the popular layout for desktop computers. Meanwhile, many key arrangements other than QWERTY have been exercised, but no design stands suitable than the former one. Keyboards are further divided into two groups, physical and virtual where the former is specified as hardware QWERTY keyboard attached with desktop PC and later indicates software keyboards visible on-screen in a mobile devices.

1.3.1.1 Physical keyboard

Traditional physical QWERTY-keyboard contains 104 key buttons in it whereas recently developed keyboard designs like Half-QWERTY [138], FOCL [12], SHK [190] etc. contain fewer number of keys. Within this group, four keyboard variations are observed namely QWERTY, optimized layout, intelligent keyboard and chord keyboard. It has been observed that QWERTY layout is difficult to user initially, but with easy learning it becomes fast and accurate method for text input. Originally, the QWERTY keyboard layout was designed to minimize mechanical jamming of the keys in typewriters [233]. Further, to optimize ten finger typing speed, Dvorak [51] keyboard layout was developed. In case of first generation mobile device, 12-key telephone keypads are used where selection of a character requires one or more than one key presses [45, 128]. Another variation in physical keyboard design, namely "stick keyboard", was developed by Green et al.'[72], where the standard QWERTY keyboard is compressed into a single home row. Here, key selection remains similar to the key selection task in 12-key keypad. Some intelligent keyboards were also developed to increase user comfort. As an example, Half-QWERTY keyboard [138] divides the layout into two equal parts and displays one part at a time. Whereas, switching between two parts can be done by pressing spacebar. For mobile devices, the half-QWERTY keyboard was developed by Matias et al. [139]. The Single Hand Key Card (SHK) [190] is an eighteen key keyboard designed to be used with four fingers of one hand placing two characters on a key. The card is held between the thumb and the fingers with the base of the card resting against the palm. These intelligent keyboards execute complex algorithms for setting the keyboard state. In intelligent keyboards, the state of the keyboard is implicit because the computer handles it automatically. With chord keyboards [70], the large set of states is not hidden. Instead, the user explicitly chooses the keyboard state by pressing several keys simultaneously.

1.3.1.2 Virtual keyboard

As an alternative to hardware keyboard, virtual keyboard also called soft keyboard was introduced and explored vastly in both large and small screen display devices. We categorize all virtual keyboard based text entry methods into three major groups (following the grouping proposed by P. Isokoski [93]) namely QWERTY, optimized and intelligent keyboards. The popular choice is to preserve the QWERTY leyout in virtual keyboard layout maintaining the consistency in user habits. However, the QWERTY layout is suboptimal in terms of two thumb and multi-finger text entry. So, quantitative optimization was performed based on mainly two objective functions: movement time (MT) and movement distance (d) [18]. A substantial amount of research on optimized touchscreen keyboard layouts has been done ([46, 63, 120, 121, 129, 182, 236, 238] with an increasing degree of sophistication and mathematical rigor. Lewis et al [121] used Fitts-digraph model (a combination of Fitts law and bigram probability) to optimize the layout. Goodman et al. [67] proposed a virtual keyboard which automatically corrects typing errors using a model that combines pointer hit location with character level language model. Isokoski [92] designed a software keyboard augmented with a circular marking menu [116] holding eight character blocks in it suitable for single finger touch or stylus based interaction. Nesbat [153] presented a soft keyboard based on 12-key telephone keypad assigning letters to keys based on their bigram probabilities. Frequent letters are accessed by double tapping the pen where others are accessed by double tapping two different keys. The main objective of developing text entry interfaces is to enhance the text typing rate with error correction support. It may be noted that virtual keyboards in mobile devices are not very fast while typing with stylus or a single finger such as index finger. As to literature, the speed accuracy trade-off in text entry can be mitigated by two ways: a) disambiguating between the point of touch through finger or thumb and the user intended point on the device screen and b) minimizing language redundancies through a language model [111]. T. Masui [137] proposed an intelligent text entry interface namely *POBox*. The keyboard layout is standard QWERTY layout and on each key press, a menu containing most probable to be completed words is displayed. Other than English, *PoBox* was also implemented for Japanese language. A closely related system is the reactive keyboard [44] that provides similar predictions for desktop keyboarding. *ShapeWriter* [110], another intelligent text entry method, allows user to construct shape gestures while composing a word. Kristensson and Zhai claimed that each word in the lexicon has a corresponding template unique gesture [110]. The user's intended word is found by a pattern recognizer that primarily looks at the scale-translation invariant shape similarity between user's gesture and the set of template gestures.

1.3.2 Handwriting recognition

Handwriting recognition has been evolved as an effective text entry method during the late 20th century [163, 193]. The recognition can be done in two ways, on-line [163, 193] and off-line [104, 151]. We can differentiate these two based on information available to the recognizer. Methods for on-line handwriting recognition are primarily data-driven which rely on collecting and classifying vast amounts of users' handwriting. Another less popular approach is based on template-matching, for instance energy-deformation or elastic matching methods [193]. For text entry in mobile devices, handwriting recognition has the benefit that users already know how to write. Recent study concludes that state-of-the-art handwriting recognition achieves significantly higher text entry rate after several hours of practice [113].

1.3.3 Unistrokes

Goldberg and Richardson [66] proposed the influential text entry system called *Unistroke* gesture. Unistrokes is designed for character-level text entry in pen-based mobile devices and can be built motivated by three primary design goals: 1) efficiency, 2) accuracy and 3) visual attention. *Graffiti*, marketed by *Palm Inc.*, is another set of character gestures, but heavily inspired by Unistrokes. In the alphabet set, individual strokes resemble the corresponding letters more prominently. Another robust set of character-level stroke alphabets namely *EdgeWrite* was designed by Wobbrock et al. [227]. The *EdgeWrite* alphabet can be easily used within a square area surrounded by edges which leads to stability and minimizes error. Perlin proposed a word-level pen-based text entry method called Quikwriting [162] where the concept is that the user need not to lift up the pen between inputted letters until a word is completed. Mankoff and Abowd [135] designed a pen-gesture text entry method called *Cirrin* (Circular input). The circular shaped keyboard was made in such a way that while composing a word, no such character, which is not a member of that word, will be accessed.

1.3.4 Speech recognition

In recent times, speech-based text entry task becomes highly appealing which allows eye-free and hand-free interactions as well as provides significantly faster text entry. Speech-based text entry primarily consists of two steps: (a) automatic speech recognition (ASR) and (b) error correction for accuracy. Speech recognition mechanism performs well enough in desktop environment. However, its performance degrades drastically in noisy situations. The development of speech recognition algorithms suitable for mobile devices

have also been practiced [235]. Mobile devices lacks sufficient memory for storing and processing of speech signals. Limited battery power in the smartphones also hinders the deployment of speech recognition process on mobile devices [192]. Fischer et al. [56] proposed server-based speech recognition technique which suffers from real-time network delays. Some speech recognition engines were also developed for mobile devices (such as *PocketSphinx* [86] and *PocketSUMMIT* [80]). Vertanen et al. [212] built *Parakeet*, a continuous speech recognition system whose interface was designed to correct errors easily on a hand-held device while on the move.

1.3.5 Gesture recognition

Several text entry techniques which follow algorithms of recognizing gestures to compose character(s) and word(s) are known. All techniques can be divided into two groups: 2D gesture created on large or small touch-enabled surfaces and 3D gestures developed in-air. Character and word forming gesture recognition mechanisms, executed in both large and small screen devices, are considered in the first group. The most prolific 2D gesture recognizer is *$1 recognizer* [229], which is a 4-step algorithm to recognize a predefined gesture extracted from a finite alphabet of unistroke gestures. The algorithm was later extended to multi-touch (*$N-gesture recognizer*) [6,7] and*Point Cloud* ($P-family) [211] to enable gesture recognition in different context. Other algorithms like *Octopocus* [10], *turning angle* algorithm [88] use template alphabets to recognize gestures. Symbolic gestures were extensively studied by many researchers [6, 122, 166, 211, 229] as they are common to everyday interaction. For 3D gesture-based text entry, Ni et al. [156] and Kristensson et al. [108] explored *Graffiti* unistroke gesture set into 3D space. Mid-air handwriting recognition has also been exercised using various sensors and camera-based recognition systems (e.g. [4,170,240]). Kin et al. [102,103] proposed a multi-touch system for character recognition based on an algorithm for reconstructing 3D trajectory from the acceleration sensor signal.

1.3.6 Eye movement and fixation

Movement and fixation of human eye gaze were explored as an alternate mode of text entry in last few decades [131]. Initially, gaze was used as an input modality of physically disabled person but gradually it becomes common for able-bodied also. Since last three decades, different research works have been carried out to provide fast and accurate gaze-based text entry (i.e. *eye typing*) systems. Existing eye typing interfaces allow character entry through three mechanisms [11], *Eye Typing* where eye movement and

dwelling are involved [77, 78, 133, 186], *Eye Gesturing* where drawing gestures with eye makes corresponding character or chunk entry [91, 228] and *Continuous Eye Writing* where eye movement path coverage initiates a command of specific character or chunk entry [208, 209].

1.4 Issues and challenges with the state of the arts

Many text entry methods have been reported so far to perform text composition task in the wide range of digital devices. Nevertheless, there exist five challenges of developing text entry method in general, not specific to any device, people, interaction etc. These issues are elaborated below.

1.4.1 Localization

The design of the desktop hardware keyboards varies depending on the protocols of the country. As an example, German language QWERTY keyboard is designed as QWERTZ and spanish as AZERTY. As the hardware keyboard supports entering a single character in each button press, a different keyboard incorporation within operating system requires remapping of keys. On the other hand, for replication of text entry methods which depends on the vocabulary, it is needed to recreate the dictionary with target language words. Collecting such words requires huge effort. Moreover, text entry methods relying on language models (i.e unigram, bigram, trigram models) need sufficient text data to train those models. It is challenging to arrange the training corpus which represent what user intended to write. As an example, for developing Indic language based intelligent text entry method for mobile phones, we require huge collection of *Facebook* or *Google* chats and blogs written by users in particular native language (in Unicode) to prepare the training corpus for that language. The speech and handwritten recognition based text entry methods are unable to perform without the support of user's articulations. Thus, to redesign any speech recognition system from a language L_1 to L_2, developers need to rebuild the acoustic model freshly for L_2.

1.4.2 Error correction

We can not avoid user error occurrence during text composition, even the method is equipped with strong language model. According to P. O. Kristensson [107], user errors can be broadly classified into two groups, cognitive errors and motor errors. The first one is occurred as wrong spelling of the intended word is stored in user's mind. In Indic

languages, this type of error is common as users often get confused among graphically or phonetically similar characters. The other one, motor error, is committed by user due to tremor or stress (like errors due to *fat finger* problem are inevitable while performing thumb-based interaction in Indic language text entry systems). Both class of errors, specifically motor errors, get increased as the task becomes more complex. Apart from these two error types, a new class of errors is observed which occurred due to system. It occurs sometime when the recognizer has incorrect model for the user which makes some obvious mistakes. As an example, many word-level prediction systems (such as *Shapewriter* [114] and *Panini* [53] keypad for mobile text entry in Indic languages) do not find dictionary matched word(s) if a out-of-vocabulary word enters into the texts in composition. As user errors are unavoidable, it is required to detect and correct them for which, an efficient mechanism is required. As a solution, some researchers implement error correction mechanisms which worked on different modality. As an example, error occurred in a composed text through speech recognition system [212], users are allowed to correct errors through touch gestures. In fact, there exists an urgent need of developing an efficient error correction interface which will improve text entry rate and reduce frustration among users.

1.4.3 Editor support

Efficient error correction methods also require good editor support. The text entry support is provided by operating system through desktop keyboard. The process by which character-by-character text entry is done is one directional where no proper way exist to retrieve previous text dispatched through user's working window. In spite of providing intelligent text entry method, present desktop system does not have operating system support to allow users to edit previously typed words for better alternatives, if error occurs. This becomes the real issue of any desktop bound text entry method.

1.4.4 Feedback

According to P. O. Kristensson [107], an important criteria of being intelligent text entry mechanism is to provide feedback to the user after text entry. The immediate feedback, in this context, stands as presenting the user inputted texts. It is required for users if they committed any mistake during the composition. The efficiency of text input algorithm also can be judged on the basis of tolerance level (i.e. how much noise in the input they can handle). Most of the word processing applications support annotating non-dictionary words by spell checkers which users prefer while using a text entry method,

such as desktop keyboard. One way of providing feedback is to predict and display possible to be completed and next words at runtime. This word-level prediction utility not only gives user a opportunity to correct errors (by selecting correct words from the prediction list) but also indicates the correct word spelling to users. Another possible variation of prediction carrying smaller impact is character-level prediction which helps user to compose words by providing unambiguous character flow which leads to avoiding error(s) in composed text chunk. Another effective feedback mechanism which can be augmented with on-screen keyboard would be providing visual feedback to next probable characters. As user eyes are attracted with changing color of the character keys in the keyboard, highlighting next probable character would become effective with respect to increasing text entry speed.

1.4.5 Context of use

This challenge is specifically on the basis of designing for the heterogeneous contexts that users spend with mobile devices on daily basis. In this case, creation of a text entry method with higher text entry rate and lower error rate is not enough. The method should understand and support users' needs in particular situations. In current situation, researchers do not posses sufficient knowledge about how users use or intends to use mobile text entry methods. We also do not have any idea about user preferred modality in different situations and contexts. So, it is urgent to come up with intelligent text entry method which accurately sense the surroundings based on models of both environment and users' preferences. Overall, an interface is required that enables user to easily monitor, understand and control its behaviour.

1.5 Text entry in Indic languages

Obeying the constraint of standardization, government of a country tries to adapt English hardware QWERTY keyboard after introducing computers to native script which does not necessarily a feasible solution. Now-a-days, compatibility is possible for software keyboard because of standardized Unicode font. The constraints for the hardware keyboard are, the number of key is limited and each script is assigned to a limited number of code points (128) in Unicode. These two limitations force choices on the character allocation both in the keyboard and in the Unicode character set for the script. In this section, we identify text entry issues in Indian languages.

Unicode is evolved as a viable standard in most Asian countries resolving compatibility problems. Unicode (2003) recognizes the structural similarities among

the Indic languages with the encoding differs across different countries. The Unicode character sets for the Indian scripts are based on ISCII-1988 (Indian Standard Code for Information Interchange). In Unicode, the character sets for the Indian scripts are as follows: Devanagari ($U + 0900 to U + 097F$), Bengali ($U + 0980 to U + 09FF$), Gujarati ($U + 0A80 to U + 0AFF$), Gurmukhi ($U + 0A00 to U + 0A7F$), Kannada ($U + 0C80 to U + 0CFF$), Malayalam ($U + 0D00 to U + 0D7F$), Oriya ($U + 0B00 to U + B7F$), Tamil ($U + 0B80 to U + 0BFF$), and Telugu ($U + 0C00 to U + 0C7F$) (U indicates the number actually is in Unicode chart which can be indexed as \U followed by number). For these scripts, the characters are organized in the same manner (like vowels, consonants, glyph etc.) and are found in the same location [128]. Also, Indic language scripts are phonetic and most of them fall under alphasyllabary category [97]. The Indic scripts are halant-based, which means that conjuncts are being employed to delete the inherent vowel.

As each Indic language script is assigned to only a limited number of codes in the Unicode chart, the complex character creation is being performed by a shaping engine which combines several glyphs in order to create a character shape. This shape changing task involves the notion of the orthographic syllable and the use of zero-width characters.

The basic unit for the shaping engine is an orthographic syllable which contains base consonant. Other consonants, dependent vowel and vowel signs can be attached to the base consonant. It may be noted that the order of input may not necessarily match the final shape. For example, *ki* is typed as *ki* but displayed as *ik*. The rules for input order and display may vary across the different Indic scripts because of differences in implementation. It is the task of the shaping engine that to identify syllables for each script in a stream of text and then apply rules to define the shape of the entire syllable. To explain the working steps of the engine, suppose we take one example of input in Bengali as প [p]+ ◌্ [্]+ র [r]+ ি [i]= প্রি [pRi]. The shaping engine first performs the combination of the diacritic for r (halant) and p and placed a curved line below to the p and the glyph is displayed at the beginning of the syllable. Once an orthographic syllable is formed and displayed, it is treated as a "cell" where the cursor cannot be placed within it.

In some Indic languages, there exist characters having equivalent Unicode representation. For example, the word रिज्रर्व [rijRb] can be composed as र [r] + ि [i] + ज [j] +. [़] + र [r] + ◌् [्] + व [b] (contains seven characters) as well as र [r] +ि [i] +ज्र [j] +र+◌् [्] +व [b] (consists six characters) [31]. Note the multiple representation of characters with *nukta*. In addition to this, the use of "Zero-Width Joiner"[1] (Unicode value U200D) and

[1] Zero-width joiner, http://en.wikipedia.org/wiki/Zero-width_joiner

the "Zero-Width Non Joiner"[1] (Unicode value U200C) represent a conjunct in different ways. For example, "क्ष" [kSha] (unicode sequence $U0915 + U094D + U0937$) in different forms can be represented as " क्ष" (unicode sequence $U0915 + U094D + U200D + U0937$) and "क्ष" (unicode sequence $U0915 + U094D + U200C + U0937$) [40]. According to Unicode consortium [39] various characters are analyzed visually as consisting of multiple parts. These compositions, in fact, are not valid to compose text although they appear to be correct [39].

Many languages exist within India and there are variations in developing text entry method because of differences in both the scripts and the accepted standards. There are six main issues concerning standard text entry method development in Indic scripts.

- *Large character set:* Indic languages use a large number of base characters. Also, text composition includes combination of consonant and dependent vowel, consonant and consonant with the help of *halant* etc. All possible such type of combinations can not be accommodated into the standard keyboard. Researchers propose to map more than one characters in a single key with the use of special keys, namely *Shift*, *Ctrl*, and *Alt* key, for selection of suitable characters [128]. affect the word prediction performance by decreasing the keystroke savings and imposing huge cognitive load on users.

- *Keyboard layout:* In the QWERTY keyboard, the assignment of characters on specific buttons is decided based on the usability principles. The Indic keyboard design is driven by a need for compatibility across all Indic scripts, which is a linguistic problem solution. In case of mobile phones where the display is even more restricted, key assignment becomes critical.

- *Input sequence:* The diacritics in Indic languages are not necessarily written or read in a linear sequence. In other words, the writing order and the phonological order may not match in Indic script [90, 128]. Further, the position of a character in a word may not be fixed. For example, to compose a word निर्मित [niRmit] user has to select the characters in the order: न [n] +ि [e]+र [r] +◌्+म [m] +ि [e] +त [t]. Note that this type of required ordering demands enough cognitive load on users in addition to trapping into failure of the prediction system.

- *Typographical variants:* In Indic language, several words have multiple correct spellings and alternate representation forms [30,31]. The character "*anuswar*" can be used as both half-na (e.g. हिंदी [hiNdii] and हिन्दी [hindii] and half-ma (e.g. मुंबई

[1] Zero-width non-joiner, http://en.wikipedia.org/wiki/Zero-width_non-joiner

[muMbaii] and मुम्बई [mumbaii]) [30,31]. In addition to this, it also have dual forms which are technically correct but only the one form is acceptable (as in लिए [lie] and लिये [liye]) [9] in standardized Hindi. According to "Centre for Development of Advanced Computing"(CDAC), some misspelled words in Indic languages are more significantly in use than their grammatically correct counterpart. For example, the word जांच [jaaNcha] is incorrect but is used more often than its correctly spelled form जाँच [jaa.Ncha] [30].

- *Presence of phonetically or graphically similar characters:* There exist characters or their combinations which are phonetically similar (sounds alike) [176,197] as (श [sh], ष [Sh] and स [s]), (ऋ [Ri] and री [rii]) and (ई [i] and यी [yii]) and (ए [e] and ये [ye]) etc. Some of the characters are so much similar to the others in shape that there remains a finite chance of confusion between भ [bha] and म [ma]; घ [gha] and ध [dha]; and ख [kha] and रव [rava] [64]. These confusing characters make task of text entry more erroneous.

- *Text entry interface:* Text entry mechanism should support a higher text entry rate, minimum keystrokes, lower cognitive load on users and minimum hand movement time for composing text accurately [60, 128]. Also, the system should be easy to learn and provides an effective way of correcting mistakes. An effective text entry method must address common challenges for intelligent text entry methods namely localization, error correction, editor support, feedback, and context of use [107]. The same is more pertinently true in the context of Indic languages and indeed it is a challenge to deal with all issues.

1.6 Research Objectives

Addressing the limitations of existing text entry methods and common challenges of intelligent text entry, in this dissertation, we set our objective to develop text entry interfaces suitable for mouse, touch and gaze-based interaction in Indic languages. We mention scope and objective of our works in the following.

- In the first work, we have focused on developing user friendly virtual keyboard based Indic language text entry method suitable for both desktop and mobile devices supporting both mouse and touch-based single pointer interaction. Our main aim is to elevate the existing text entry rate experienced through Indic language methods as well as reducing user error rate in composed text which is yet to be explored precisely. Moreover, mouse and touch-level interaction issues

with existing desktop and mobile virtual keyboard solutions are grossly ignored. Motivating with these issues, we plan our main objective to develop efficient virtual keyboards for Indic language supporting interactions with large and small screen devices. The keyboard design objective is divided into two parts, desktop-based and mobile-based. Objectives for designing virtual keyboard for desktop devices are a) designing layout with optimum placement of characters minimizing mouse movement time and b) placing of frequently used dependent vowel or glyph group supporting less area occupying of the keyboard. On the other hand, the keyboard meant for large display devices may not be to the mobile device with small display. Moreover, small size of the keys and single finger/thumb based interaction result wrong key selection due to *fat finger* [24] problem. This becomes a critical issue in general and particularly in Indian languages. Addressing these limitations, we define objective to develop efficient text entry interface in Indic languages as follows, a) accommodating a large alphabet set in a small-sized display area, 2) designing interface layout by addressing interaction problem along with mitigating search time to find a key.

- In our second work, we have targeted to develop word prediction method to be augmented with virtual keyboards in Indic languages. Google provides a typing tool called *Google IME* in different Indic languages which supports prediction (word completion without error correction) facility augmented with an alphabetical keyboard layout to enter text directly in application such as Microsoft Word. In contrast, whenever user selects *Backspace* key to correct typing error, the prediction system fails to provide any further prediction support. *Lipik* [174] is a multi-lingual predictive software which supports text composition in Indic languages where user needs to press a *spacebar* after completion of a word to populate the next possible words. However, the interface lacks in populating correct words if user enters wrong character(s) in the composed text chunk. Existing word-level prediction mechanisms in Indic languages are unable to detect and correct user errors. Addressing these limitations, the major objectives of the proposed word-level prediction system are as follows: a) handling several types of error occurred at any position of a word, b) implementing an efficient multi-variate method for ranking to select the best candidates for the prediction list under typing, c) mitigating the complexities of Indic languages text entry. Moreover, the same method can be augmented with mobile text entry interfaces addressing the constraint of implementing prediction algorithms.

- Our third work is based on developing gaze-based text entry method for Indic languages. To do this, we have analyzed common issues of developing gaze-based text entry (*Eye typing*) interfaces and then investigated language or specific interaction related issues. With reference to the work related to the layout of eye typing interface, the approaches [79, 145, 186, 208, 239] mainly focused on utilizing less screen area deciding the reduced size of the key buttons, smaller space between buttons and minimum number of buttons present in keyboards [79, 145, 186, 208]. However, there is no work reported which develops layout designing strategy focusing on eye movement optimization. Several approaches [79, 145, 186, 208, 239] have been proposed for minimizing [79, 208, 239] as well as diminishing [145, 186] dwell time during eye typing. Still, there exists a scope to explore a better approach which can reduce the key searching time along with diminishing dwell time. To the best of our literature review, no work exists on developing eye typing interface in Indic language. The major objective of gaze-based text entry method development are as follows: a) developing an on-screen keyboard layout with optimum eye movement, b) diminishing dwell time and error rate in gaze-based text entry providing visual feedback through highlighting next probable characters.

1.7 Contributions of the Thesis

There are several text entry mechanisms known for text entry in English, which is, however, scarce particularly in Indian languages. This thesis attempts to bridge the gap and investigates different modality of interactions. Outcome from our research dissemination in the following.

- A virtual keyboard layout has been proposed, which is mechanism suitable for text entry in Indic languages where keys are arranged optimizing mouse movement time.
- We design a user interface suitable for text entry in Indic languages with handheld mobile devices. The interface accommodates a large alphabet set in a small-sized display area with a character prediction facility. It also provides solution to the *fat finger* problem and reduces visual search time.
- We develop an efficient word-level prediction mechanism suitable for fast and accurate Indic language text entry with less number of key presses.
- Apart from dealing with the linguistic complexities, the proposed prediction mechanism handles several types of user errors occurred during text composition

in Indic languages.

- We implement an efficient multi-variate method for ranking of the candidate characters for the prediction list.
- An on-screen keyboard layout with minimum eye movement is developed in English as well as Indic languages.
- An efficient mechanism is proposed diminishing the dwell-time while performing eye typing task.
- We provide a visual feedback after selecting a key through highlighting next probable characters in order to mitigate key searching time.

1.8 Organization of the Thesis

The introductory chapter provides statistics which reveal growing demand of text entry task. It also discusses about past and presently used text entry devices and envisions future trend and expected demands. A vast work on text entry methods is summarized then. We also focus on issues and challenges of text entry in general and specific to Indic languages. Then, we discuss text entry scenario in Indic language, specific linguistic and device interaction problems followed by scope and objective of the thesis work. Finally, we conclude with the research contributions that come out from the work. The rest of the thesis is organized as follows.

Chapter 2 : Related Work

This chapter gives a detailed literature survey of text entry mechanisms suitable for mouse, touch and gaze-based interactions in English as well as Indic languages.

Chapter 3 : Virtual Keyboard Design in Indic Languages

We describe the proposed mechanisms of virtual keyboard design for mouse-based desktop and touch-enabled mobile devices in this chapter. The experiments and experimental results with proposed and existing keyboard designs are also given in this chapter.

Chapter 4 : Word-level Prediction in Indic Languages

We describe proposed word-level prediction mechanism augmented with text entry interface in Indic languages, which can detect and correct user typing error, in this chapter. The user experiment results with augmented text entry interface and existing interfaces in desktop as well as mobile devices are provided in this chapter.

Chapter 5 : Development of Gaze-based Text Entry System

The development of gaze-based text entry interface is described in this chapter. The user experimental setup and results with the proposed and existing interfaces in English along with the proposed design for Indic languages are described in this chapter.

Chapter 6 : Conclusions and Future Research

This chapter concludes our study in the domain of text entry mechanisms and discusses potential future research directions in this field.

Chapter 2

Related Work

In this chapter, a survey of literature related to the research investigation made in this dissertation is reported. This dissertation aims to develop text entry methods suitable for mouse, touch and gaze-based text entry in Indic languages. As the current digital devices are not providing hardware key buttons on the device, people need to compose text through software or virtual keyboard. Design of virtual keyboard depends on many factors like device size, language alphabet size, interaction pattern, usage habit of the people etc. Sufficient work on designing virtual keyboards have been reported throughout the last decade in English, the most popular language to interact with digital devices. On the other hand, few keyboard designs have been achieved for text entry in Indic languages. These designs support character by character text entry achieving very less text entry rate. The first section of this chapter describes virtual keyboard design principles for large desktop to small mobile display devices in English and Indic languages. One popular way to enhance text entry speed significantly is augmenting word prediction method with a keyboard. A detail about available word-level prediction methods in English as well as Indic languages are described in this chapter. Also, in this thesis, we explore eye movement and fixation driven text entry methods for English and Indic language text entry. We survey the existing methods related to the eye-gaze based text entry in this section.

This chapter is organized as follows. Various text entry systems suitable for desktop and mobile devices in English and Indic languages are reviewed in Section 2.1. Section 2.2 covers a discussion about the word-level prediction strategies augmented with text entry interfaces in English and Indic languages. Work related to gaze-based text entry mechanisms are presented in Section 2.3. Finally, the chapter is summarized in Section 2.4.

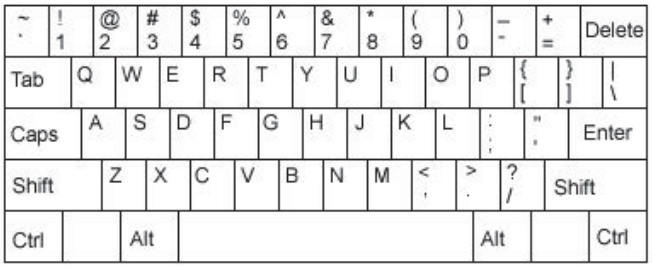
2.1 Virtual keyboard design

In this section, we present virtual keyboard based text input methods into two parts: (a) for large/medium display devices and (b) for small screen display devices. Virtual keyboards in English followed by existing Indic keyboards are discussed in each part.

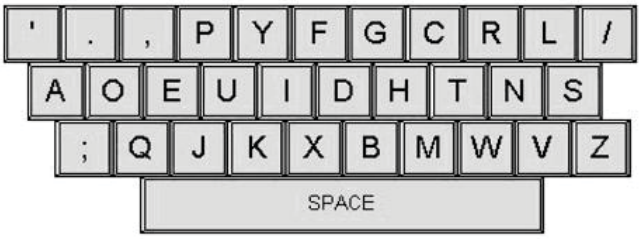
2.1.1 Virtual keyboard design for large/medium display devices

QWERTY keyboard is the most commonly used text composition tool. The keyboard layout was invented by *Sholes* [178] and the name came from the first six characters of keys in a keyboard [128]. The keys are arranged primarily to overcome the mechanical jamming occurred in typewriters (shown in Fig. 2.1(a)) Nevertheless, the layout was not designed in order to optimize human capabilities and limitations, not even the single and multi-finger typing. As an alternate, Dvorak et al. [51] proposed *Dvorak* keyboard layout which supposed to be easier to learn, more accurate, faster and less fatiguing. The key arrangement supports less finger motion, more typing rate and less error compared to QWERTY keyboard [175].

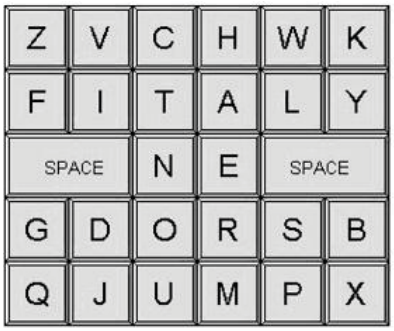
Specific to the single pointer based text entry, some virtual keyboards were designed. *FITALY* keyboard was designed to optimize hand movement during text entry with one finger, mouse or stylus [57]. It has two space bars in the layout (see Fig. 2.1(c). The keyboard name was taken from character sequence in second row. The frequently used characters like E, T, A are placed closed to the space bar. The keys in the *Lewis* keyboard, developed by Lewis et al. [120] is alphabetically arranged. The layout yields better text entry rate than QWERTY for single finger typing (see Fig. 2.1(e)). Another keyboard layout, *OPTI*, was optimized to increase typing speed using trial and error. The design involves Fits' law [58] and digraph frequency [15] of characters in English (shown in Fig. 2.1(d)). Metropolis algorithm has been proposed by Zhai et al. [238] to minimize energy (tension). The algorithm is a *Monte Carlo* method widely used in searching for the minimum energy state in statistical physics. The high-performance keyboard design problem can be mapped into searching for the structure of a molecule (keyboard) at a stable low energy state determined by the interactions among all the atoms (keys). Their approach follows *Random walk* in the virtual keyboard design space. In each step, the algorithm picks a character key and moves it in a random direction by a random amount to create a new configuration followed by evaluating *Fitts' law*. After that, whether the new configuration is kept as the starting position for the next iteration depends on the Metropolis algorithm. Developers replace the key circle shapes with hexagons in the design which cover up the gaps and make more efficient use of the total space



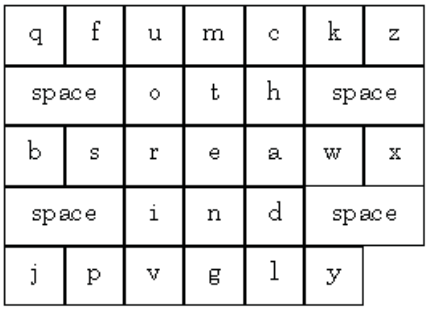
(a) QWERTY keyboard [178]



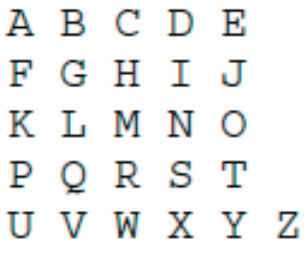
(b) Dvorak keyboard [51]



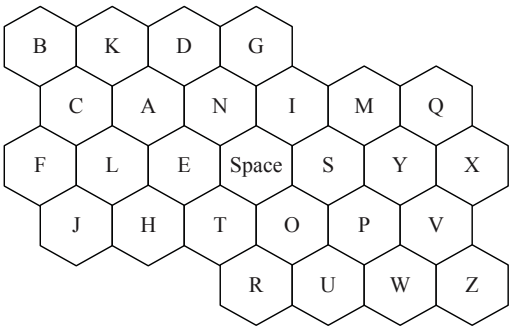
(c) FITALY keyboard [57]



(d) OPTI keyboard [129]



(e) Lewis keyboard [120]



(f) Metropolis keyboard [238]

Figure 2.1: Popular virtual keyboard layouts in English

(see Fig. 2.1(f)). Further, to design virtual keyboard layout suitable for single finger or stylus based text entry in PDA and Smartphones, DellÁmico et al. [46] applied *Quadratic Assignment Problem* to solve the optimum key arrangement problem.

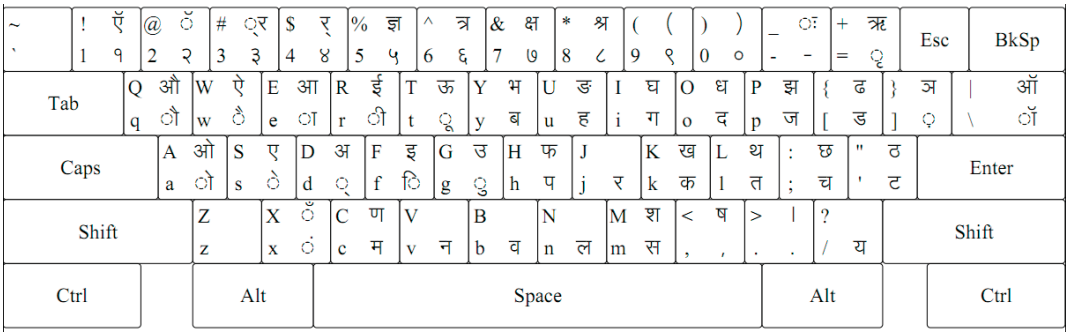
Various layouts have been proposed to enter texts in Indic languages which have been incorporated in hardware as well as in virtual keyboards. This section highlights some of the designs implemented in Indic language virtual keyboards in large as well as small screen devices.

InScript keyboard layout (Fig. 2.2(a)) [65, 196], developed by CDAC, was standardized by Government of India for composing text in Indic languages [194]. This is the standard keyboard for 12 Indic scripts including Devanagari, Bengali, Gujarati, Gurmukhi, Kannada, Malayalam, Oriya, Tamil, Telugu etc.

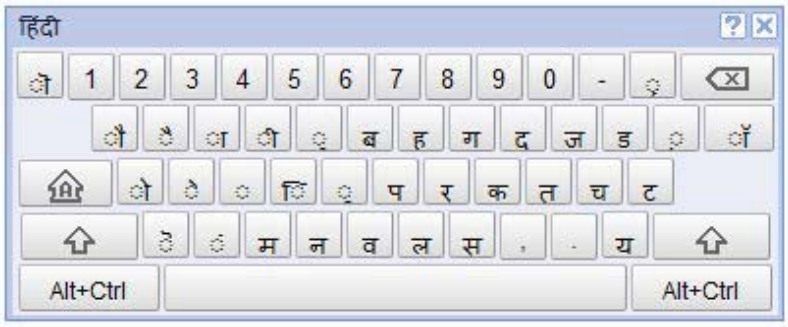
As the hardware keyboard is majorly made with English keys and the manufacturers never think of making keyboards with Devanagari keys, the decision has been taken to place the stickers that can be stuck on top of the keys so that one can understand where a particular language character is mapped on a particular keyboard. Now, as the number of characters present in Indic languages are more than that of English, mapping all characters into keyboard was an issue. To address this, more than one characters in the *InScript* keyboard are mapped to a single key in the keyboard. Users need to select the "Shift" button in order to use the character associated in a key.

A software version of the *InScript* keyboard, that is, virtual keyboards, following *InScript* layout [196], contain each key having two character mapping which can be accessed using "shift" key alternatively. This has been implemented in different commercial softwares following two design principles: a) all characters present in the layout are displayed at a stretch and the alternate characters can be accessed using special key (such as *Lipik* interface [124] in Fig.2.2(e)) and b) there are two separate layouts for displaying all characters where one layout shows characters which do not require "Shift" key and other layout contains the characters which needs "Shift" key to be entered (see in Fig.2.2(d)). At a time, only one layout is visible and alternate one can be activated using "Shift" key. Google [69] (Fig. 2.2(b)) and Microsoft [144] keyboard layouts follow this design principle.

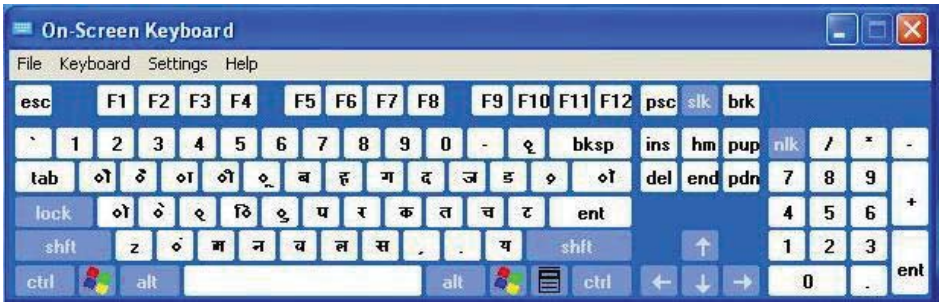
A popular virtual keyboard layout called *Avro* [159] for Bengali language is shown in Fig. 2.2(f). Here, keys are grouped in alphabetical order as well as some frequently used complex characters are kept to minimize the key press required in forming them. *Guruji* [76] has developed another alphabetical keyboard to support users in composing Indic language texts by means of reducing text composition time. This keyboard uses different colors to distinguish consonants, vowels and numeric keys (see in Fig. 2.2(g)).



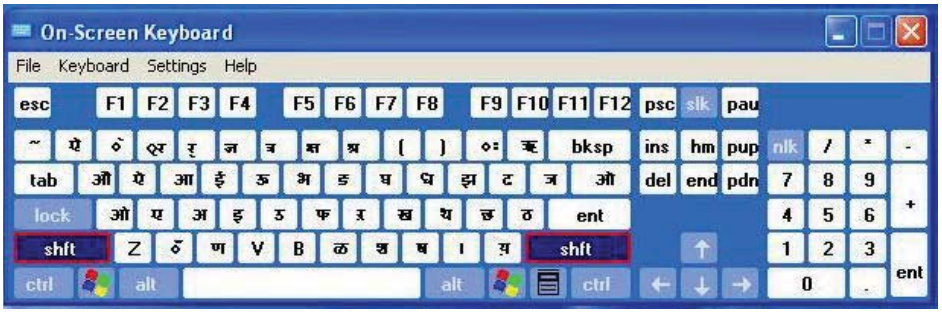
(a) InScript keyboard layout [65]



(b) Google keyboard layout [69]

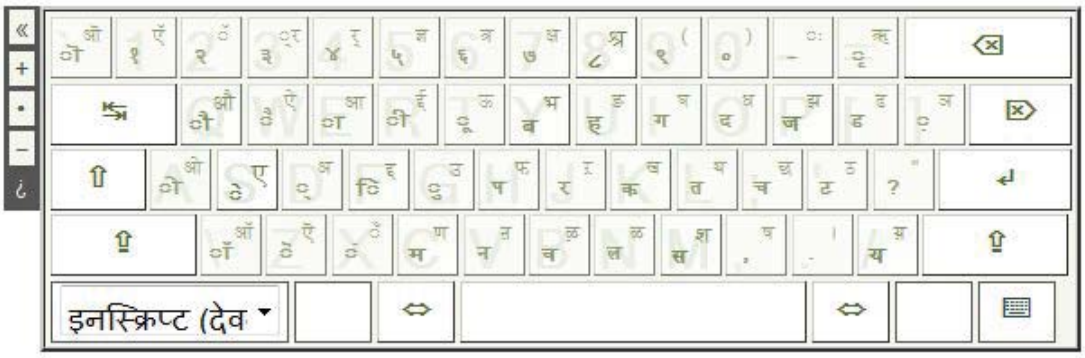


(c) Microsoft keyboard (without pressing Shift key)

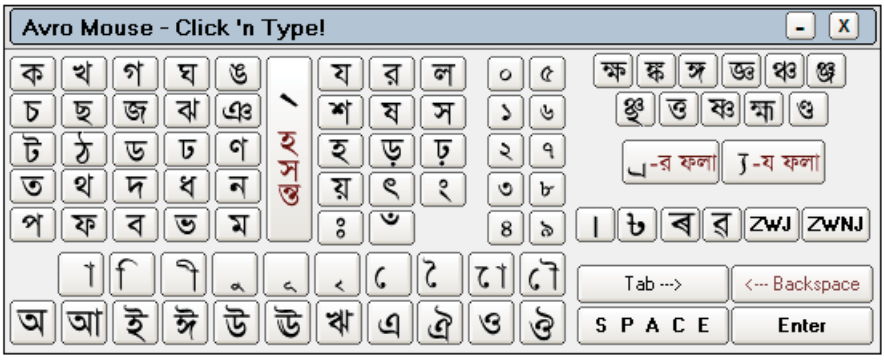


(d) Microsoft keyboard (with Shift key pressed)

Figure 2.2: Popular virtual keyboard layouts in Indic languages (Contd.)



(e) Lipik keyboard layout [124]



(f) Avro Bengali keyboard [159]

(g) Guruji Telugu keyboard [76]

Figure 2.2: Popular virtual keyboard layouts in Indic languages

2.1.2 Virtual keyboard design for small display device

Researchers observe that the QWERTY virtual keyboard layout is suboptimal in case of single pointer text entry (i.e. typing with single pen, stylus or finger) [16,113]. However, the layout remains good for the practical use. Many virtual keyboard layouts developed for large/medium devices have also been explored for mobile devices (*Lewis* [120], *OPTI* [129], *Metropolis* [238] etc.). Since 1980s, after realizing the shortcoming of QWERTY as a virtual keyboard, researchers focused on designing optimized virtual keyboard layouts with more rigorous optimization methods [119, 129, 238]. Lewis et al. [120] used Fitt's law and character-level bigram to develop more efficient keyboard layouts. In the optimized *OPTI* layout, Mackenzie and Zhang [129] used similar model to accommodate keys. *Alphabetically Tuned and Optimized Mobile Interface Keyboard* (ATOMIK) [236] was designed to address the problems with QWERTY keyboard and to get better text entry rate in single-finger typing in mobile devices (Fig. 2.3(a)). The layout is alphabetically tuned having general trend that letters from A to Z are placed from the upper left corner to the lower right corner of the keyboard. According to the experimental result, the arrangement mitigates mental load of novice users in finding keys. Kristensson and Zhai [237] developed *Interlaced QWERTY* (*iQWERTY*) keyboard where keys of each row of QWERTY are divided into two interlaced rows in order to help users to visually locate the characters in the keyboard (Fig. 2.3(b)). Bi and Zhai [16] designed virtual keyboard layout applying *Metropolis* optimization. They obtained two layouts, one which was optimized without QWERTY constraint (called as *freely optimized*, see Fig. 2.3(c)) and the other where optimization was achieved with QWERTY constraint (*Quasi-QWERTY*, see Fig. 2.3(d)). In the layout, common consecutive letter pairs are placed on the opposite side of the keyboard to improve finger movement efficiency. Bi et al. [18] extensively analyzed the effectiveness of mathematically optimized methods, specially Fitt's law for mouse or stylus movement to develop multilingual virtual keyboard ($K5$ keyboard layout)(Fig. 2.3(e)). Also, they analyzed the approaches for inputting diacritic characters in optimized multilingual touch screen keyboard. Dunlop and Levine [49] proposed two touchscreen keyboard layout designs (Fig. 2.3(g)) based on three design metrics namely minimizing finger travel distance in order to maximize text entry speed, to maximize the quality of spell correction by reducing tap ambiguity and maximizing familiarity through a similarity function with the standard QWERTY layout. This methodology supports single finger text entry in mobile devices. In addition to the improvement in speed, these layouts achieve moderate reduction in neighbour key selection ambiguity which would lead to improved tap interpretation and spell correction. Mankoff and Abowd [135] proposed *Cirrin* keyboard layout to facilitate stylus based text

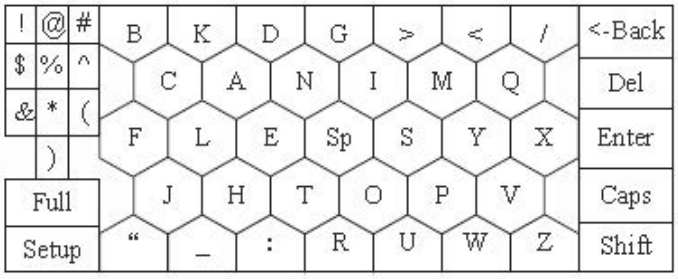
entry. In this keyboard, input was generated from the coordinates of the point where the stylus crossed the interface. For example, composition of word "cirrin" is shown in Fig. 2.3(f). Keyboards supporting dynamic reorganization of the keys were developed to reduce the distance between tapped and intended character. The examples are like Sibylle [215] and FOCL [12, 126]. When a key is pressed, the layout changes according to the bigram frequencies between characters in corresponding language. Theoretically, the performance of such systems is significantly improved, but they distract users to anticipate the next keystroke. Also, key searching time, after each keystroke, gets increased.

Placing a less number of keys in small screen mobile devices becomes a design issue, in general, which has been addressed by multi-key arrangement, where each key holds multiple characters. As a result, each key size becomes larger which, in turn, reduces the chance of occurring touch-level key selection error. User needs to follow multi-tap method to select a character in this type of virtual keyboard. Virtual keyboard was designed following the key arrangement as 12-key (3×4) telephone keypad (Fig. 2.3(h)) [74]. The keypad contains number keys $0 - 9$ and two additional keys (* and #). Characters are placed alphabetically three on each key spread over $2 - 9$ keys. The keys are arranged following common character placement in most of the mobile phone hardware keypad. Dunlop et al. [50] proposed a semi-ambiguous touch-screen soft keyboard for English, called *QWERTH* (see Fig. 2.3(i)), in which they grouped character keys according to their bigram probability occurrences (like one key is 'TH' as their bigram probability is higher, and so on). Five such character groups were placed in a row of the keyboard. As this key arrangement demands a large button size, and hence is more suitable for large fingers and thumbs.

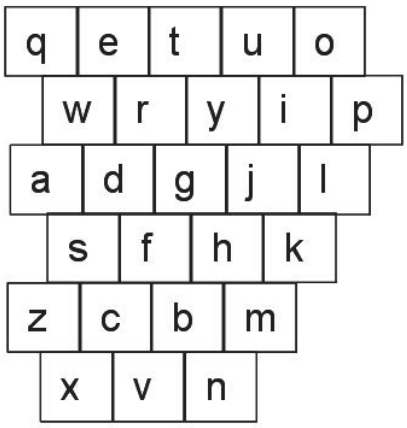
Few works have been done considering touch-screen mobile phone based Indic language virtual keyboard design [8, 14, 53, 96, 97, 115]. These designs can be majorly categorized with respect to design principles as a) alphabetic, b) phonetic, c) frequency based d) 12-keypad and e) others.

For Indic language text typing, the *InScript* layout [195], designed on top of the QWERTY keyboard and is available on most operating systems [96] (for example, the Hindi layout is shown in Fig. 2.4a). In spite of wide availability, *InScript* layout has not been widely adopted by users till date. As a consequence, some virtual keyboards have been implemented following the standardized *InScript* layout for text entry in mobile devices (e.g. *Sparsha* layout in Telugu, Fig. 2.4d) [8,96].

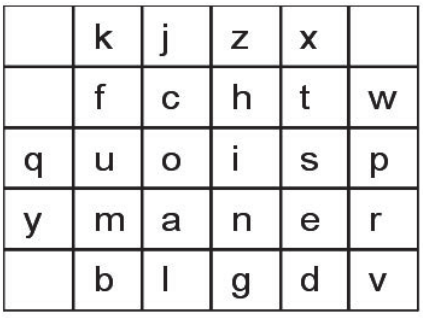
Saral [115] is a Devanagari script text input design specifically proposed for mobile phones. *Swarachakra* [96], the Indic language text input system extended from *Saral*,



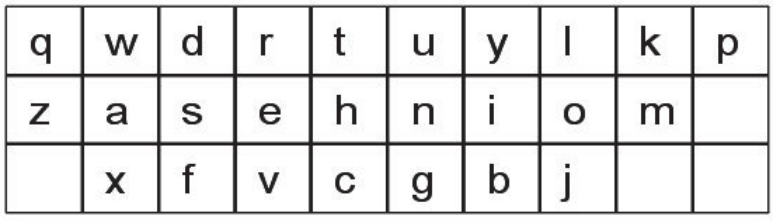
(a) ATOMIK keyboard layout [236]



(b) iQWERTY keyboard [111]

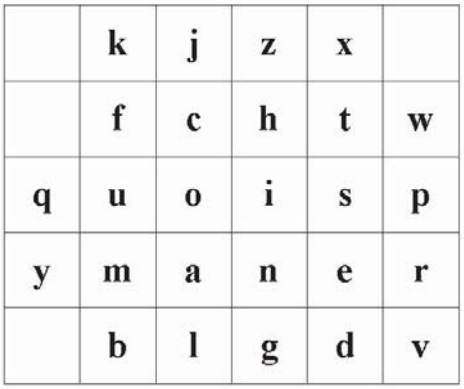


(c) Freely Optimized keyboard [16]

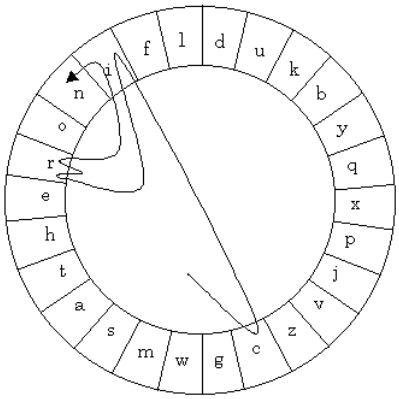


(d) Quasi-QWERTY keyboard [16]

Figure 2.3: Virtual keyboard layouts for mobile devices (Contd.)

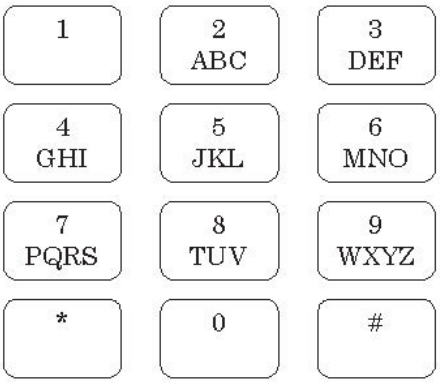


(e) K5 keyboard [18]



(f) Cirrin keyboard [135]

(g) Pareto-optimized keyboard layout [49]



(h) 12-key keyboard [74]

(i) QWERTH keyboard layout [50]

Figure 2.3: Virtual keyboard layouts for mobile devices

developed by Joshi et al. [96], maintains alphabetic as well as phonetic character arrangement in the keyboard (for Bengali language, see Fig. 2.4c). After selecting a consonant character, the interface displays all possible glyphs (dependent vowels attached with the consonants) arranged in a circular panel over the key. In the *Just Hindi* keyboard, the first character is only displayed and pressing of it initiates the display of a group of characters. Slight variation in the design principle was implemented by Jung et al. [97] in their proposed keyboard where the inflection glyph, after selecting a consonant key, are displayed at the top of the interface in a horizontal fashion. Mayabi soft keyboard [141] is also an on-screen keyboard with Bengali phonetic and English language support for Android platform in mobile devices.

Bhattacharya et al. [14] designed text input interface for composing text in Bengali (Fig. 2.4e). As number of characters present in alphabet is moderately large, they propose a two-level design. 5 two-level virtual keyboards were designed representing three design paradigms (alphabetic, frequency-based and adaptive). The comparison study reveals that for mobile devices, the two-level adaptive design gives the best performance in terms of text entry rate and accuracy. Hinckle et al. [82] developed a multi-layered frequency based keyboard layout suitable for mobile devices where an extra hovering layer is appeared on selecting consonant characters. Further, they applied multi-objective optimization approaches to design virtual keyboards in many Indic languages which yield a moderately higher text entry with minimum visual search time and user errors while typing.

Luna Ergonomics Pvt Ltd developed a technology (*CleverTexting*) [53] to compose Indic language texts through 12 keys of mobile devices (Fig. 2.4b for Hindi layout). *CleverTexting* technology is a new way of typing on the mobile phone in any language of the world (for example to send SMS) with the existing keypad (often known as Panini keypad), that offers the convenience of typing with single key presses and is comfortable to user thumb. It uses statistical prediction, which after selecting a character, dynamically changes the keys of the layout and brings new keys. It is useful to users when they got habituated with the mechanism; otherwise, every key-press changing sometime distracts users and leading to loose their interest soon.

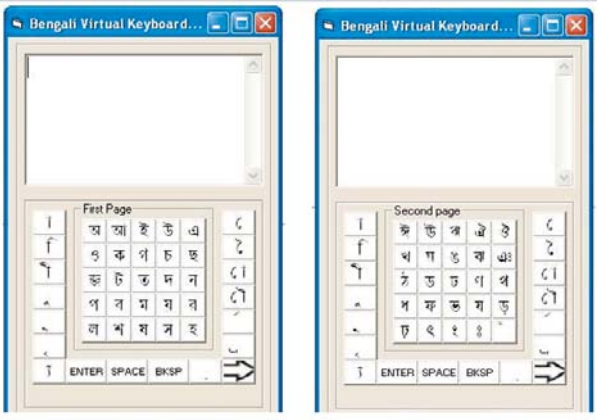
Among the touchscreen based proposals, the 'Intelligent multi-layered input' scheme [172] provides a radical approach to use vowels only to write phonetic Indic scripts. This approach uses a systematic grouping and intelligent layout based on frequency of use. Its benefit is the minimum number of keys required, but drawback might be the high requirement on the learning curve of the user.

(a) Inscript Hindi keyboard [195]

(b) Panini Hindi keyboard [53]

(c) Swarachakra Bengali keyboard [96]

(d) *Sparsh* Telugu keyboard [15]



(e) Bhattacharya and Laha, Bengali keyboard [15]

Figure 2.4: Mobile on-screen keyboards in Hindi, Bengali and Telugu

2.2 Word-level prediction mechanisms

In recent years, several methods have been reported to develop word-level prediction systems. We categorize our survey as the basic approaches to word prediction techniques, in general, and for Indic languages, in particular.

2.2.1 General techniques to word prediction

The conventional text prediction methods can be classified as statistical and syntactical methods [3,28,29,44,60,199,210]. Statistical methods are based on probabilistic language model to predict desired words (such as unigram, bigram and trigram) [98, 99, 136]. In addition to this, some systems utilize the last recently used information for prediction [59]. It has been found that the efficiency of word prediction can be improved with the use of recency model [21,55,60,81,117,152]. On the other hand, the syntactical methods use the syntactical information such as parts-of-speech tag information [27, 54, 55, 98, 136, 140, 161, 231]. Here, the goal is to ensure that systems can suggest grammatically correct words to users. However, Fazly [55] observed a small amount of improvement in syntactical method over statistical approach with the cost of huge computation time. She also reported that the simple bigram approach is approximately 6.5 times faster compared to linear combination between statistical and syntactical method [54]. Effect of prediction quality on text communication driven by AAC (Augmentative and Alternative Communication) users was also investigated by Trnka et al. [201].

Few systems support error correction along with prediction whose working principles are stated below. The VITIPI system is developed by Boissiere et al. [22] predicting those words which are not present in vocabulary by inferring certain analogies. It is also claimed that the system enhances the text-entry speed and improves the quality of the composed text with respect to text prediction system such as HandiWord [59]. In VITIPI, while a user starts writing with first few letters of a word, it displays either the complete word or a part of it, unless there is no ambiguity. It may be noted that, VITIPI system doesn't display the list of unexpected words but provides the ending of words without end-user's intervention. An overview of VITIPI system is shown in Fig. 2.5. It may be noted that if the vocabulary size is large, in this system, length of common character chunk gets smaller resulting decrement of keystroke savings. To make a correction, the system uses *mathematical distance* (Edit Distance) as well as 200 French orthographic rules [22]. Those rules rewrite a part of a word with an equivalent character group. VITIPI predicts 26% words of the French vocabulary having 5,930 entries. It also corrects 72% of typing errors and 75% of orthographic mistakes [23].

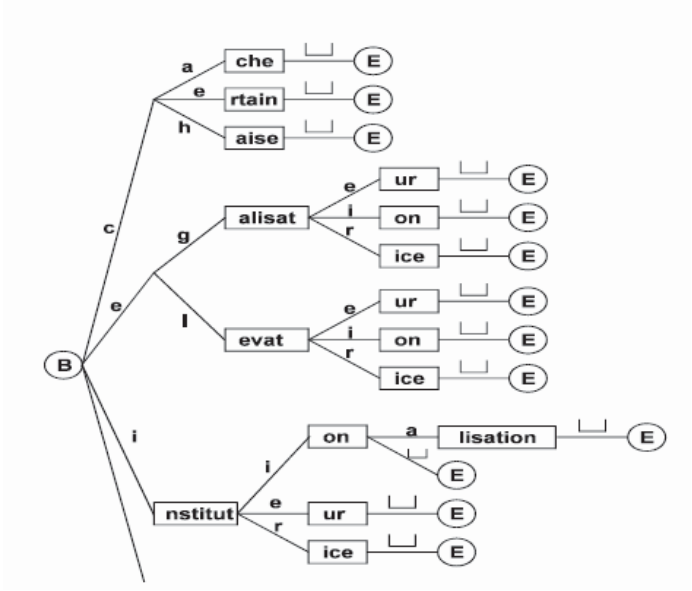
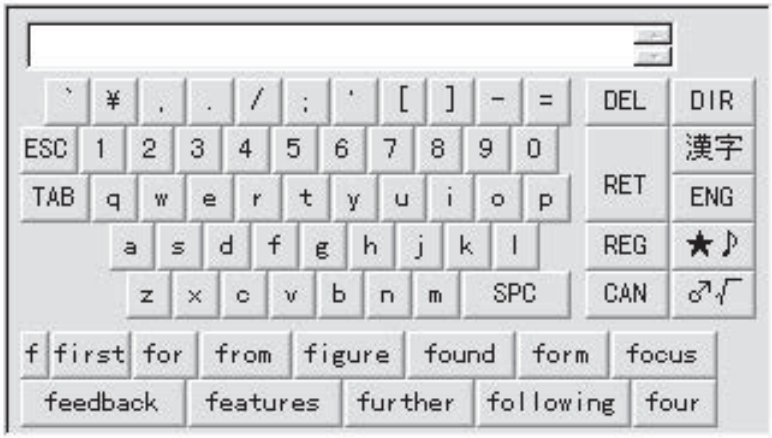


Figure 2.5: Working of VITIPI system [22]

POBox (Pen-Operation Based On eXample) is a prediction system which uses pattern matching to predict text [137] (Fig. 2.6). If input text pattern does not accurately matches any of the words in the vocabulary, then approximate string matching is performed based on two strategies: spatial approximation and pattern matching allowing errors. This strategy is effective especially when the soft keyboard is small and precise selection is difficult. For example, if the user fails to tap the right position of a soft keyboard and instead taps a wrong character. As a specific example, "*dtns*" to enter "*dynamic*", no word in the dictionary matches "*dtns*" and POBox automatically searches the dictionary using the less strict pattern *[ersdfxc][rtyfg][hjbnm][weasdzx]*, based on the arrangement of ASCII keyboard. In pattern matched allowing errors, this strategy uses shifter algorithm with limitation the wildcard to the basic ".*" pattern in order to achieve simple and fast processing [137]. For example, when a user does not remember the spelling of "***M***e***d******i******t***e***r***ra***n***ean" in the candidate list, he can specify "***mdtrn***" to see the list of words which are close to the pattern and then can find the right word in the list. Few word prediction programs are commercially available in the market like WordQ, Accessibility Suite, Co:Writer etc.[1]. Some systems incorporating other text entry rate enhancement strategies were also exercised by researchers like Dasher [217], SwiftKey [191], Swype [157] etc. All of them are available for non-Indic languages.

[1] A list of word prediction softwares with their descriptions are available at `http://www.abledata.com/abledata.cfm?pageid=19327&top=11314`



(a) POBox - normal typing

(b) POBox - using string approximation

Figure 2.6: POBox interface [137]

Shark2 / Shapewriter [110] is a gesture-based text entry mechanism for touch-screen keyboards. This method allows user to type quickly in a mobile phone by sliding or swiping the finger over a touch-enabled on-screen keyboard. As an example, to write the word "the", user presses the finger on the T key, slides to the H and E keys, and then lifts up the finger on the E key. This mechanism identifies the shape of this finger movement (gesture) using a pattern recognizer. With practice, the shapes of words can be learned by users which make them able to quickly recall the shapes for words without looking much at the keyboard.

We highlight some of the existing systems which deal with correcting errors based on auto-correction [83] mechanism. They usually don't save any keystroke even if there

is no typing error. On the other hand, some on-screen keyboard correction methods have been explored in recent literature [107]. A combination of language model and a touch model was applied by Goodman et al. [67] to increase accuracy on soft keyboards. The combined model provides the best key sequence for a touch sequence by maximizing the likelihood, significantly reducing error rates. Rudchenko et al. [167] with their Text Text Revolution game, showed that per-user touch models and personalized key-target resizing can further improve typing accuracy. *Automatic Whiteout* [34] is one of them which aimed to improve typing speeds and accuracy by automatically correcting user typing errors before they are displayed on the screen. *Whiteout* considers "off-by-one" errors which are mostly occurred due to accidentally pressing the adjacent to the intended key. The working methodology gets improved in *Automatic Whiteout++* [33] mechanism which uses character level trigram probability along with adjacent information and the time between previous and the current keystrokes to predict characters. They potentially reduce distractions and enable a user to continue inputting text without interruption.

2.2.2 Word prediction in Indic languages

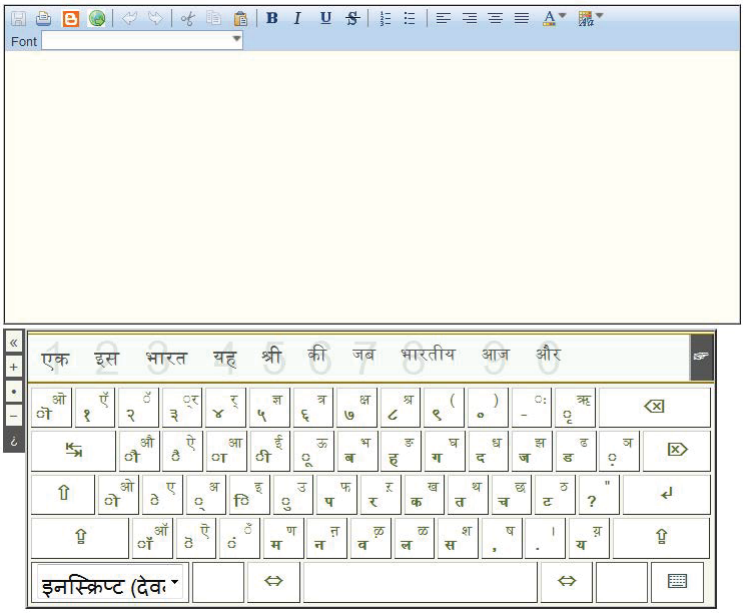
In Indian scenario, only few works have been reported so far. In the following, the existing works are reported.

SulekhA is an interface for neuro-motor disabled people augmented with word prediction system developed by Mukherjee et al. [148]. It implements scanning mechanism for Bengali text composition. The keys in virtual keyboard attached with the interface are arranged in alphabetical fashion. To predict a word, it uses the bigram probability and display top 12 words into prediction window.

Some utility programs are available on the Internet for search-query composition in Indic languages. These are somewhat similar to word prediction during document composition. *Google Search* [69] provides a word/multi-word prediction mechanism to predict search keywords. It is augmented with InScript[1] virtual keyboard and displays result vertically in the prediction window (Fig. 2.7(a)). In this system, when a user enters a prefix of a word, it returns top ten possible suggestions. In addition, it also provides multi-word to be predicted and displays those in the prediction window. After selection of the predicted word from the window, it searches the Internet and returns results. It may be noted that *Google Search* corrects error in the search query words while composing them, but only for non-Indic languages. Also, Google provides a typing tool called *Google IME* [69] in different Indic languages. It supports prediction (word completion without error correction) facility augmented with alphabetical keyboard layout to enter

[1] InScript keyboard, http://en.wikipedia.org/wiki/InScript_keyboard

(a) Google interface for Hindi text composition [69]



(b) Lipik interface for Hindi text composition [124]

Figure 2.7: Indic language interfaces augmenting word-level prediction

text directly in application such as Microsoft *Word*. To enter text character-by-character, users need to select keys from virtual keyboard whereas they require hardware keyboard access to enter *Space* and *Backspace* keys. Whenever user selects *Backspace* key to correct typing error, the prediction system fails to provide any prediction.

Lipik [124] is a multi-lingual predictive software which supports text composition in Hindi language. It has a built-in virtual keyboard based on *InScript* layout [194] to enter texts. The prediction window is horizontally organized and placed between text and keyboard area (Fig. 2.7(b)). It displays up to 10 suggestions at a time. Once a word is completed, user needs to press a space bar to populate the next possible words. Note that it necessitates one additional key press on each successful completion of a word apart from the selection of a word from the prediction window.

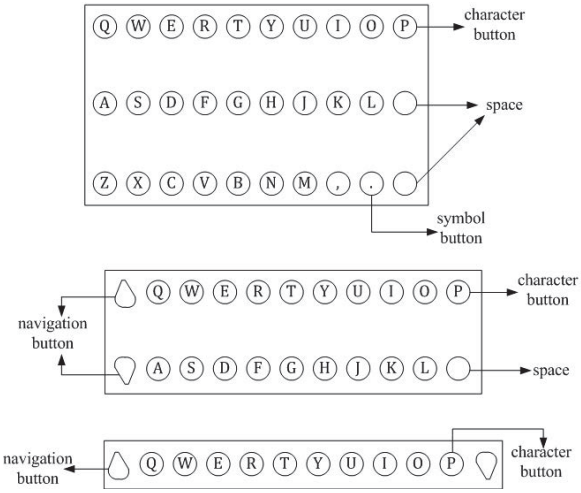
All the above mentioned works do not handle typing errors while composing text and simultaneously saves keystroke. In fact, in order to correct the spelling error(s) user has to manually delete the entered character when he notices an error otherwise errors remain in the composed text (it appears to user that the entered word is not present in vocabulary, so he ignores the errors and continues the text composition).

2.3 Eye gaze-based text entry mechanisms

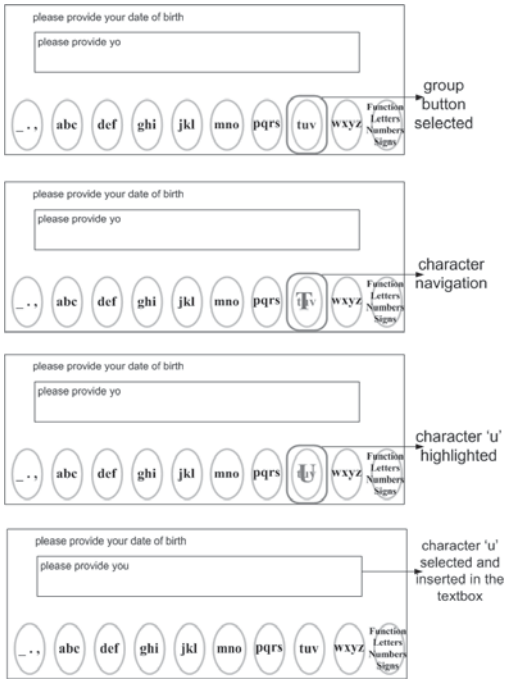
A number of research work have been carried out to provide fast and accurate gaze-based text entry systems. All existing systems mainly follow three mechanisms [207]: *eye typing* where eye movement and dwelling are involved, *eye gesturing* where drawing gestures with eye are involved and *continuous eye writing*, where eye movement in specific order to enter text. This work is relevant to the first category of eye-gaze based text entry mechanism and hence we limit our survey to the existing gaze-based eye typing only. The work on this topic mainly focuses on the development of on-screen layout for eye-gaze typing, minimization of dwell time and minimization of visual search time. In the following, we discuss the existing eye typing interfaces in English. It may be noted that, to the best of our survey, no such gaze-based text entry system has been developed for Indic language text entry.

2.3.1 On-screen keyboards for eye gaze typing

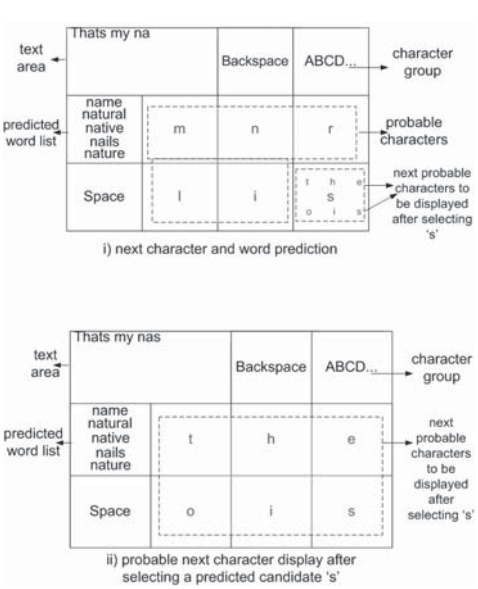
Špakov and Majaranta [186] proposed an on-screen keyboard so that screen area can be efficiently utilized. They designed three variations of the scrollable keyboard having different number of rows (one, two and three) as shown in Fig. 2.8(a). The two- and one-row keyboards contain navigation keys which are utilized to display the characters residing next layers. In their layouts, a suitable distance between two successive rows has been maintained because the drifting problem of eye-gaze is more in vertical direction than in the horizontal direction [186]. It may be noted that the scrollable keyboard layout variations with three, two and one rows need less screen area than other keyboards.



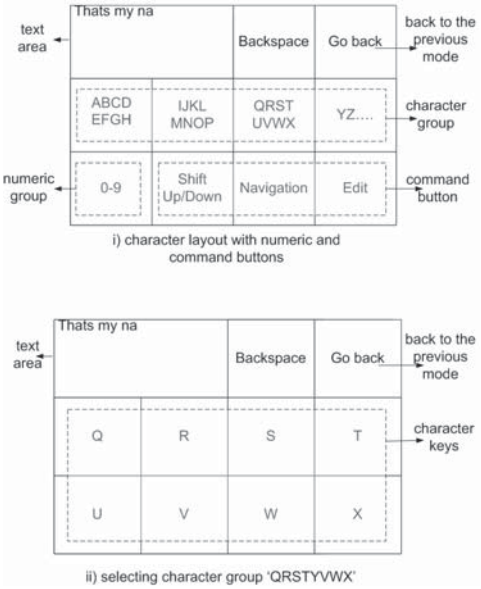
(a) The scrollable keyboard in three variations [186]



(b) Symbol Creator interface [145]

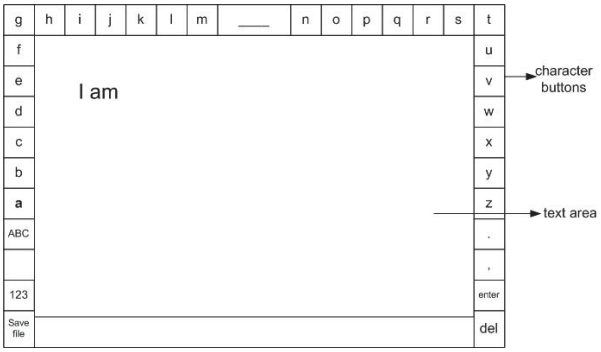


(c) The GazeTalk interface-Mode 1 [78]

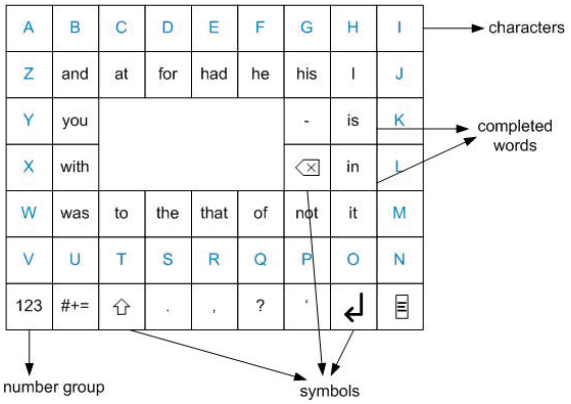


(d) The GazeTalk interface-Mode 2 [78]

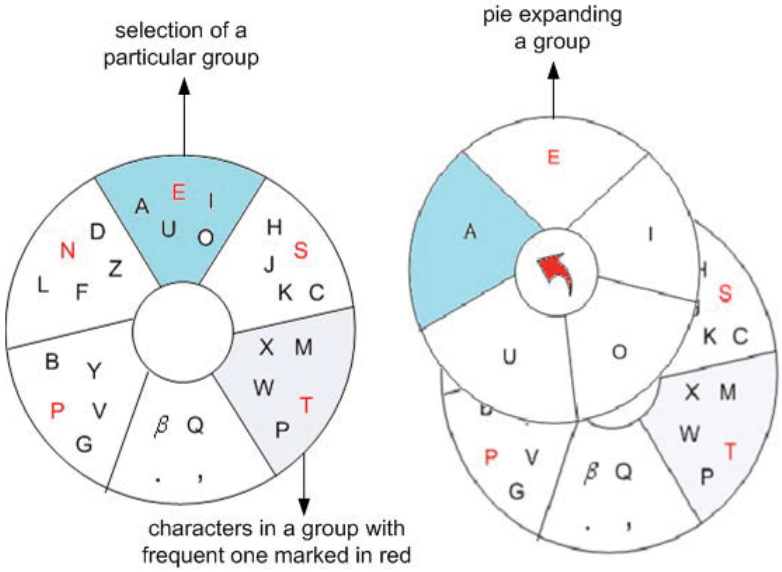
Figure 2.8: Different gaze-based interfaces - Part 1



(e) The Iwrite interface [208]



(f) The AVIN eye typing interface [239]



(g) The pEYEwrite pie-menu interface [209]

Figure 2.8: Different gaze-based interfaces - Part 2

GazeTalk [77] is a text entry system which supports two modes of entering texts: a) next character and word prediction mode and b) alphabetic layout mode which displays full keyboard with character and numeric grouping alongwith command buttons. Initially, keyboard shows word prediction list containing candidate words (when a chunk of character is typed) as well as next words (when a word is completed). Also, the keyboard consists of six buttons (placed in a three $\times$ two matrix layout) containing six next most frequent letters (generated by a letter prediction algorithm) which are updated automatically depending on the last entered character (Fig. 2.8(c)(i)). The next character buttons, on eye hovering, also shows the next six probable characters (Fig. 2.8(c)(i)) to be placed on six buttons (in row-major order) after the hovered character entry (Fig. 2.8(c)(ii)). For the other mode, if user presses the 'ABCD' character group button, then characters are displayed into four groups (each containing eight characters except last which contains two characters) in the keyboard (Fig. 2.8(d)(i)). Remaining one button represents numeric group and three buttons carry commands (Shift up/down, navigation and edit) useful for text editing task (Fig 2.8(d)(i)). Selection of a particular group further explores and all the candidate characters are spread onto eight buttons (Fig. 2.8(d)(ii)). As there are only a few number of buttons in the keyboard, it allows to use increased button size which enable easy selection even with inaccurate pointing devices.

Miniotas et al. [145] developed an on-screen keyboard which also occupies less screen space (Fig. 2.8(b)). In this keyboard, one symbol button represents multiple characters. The leftmost key is a spacebar. It can also be used to enter full-stops and commas. The rightmost key is a functional key used to switch between keyboards containing *Function*, *Letters*, *Numbers*, and *Signs*. All keys in the interface are aligned in a single row at the bottom of the screen. To select a letter, the user first has to gaze at the key containing that letter. After the gaze has been kept steadily on the key for a certain amount of time say 400 ms, the first letter contained by the key is highlighted. If this is the desired letter for entry, the user is to confirm its selection by simply shifting their gaze towards the key containing the letter to be entered next. However, if the user's gaze, does not leave the current key's area upon highlighting of the first letter, the second letter on that key is highlighted after another 400 ms. Once again, the user has two options here, simply hold their gaze further on the key to proceed to the next letter, or shift their gaze away if the current letter is the required one. The software interprets the event of shifting the gaze away as the command to enter the letter highlighted last. That letter then appears in the text entry field.

A way of selecting interface objects is by looking at it followed by verifying the

selection through moving the eyes toward another area of the interface. This principle is followed in *Iwrite* interface (Fig. 2.8(e) [208]). There, screen buttons are placed at outer frame which drives the selections by gazing to the outer area of the interface. The design also places text window at middle of the interface. The positioning of characters in the interface reduces *midus touch* error (wrong selection of characters which are placed at neighborhood of the intended character) in eye typing.

Zheng et al. [239] designed a user interface with three levels of layering support which was called as 'Assisted Visual Interactive Notepad' (AVIN) (Fig. 2.8(f)). The first layer (i.e. the outer ring) shows the English alphabets, arranged alphabetically in a clock-wise fashion starting from the upper left-hand corner. The second layer (i.e. the inner ring) is a set of "frequently used" words which are updated based on currently composed word chunk. The third layer of the on-screen keyboard comprises a central region, which is the area where the typed characters will appear (referred to as "text box").

Pie menu concept has also been applied in designing the eye typing interface (Fig. 2.8(g)). The circular interface is divided into six slices where five characters are grouped together in a slice. To select a character, user needs to hover the eye on the group slice which further creates an enlarged circle having one character in a single slice, followed by hovering on the specific character. This method does not require any dwell time while performing eye typing task [208].

2.3.2 Minimizing dwell time

One of the challenges in eye typing task is to deal with the speed-accuracy trade-off. A long dwell time is good for preventing wrong selections but a long fixation on target can be tiring for eyes. In contrast, shorter dwell time enhances the chance of *midas touch* problem. The dwell time also sets a limit for the maximum typing speed as the user has to wait during that time before each selection. Majaranta and Räihä [134] conducted gaze typing evaluations with novices using a constant and fairly long dwell time between 450 ms to 1000 ms. Špakov and Miniotas [187], and Majaranta and Räihä [134] studied automatic adjustment of dwell time. Morimoto and Amir [147] proposed a Context-Sensitive (CS) keyboard for eye typing which uses two QWERTY layouts in two different colors, blue and green. Here, the key selection process consists of two tasks: key focus activation and key selection. Activation of the key focus is accomplished by spending short dwell times ($\approx$ 150 ms). Whereas, key selection, made by switching contexts (a saccade to the other context) is completed within 450 ms. At selection, the key focused last in the previous context is selected. The center part of the screen is reserved for typed text display.

Researchers also investigated to diminish the dwell time constraint to increase eye typing rate further. Urbina and Huckoff implemented a pie-menu based eye typing interface [207] which diminishes dwell-time at the time of character selection through specific path-driven controlled eye movement (Fig. 2.8(g)). On the other hand, Kristensson and Vertanen [109] proposed key selection mechanism in the on-screen keyboard based on "gaze in the proximity" technique. In this technique, based on the current fixation location of the gaze at an instance, the nearest keys in the keyboard are selected as the possible candidates to be entered next. From these candidates, the algorithm chooses the best fit candidate character based on the language model and other grammatical as well as linguistic constraints.

2.3.3 Visual search time minimization during eye typing

In gaze-based eye typing, users require to search for the desired key before tapping which takes significant amount of time. Highlighting selected key buttons is among few proposed effective solutions. Hansen et al. [79], in *GazeTalk* eye typing interface, implemented single next probable character highlighting where frequent characters, generated through dynamic bigram model, are placed on keyboard buttons and most frequent character got highlighted (see Fig. 2.9).

2.4 Summary

In this chapter, we present a survey on text entry interfaces suitable for mouse, finger and touch based interactions in desktop as well as mobile devices and developed in English and Indic languages.

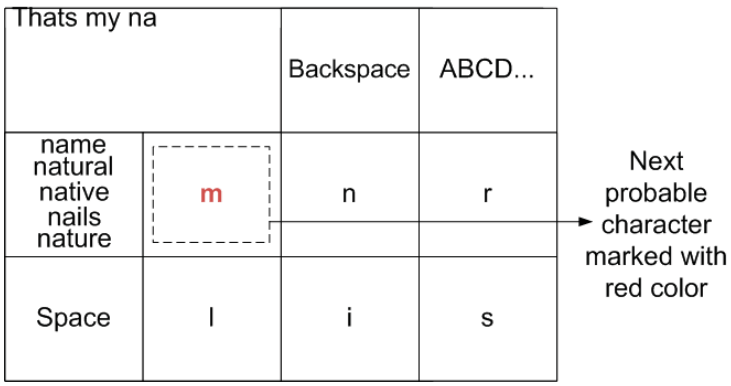


Figure 2.9: Next character highlighting in *GazeTalk* interface

From the survey on virtual keyboard design, it has been observed that, for both desktop and mobile devices, the layouts in Indic languages achieve lower text entry rates [96,97,168] compared to English virtual keyboards ($20-43$ WPM [236] for mouse-based text entry and $15-20$ WPM [127] for text entry in mobile devices). For both mouse- and touch-based text entry, size of any Indic language-based virtual keyboards becomes large as the number of characters in any Indic language is moderately high. So, mouse and eye movements over the character keys in existing keyboards increase significantly. Errors committed by users in Indic language-based text entry are much higher because of presence of conjunct characters, inflexed (*matra*) characters, graphically similar (look alike) characters etc. Moreover, a large number of possible inflexion symbols demands large area for the layout and defines a challenge to the designer to accommodate a keyboard in an area constraint display. In case of mobile devices, area constraint is even more prominent as size is much more smaller as well as key selection suffers with *fat finger* problem and finger occlusion which enhance the chance of committing errors. To accommodate many characters in a single keyboard layout, some mechanisms map more than one character in a single key which can be accessed using some special command keys like *Shift*, *Ctrl*, *Alt* etc. But this situation, in turn, increases the key presses required.

Apart from the linguistic complexity, developers have concerns, while designing a word-level prediction system for an Indic language. Without having proper normalization, searching the word written in one form will omit the words in another form [30, 31]. This, occurred with existing prediction systems, inevitably increases the miss ratio in word prediction, although alternative words are present in vocabulary. Also, the correct ordering of input sequence demands enough cognitive load on user in addition to trapping into failure of the prediction system. Overall, the above mentioned issues both from user and developer perspective, make the text entry task less accurate, more time consuming and demands more keystrokes, even with the support of word prediction utility.

The survey on existing gaze-based text entry interfaces reveals that there is a lack in addressing the trade-off between size and space of keys of on-screen keyboard in view of avoiding *midas touch* problem. Moreover, existing eye typing mechanisms of providing visual feedback, applied in some eye typing interface, fails to support the enhancement of character-based text entry speed. It may also be noted that gaze-based text entry interface development in Indic languages is yet to be reported in the literature.

Chapter 3

Virtual Keyboard Design

Virtual keyboard based text entry methods are extensively used as they are having flexibility to be designed easily for any language alphabet set. In particular, virtual keyboard based text entry in Indian languages is one of the more viable alternatives. However, Indic language text entry possess many complexities due to its inherent characteristics like presence of large character set, glyph, conjunct, halant, frequent occurrence of complex input sequence, typographic variation and presence of phonetic and graphical similar characters etc. These also affect single-tap virtual keyboard designs which support mouse and touch level text entry tasks in large and small display devices. Beside this, key arrangement in existing virtual keyboards offer more mouse movement during text entry in case of large screen devices as well as are unable to address touch level inaccuracies, for example, *fat finger* problem. In this chapter, we propose two virtual keyboard based text entry methods suitable for desktop and touch enabled mobile devices in Indic languages. For developing virtual keyboard suitable for desktop devices, we propose an optimum key arrangement mechanism in order to reduce mouse movement while composing texts. On the other hand, proposed virtual keyboard for touch enabled mobile devices provides a solution to incorporate large character set into a small display area and mitigating *fat finger* problem. The proposed keyboard design also reduces visual search time by suggesting next probable characters. The different tasks involved in our approaches are discussed in this chapter.

The rest of the chapter is organized as follows. Section 3.1 describes the design and evaluation of virtual keyboards for mouse-based text entry in Indic languages. The touch-enabled mobile device based keyboard design is presented in Section 3.2. Section 3.3 summarizes the chapter.

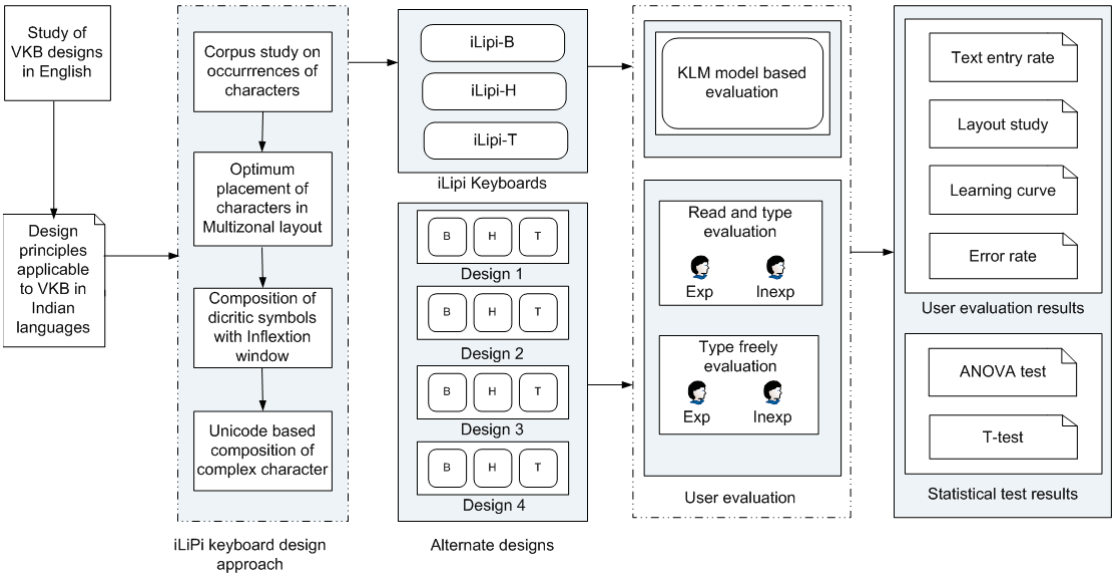
3.1 Virtual keyboard design for desktop devices

We address the issues of designing virtual keyboards for Indic languages in this section. The key idea in the proposed approach is as follows. We have studied existing design principles of virtual keyboards those are mainly meant for text entry in English and identified some principles which can be incorporated in Indic language-based keyboard design task. Most of the English virtual keyboards consider digraph probability [15] to place the characters in the layout. The same is also considered in our proposed design with a proposal of placing most frequently occurred space character at the center position. In addition to this, some special solutions which need to be incorporated in case of Indic languages have been taken into consideration in our proposed layout. Initially, a rectangular layout is chosen to accommodate five rows each having a capacity of holding 14 buttons. Then within this structure, three concentric zones have been planned. We process *Wikipedia* corpus for calculating characters' frequencies. We sort them according to their frequencies and place them at different zones such that the most frequent characters are in the inner zone, the next highly frequent characters are in the layer surrounding to the inner zone and the less frequent characters are in the outermost zone. Next, we apply GA-based optimization for each zone individually to optimize eye and mouse movement. Finally, to accommodate glyphs, we propose a hovering window approach on the consonant characters of the language. In this section, we discuss our detailed approach to design virtual keyboards in some of the Indic languages.

We call our proposed keyboard as *iLiPi*. A schematic diagram show different to evolve *iLiPi* keyboards in three representative Indian languages are shown in Fig. 3.1. The different steps are discussed in details in the following sub sections.

3.1.1 Optimum layout of keyboard

The proposed layout of virtual keyboard in Indic languages is shown in Fig. 3.2. The layout has the provision to accommodate 70 keys having each key of 35×35 $pixel^2$ (0.89×0.89 cm^2) with total area of the layout is 514×185 $pixel^2$ (13.07×4.705 cm^2) while measuring each pixel as 0.2543 mm. This is the moderate size of the keyboard we have chosen after a number of trials with the users. Being space between characters occur frequently (after completion of each word) in the corpus, we have considered a space key double the size of the other keys in the layout. This is in concurrence to the design principles followed in majority of virtual keyboards for English text entry. We propose a novel concept of multi-zonal layout. The concept of multi-zone virtual keyboard and arrangement of keys in each zone are discussed in the following.



* *iLiPi-B*, *iLiPi-H* and *iLiPi-T* denote virtual keyboards in Bengali (B), Hindi (H) and Telugu (T) languages, respectively

Figure 3.1: Work flow of *iLiPi* keyboards design approach

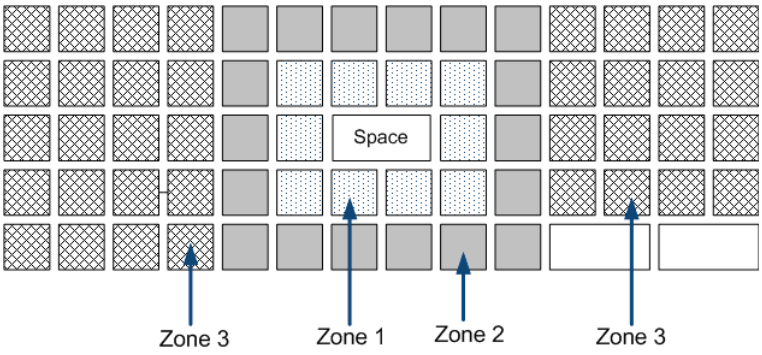


Figure 3.2: Multi-zonal layout for *iLiPi* virtual keyboard

A thorough user study reveals that users mostly focus around middle region of the interface. Taking this into account, we can reduce the visual search time, if we place the frequently occurring characters on or around the central region of the keyboard. If all frequently occurred characters are not possible to accommodate in the central region, then we may place them in multiple zones, that is, the most frequently occurring characters in the central zone, next most frequently occurring characters in the outer zone surrounding to center zone and so on. Initially, we set the space character at the middle of the central zone (as it has been found that space is the highest probable character among any Indic language alphabets) following most frequently occurred characters around it

and other less probable characters can be spatially arranged in other zones depending on their frequencies of occurrences [182]. Note that such arrangement is possible only if the distribution of frequency of occurrences of characters conforming to the arrangement is found out for a given target language, let it be *IL*. To verify this, we have to find out the frequency of occurrence of each character for the language *IL*. The same can be obtained from a (large) corpus of *IL*. From such a corpus analysis, we classify all characters in *IL* into three groups: character containing high, medium and low frequencies of occurrences. As a case study, Fig. 3.3 presents frequencies of occurrences of characters except space in Bengali processed from *Wikipedia* corpus of Bengali pages. Based on the distribution, we assign characters containing probability on or above 0.04013176 into high probable zone surrounding the space character (Zone 1) in the layout. Similarly, the characters whose probability values lie between 0.02873426 and 0.01152528 are considered to be the constituent of Zone 2. The rest of the characters would be placed into low valued zone called Zone 3.

Next, we intend to arrange characters in each zone in the layout so that it involves minimum mouse movement time (like between two composed words, space is required. After giving space, there is a huge probability of getting next word from the characters placed in the same zone around it which results minimal mouse movement). For further fine-tuning of the key arrangement, we prioritize the positions around characters (the positions around a character are filled up with others having optimum probability as well as distance from the given character). We assume that each key contains a single character. This problem thus turns out to be an optimization problem, that is, searching

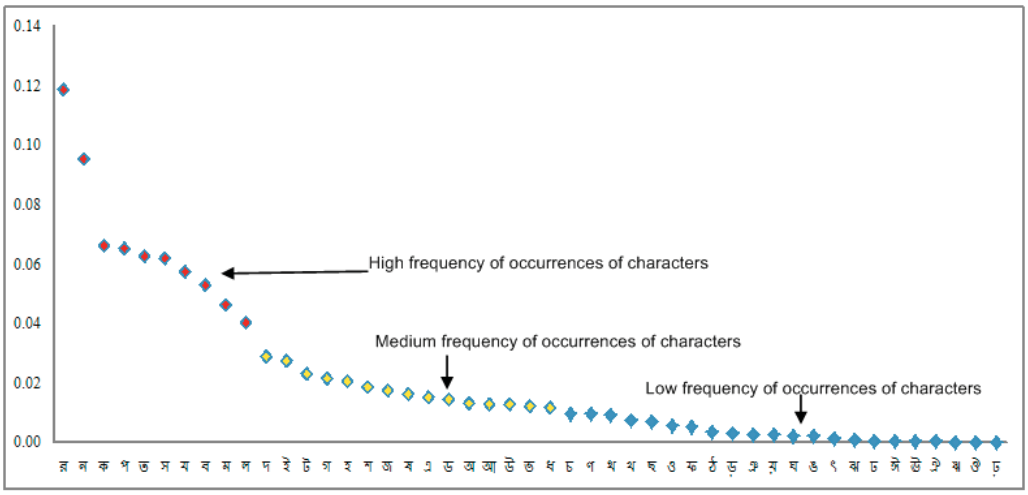


Figure 3.3: Probability of occurrences of characters in Bengali language

an arrangement of characters on keys with respect to a minimum number of mouse movements. We apply a genetic algorithm-based method to find an optimal arrangement of characters on keys. Our approach to genetic algorithm-based solution is described below.

The problem is to find an arrangement of characters in three zones in order to enable minimum mouse movement time while composing texts. In this optimization problem, we consider the average mouse movement times require to compose any text as the objective function. To calculate the mouse movement time, we use Fitts' law [58] which is defined in Equation 3.1.

$$MT_{ij} = a + b \log_2(\frac{D_{ij}}{W_j} + 1) \tag{3.1}$$

Here, MT_{ij} is the movement time from an i-th key to a j-th key, a and b are two constants. The values of a and b are chosen empirically as 0.2 and 0.1, respectively [129]. D_{ij} denotes the Euclidean distance between the two keys i and j and W_j is the size of the j-th key with respect to the i-th key. The value of the mean movement time ($\overline{MT}$) can be calculated using Fitts'-digraph model [182] as stated in Equation 3.2.

$$\overline{MT} = \sum_{i=1}^{N} \sum_{j=1}^{N} MT_{ij} \times P_{ij} \tag{3.2}$$

Here, P_{ij} is the digraph probability of occurrence of j-th character after i-th character and N denotes the total number of characters in the keyboard. The P_{ij} values for any two pair of characters can be calculated from the corpus of the language IL.

We follow the real coded ordered GA [232] to solve the above mentioned problem. We define a chromosome as three sets of characters present in three zones. The chromosome can be defined as

$$chromosome = [C_{Z_1}|C_{Z_2}|C_{Z_3}]$$

where C_{Z_1}, C_{Z_2} and C_{Z_3} represent the sets of characters in Zone 1, 2 and 3, respectively.

We decide three sets of characters in three zones based on their frequencies of occurrences as discussed above in this section. Note that the length of the chromosome is same as the total number of characters in IL. To decide an initial candidate, we choose a random arrangement of characters in their respective zones. We decide 50 such random arrangements for each candidate and consider as the initial population in our work.

After generating the initial population, we calculate the cost values based on the objective function (Eqn. 3.2) for each candidate and arrange them in ascending order.

We consider rank-based selection method [158], that is, we choose 50% of the current population with higher rank values for mating pools to generate offsprings in the next generation. We divide the candidate parents into two groups: one group contains 13 individuals with higher cost values and the rest contains in another group. We randomly take two parents from the two different groups and perform mating between them.

We follow the *substring* crossover technique [84] which prefers that a part of the first parent should be copied in the offspring and the rest should be taken in the same order as they appear in second parent. To be more specific, let us consider any two chromosomes A and B corresponding to two candidate parents in the current population. We consider P_1, P_2, P_3 and P_4 as the crossover points in the parts of two chromosomes (corresponding to a zone of characters). Our crossover mechanism is to copy a part of the length $|P_2 - P_1|$ from chromosome A and paste it into the child chromosome C in between P_3 and P_4 both inclusive. For the rest of the parts in chromosome C, we fill with characters in the order as they appear in the parent chromosome B. We illustrate the proposed crossover in Fig. 3.4. We may note that positions P_1, P_2, P_3 and P_4 are to be chosen randomly such that $|P_2 - P_1| = |P_4 - P_3|$. The same would be repeated for all three parts to complete the crossover for an offspring.

Repeating the above procedure, we construct offsprings of strength 50, the population size we have decided in each iteration. Next, we consider a mutation operation which consists of swapping the locations of two characters with respect to a zone. We have carried out T number of mutations at random (T is a number between 1 and 10 selected randomly) in the resulting chromosomes.

After the mutation operation is performed, we check whether the convergence is met or not. The convergence criterion is that the mean movement time ($\overline{MT}$) difference between two individuals should be less than 0.01. If the convergence criterion is not satisfied, we move to the next generation. Otherwise, candidate corresponding to the highest rank value is selected as a solution.

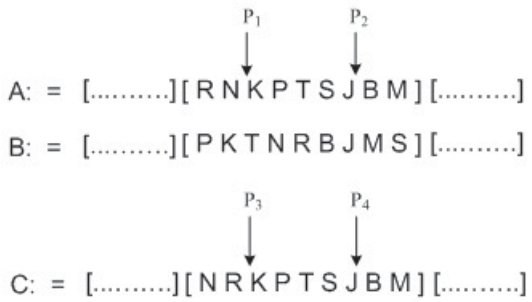


Figure 3.4: Crossover operation

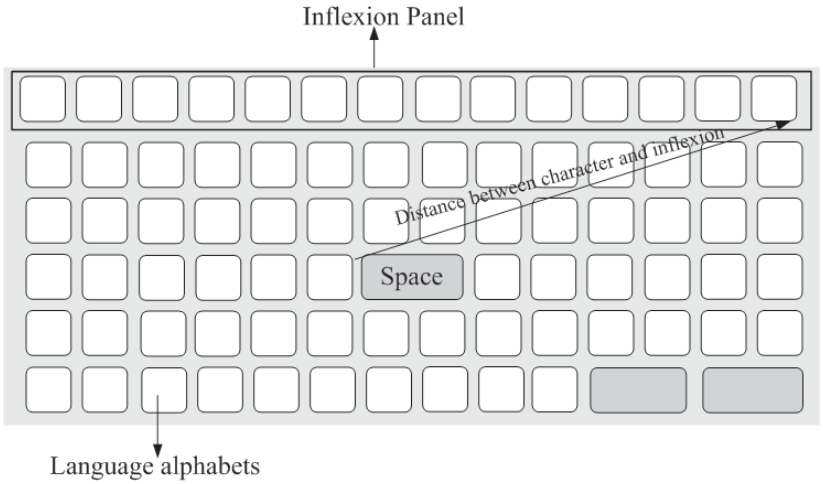
3.1.2 Inflexion window

In the optimal arrangement of characters in our proposed multi-zonal layout, we consider the arrangement of base characters in *IL*, that is, independent vowels and consonants only. Indic languages allow a large number of glyphs (inflexed characters) which can be composed with consonants and dependent vowels [223]. Occurrences of such glyphs, in fact, is very frequent. In addition to this, there are constraints of the size of the layout by positioning of a separate panel for dependent vowels. Addressing these limitations, our proposal is to hover the window containing dependent vowels (inflexion window) on single click over the characters (only consonant as it only holds dependent vowels) to compose glyphs. Our approach can be explained with an example. Let us consider a part of Hindi virtual keyboard interface as shown in Fig. 3.5(a). While user brings mouse pointer on a key say च [c], the inflexion window appears (see Fig. 3.5(b)) automatically which contains all the glyphs possible with the base character च [c] like चा [ca], चि [ci], चे [ce], चौ [cou] etc. After the appearance of dependent vowel (inflexion) window, user can tap the base character or a glyph depending on which s/he wants to compose a single character or character with dependent vowel.

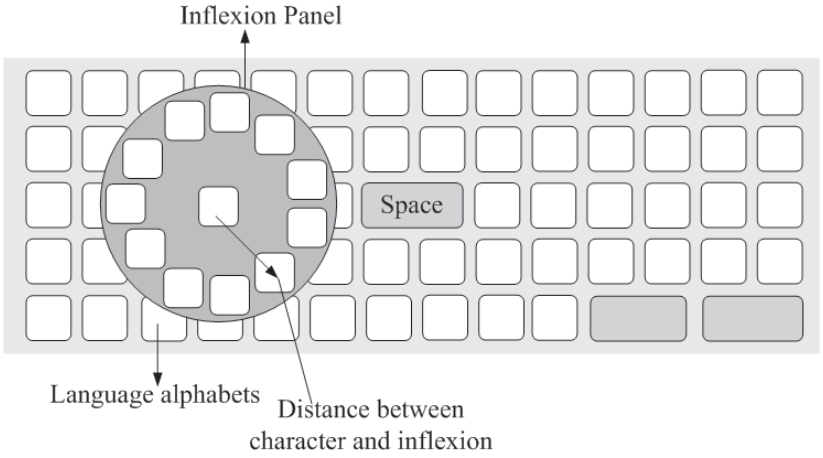
The dependent vowel (inflexion) window can be placed at any location in the keyboard. We place the window judiciously hovering on the character itself to minimize the mouse movement while typing. The statement can be justified with an example. Suppose, we place the dependent vowels in a separate panel outside the main design area of the keyboard. Figure 3.6 can explain the scenario that the placement spends more mouse movement than our hovering concept while composing glyphs. Figure 3.6(a) depicts the distance between the middlemost character of the keyboard and the rightmost dependent vowel symbol placed at the separate window. On the other hand, the situation

(a) Hindi virtual keyboard (a part) **(b)** Keyboard with inflexion window

Figure 3.5: Hindi virtual keyboard with and without inflexion window



(a) Virtual keyboard with separate inflexion panel



(b) Virtual keyboard with proposed inflexion window

Figure 3.6: Virtual keyboards with different placements of inflexion panel

explained in Fig. 3.6(b) clearly shows that hovering of dependent vowel (inflexion) panel reduces the distance between two characters hugely.

3.1.3 Composing conjunct characters

Ample occurrence of glyph and conjunct characters makes the text composition scenario in Indic languages more critical [128]. In support to the statement, literature clarifies that 1296 possible symbols can be evolved by collating the 36 fundamental symbols of Devanagari [222] in which 200 may be used (176 bi-consonantal (combinations of two consonants) and 24 tri-consonantal (combination of three consonants) conjunct

characters [149]). Further, after analyzing Bengali language corpus, it has been observed that it has 135 valid conjunct characters present in the language [181]. In order to facilitate users in composing conjunct characters through virtual keyboards, one simple and naive approach could be accommodating all possible combination of characters within the keyboard itself. But due to the presence of a huge number of characters, we can not place them in the composition interface. So, our objective is to provide an effective interface through which user can easily type conjunct characters.

In any Indic language, conjunct characters of different kinds which can be composed of pressing three or more characters. To form a conjunct character, special symbol halant/hasant (◌্) has been planned in Unicode-based diacritics. In other words, the special symbol ◌্ facilitates in combining bi-consonantal and tri-consonantal characters. Composing conjunct characters in Indic languages with our proposed approach is presented below.

3.1.3.1 Units with two, three or more consonants

A bi-consonantal unit with two consonants say C_1 and C_2 can be formed with three keystroke in the order C_1 + ◌্ + C_2. Similarly, a tri-consonantal unit with three consonants, say C_1, C_2 and C_3 can be composed as C_1 + ◌্ + C_2 + ◌্ + C_3. Examples: the word ব্রহ্ম [brohmo], where the unit হ্ম [HMo] can be composed by selecting two consonants and a halant (◌্ [.h]) between them such as (হ [H] + ◌্ [.h] + ম [m]). In another case, a conjunct character unit ক্ষ [KShno] containing three consonants and two halants can be composed as ক [k] + ◌্ [.h]+ ষ [Sh] + ◌্ [.h] + ণ [N]. On the other side, we can consider formation of conjunct characters augmented with dependent vowels, like বিষ্ণু [biShnu], where composition pays more selection of characters (ব [b] + ি [i] + ষ [Sh] + ◌্ [.h] + ণ [N] + ◌্ [.h] + ু [u]). Similar types of construction can be achieved in Hindi and Telugu languages.

3.1.3.2 Units with special signs and symbols

Apart from bi-consonantal and tri-consonantal character composition, characters can be formed with the help of various signs and symbols. For example, in Bengali language, there are several such signs exist (like ◌ঁ ◌ং ◌ঃ ◌ঁ ◌্য etc.) which can be added with the base characters to form conjunct characters. Similarly, in Hindi, a variety of special symbols and signs are used in forming conjunct characters in Hindi language which can be easily formed through our proposed virtual keyboard interface. Examples are like forming the word माँ [ma] (मा [ma] + ◌ँ [chandrabindu]) requires selection of ◌ँ [chandrabindu]

after composing the glyph मा [ma]), the character ज़ (ज [j] augmented with *nukta* (़) (U09DC) [25]) etc. Thus, the word हिंदी [Hindi] can be composed as हि [hi] + ं [bindi] + दी [di], that is, selecting हि [hi] followed by ं [bindi] symbol from the keyboard and so on. Similar approach is proposed to compose conjunct characters in Bengali and Telugu languages.

3.1.4 Experiments and experimental results

In order to validate the improvement and to prove the effectiveness of the proposed approach, we have conducted a number of experiments with three Indic languages namely Hindi, Bengali and Telugu. In this section, we discuss our experiments and the results obtained.

3.1.4.1 Virtual keyboards in Indic languages

Based on the proposed approach, we have designed three keyboards *iLiPi-B*, *iLiPi-H* and *iLiPi-T* for text composition in Bengali, Hindi and Telugu languages, respectively. The layouts of these three keyboards are shown in Fig. 3.7. Bengali virtual keyboard (*iLiPi-B*) consists of 12 vowels, 35 consonants, 11 dependent vowels is shown in Fig. 3.7(a). Virtual keyboard in Hindi (*iLiPi-H*) contains 12 vowels, 33 consonants, 11 dependent vowels (Fig. 3.7(b)). *iLiPi-T* virtual keyboard in Telugu consists of 15 vowels, 33 consonants and 14 dependent vowels (Fig. 3.7(c)).

3.1.4.2 Text composition in experiments

A user can compose a text in any of the following methods.

- *Read and type* (read a part of text and then type the text)
- *Listen and type* (listen a part uttered by assistant and type the text)
- *Type freely* (type a text of user's own)

In *Read and type*-based evaluation method, composition of text requires extra effort of shifting focus from reading to typing and back which is an addition to the eye and mouse movement times. In *Listen and type*-based evaluation, users first need to listen the text word by word uttered by instructor and then type those words. This method inducts many errors such as phonetic, transcription, sleeping errors [75, 226] etc. On the other hand, in free text composition (*Type freely*) method, users are asked to type their own text which does not require shifting focus or listening effort and also less erroneous.

(a) Bengali keyboard: iLiPi-B

(b) Hindi keyboard: iLiPi-H

(c) Telugu keyboard: iLiPi-T

Figure 3.7: Proposed iLiPi virtual keyboards in Bengali, Hindi and Telugu

We follow the third strategy in our user evaluation. We consider *form fill in* method [177] and ask our user to fill a prescribed form. The form we have used in our experiment consists of 25 entries (like name, address, occupations, hobbies etc.) and approximately needs to type 94 words and 480 characters.

After analyzing the collected user evaluation results from the log file, we comprehend that users did not select some characters in the language as those characters did not appear in their typed texts. That problem hinders us to decide the text entry rate in

a optimal way. To overcome the sub-optimality of the result, we also followed the *Read and type* method in which texts are decided so that all language characters present in the texts with uniform probability. Here, we consider a number of texts for performing *Read and type* method by users. We carefully choose the texts from existing literature in three languages in such a way that each of them is enriched with thesaurus of huge word variation of glyph and conjunct characters. We also take care that texts include almost all characters in their corresponding languages. For each languages, we select three different texts. The characteristics of the selected texts are shown in Table 3.1.

3.1.4.3 Participants

Total 48 participants participated in our experiments. Out of them 18, 15 and 15 participants are for testing virtual keyboards in Bengali, Hindi and Telugu language, respectively. Details of the participants are summarized in Table 3.10. In our experiments, we choose the participants with different educational background and computer proficiency. Their educational background varies from secondary level to post graduate level education. Thus, all participants are language literate. Participants' computer proficiency are also within a wide range, which can be classified at three different levels: (a) having at least five years of experience with computer, know operating systems, programming and application software (Level 1), (b) having $1-3$ years experience and mainly familiar with application software such as Microsoft Word, email and Internet browsing (Level 2) (c) having less than one year experience and poor in computer related tasks such as document preparation, email messaging etc. (Level 3). We categorize such a variety of participants in two categories: "*Experienced*"

Table 3.1: Text selection

Language	Text under test	Reference text	Number of words	Number of conjunct characters	Number of glyphs
Bengali	B_1	*Kapalkundala* by Bankim Chandra	153	23	304
	B_2	*Ramayana* in Bengali	163	53	352
	B_3	*Ashamanjababur Kukur* by Satyajit Ray	171	25	327
Hindi	H_1	*Shatranj Ki Khiladi* by Munsi Premchand	175	27	325
	H_2	*Nadi Bahati Thi* by Rajkamal Chaudhary	162	43	319
	H_3	*Prabandha-Parichaya* by Suryakant Tripathi Nirala	168	38	326
Telugu	T_1	*Bhagavatha Purana* by Bammera Pothana	103	36	313
	T_2	*Vishwambhara* by C. Narayanareddy	111	29	341
	T_3	*Kanyaasulkamu* by Gurajada Apparao	108	45	337

Table 3.2: Description of participants

Participant's profile				Number of participants		
Occupation	Age group	Education	Computer proficiency	Bengali	Hindi	Telugu
Student	19-27	Under Graduate	Level 1	2	2	1
		Higher Secondary	Level 2	1	1	2
		Secondary	Level 2	1	2	2
Office staff member	31-48	Graduate	Level 1	2	2	1
		Higher Secondary	Level 2	1	1	1
Business person	34-53	Post Graduate	Level 1	1	2	1
		Graduate	Level 2	1	2	2
		Higher Secondary	Level 3	1	1	1
Aged person	57-66	Graduate	Level 2	1	1	1
		Under Graduate	Level 3	1	1	1
Housewife	35-45	Graduate	Level 2	1	1	1
		Secondary	Level 3	2	2	1

and "*Inexperienced*". This categorization is based on both educational qualification and computer proficiency level. In other words, participants with level 1 computer proficiency are in "*Experienced*" category whereas those are with level 3 are in "*Inexperienced*" category. Further, a participant with level 2 computer proficiency and higher educational background (above higher secondary) belongs to "*Experienced*" category, otherwise, s/he is in "*Inexperienced*" category. There are 15 female and 33 male participants and the average age is 29.63 years (SD=4.82). Out of these 48 participants, 21 participants are familiar with typing text with mobile keypad, virtual keyboards in English or Indic languages. They belong to "*Experienced*" category. The rest of the participants are "*Inexperienced*", they are not familiar with any text entry system and new to composition of text in any language.

3.1.4.4 Apparatus

All experiments have been conducted using 2.0 GHz - 2.6 GHz Pentium Core2Duo machine with HP w17e 17-inch wide screen LCD color monitor. The screen resolution is fixed to 1440×900. The developed keyboard interfaces for the experiments is written in C# using Visual Studio 2008 which can be accessed through mouse or single pointer-based touch screen input. The key press events are recorded automatically and stored in a log file. For tracking the mouse positions, we develop a separate window hook program written in C#. All experiments are done either in Windows XP or in Windows 7 environment.

3.1.4.5 Evaluation procedure

The evaluation of the proposed layouts in three Indic languages is then carried out with the experimental setup described above. There are two ways through which our evaluation process has been carried out: *user evaluation* and *model-based evaluation* [100]. The two evaluation procedures, parameters and metrics followed are stated in the following.

User evaluation: In user evaluation method, we consider two ways for typing text namely *Read and type* and *Type freely*. Altogether nine benchmark texts have been selected from popular literatures of three languages for test. We select one short story, one novel and one translated epic text in Bengali, one novel, one prose and one story in Hindi and one novel and two stories in Telugu (also see Table 3.1). These testing texts are not a subset of training *Wikipedia* corpus. We conduct *Read and type* based user trials in two ways. In the first way, we ask users to type as quickly and accurately as possible by seeing the randomly selected printed text supplied to them. In the other way, texts to be composed are displayed to participants on top of the text composition area in the interface. The above mentioned two ways of sessions are chosen randomly for a user. During experiments, distance of the computer screen from the participant remains around 65 centimeters. The testing corpus (Table 3.1) and the keyboard for typing are chosen randomly in each session to minimize the memorizing effect of users. Each study lasts approximately 45 to 60 minutes. On an average, a user spends time in performing one to four experiments per day. Most of the users have performed text entry with all the keyboards and keyboard use conditions in their mother languages, only a few have not completed all the experiments successfully. A user, who has evaluated all keyboards in different conditions, approximately took two to six months to finish. Such a long duration is required to evaluate 60 (ten keyboards $\times$ two ways $\times$ three texts) trials per user. For both *Read and type* and *Type freely* methods, a session gap has been followed which is between one to three days. Random ordering has been followed for selecting a keyboard in user evaluation. Each user session is automatically logged into a file. After completion of a session, the session log file is analyzed to calculate text entry rate.

We also follow the *Form fill in* strategy to calculate text entry rate through user evaluation. In this method, we prepare three forms F_{H_1}, F_{H_2}, F_{H_3} in Hindi, F_{B_1}, F_{B_2}, F_{B_3} in Bengali and F_{T_1}, F_{T_2}, F_{T_3} in Telugu where the user has to fill up personal, professional and other details of their own. Any abbreviation is not allowed to fill the fields like home or work place address in the form. We also instruct users to fill up the fields with nominal number of characters (like while typing address of correspondence field, users should finish typing within 20 - 25 words etc.). Same as the previous experiment, in

Form fill in method [177], we record all activities of an entire session in a file and then analyze them.

Let T be the final transcribed or typed string entered by the user and $|T|$ is the length of this string that is the number of characters entered. Let S denotes the time taken by a user in seconds, measured from the entry of the first character to the entry of the last, including backspaces. We define the text entry rate WPM [128] as shown in Eqn 3.3. Here, $\overline{w}$ denotes average length of words for a language. It has been estimated that average word lengths are 5.11 in Bengali, 4.695 in Hindi and 7.76 in Telugu language [13].

$$WPM = \frac{|T| - 1}{S} \times \frac{60}{\overline{w}} \tag{3.3}$$

The -1 in the numerator in Eqn. 3.3 signifies that the time S is measured from the entry of the first character to the entry of the last character (i.e. the total text length $|T| - 1$) [128].

As error correction is not fully functional, for the *Read and type*-based text composition, each user transcribed word is checked with the word present in the corpus and number of errors committed by user has been calculated. At the same time, a little modified procedure has been followed to understand different categories of errors in *Type freely* based text entry. There, after composing texts through form filling method by each user at their own, the composed or transcribed texts are getting verified by developers carefully in presence of user (as users can compose nouns in terms of name of places, persons etc. where they think that the spelling of the inputted word(s) are correct which is not likely) to find out the errors in those. Further, the inexperienced people may commit mistake in spelling of composed words. The scenario, specifically occurred for *Form fill-in* method-based text composition, can be tackled by a mechanism where firstly, hard copy of the form is to be given to users for filling up and after completion, the form has been digitized and users are permitted to fill up the same form through virtual keyboard. Thus, after this session, we are able to measure the similarity between the characters written as well as typed (in this case, we consider the written text as standard). Despite of being more time consuming, the procedure seems to be effective to cater the anomalies in users' own transcribed text. Moreover, as Indic language texts consist of conjuncts ($y$uktakshar) or glyphs occurring frequently and users commit mistakes most of the time to compose the same, character level error analysis is inevitable. To compare the procedure and extend the error analysis parameters of English, after completion of each user transcription session, a preprocessing of the composed texts is to be needed where every simple and compound words can be broken into atomic character level (like

for গল্প [golpo] in Bengali language, it is to be broken into constituent character, halant and special symbols which need to be tapped in Bengali virtual keyboard i.e. গ [ga] + ল [la] + ্ [.h] + প [pa]). For error analysis, we apply *Minimum String Distance* (MSD) and *Key Stroke Per Character* (KSPC) metrics [184]. This was decided so that we analyze errors for user evaluation but not for system simulation as at the time of text composing, user may commit mistakes, but system can not.

User composed input stream through virtual keyboard can be considered as transcribed text. The input stream consists of keystrokes (some correct, some erroneous) and editing strokes (backspace, delete, cursor movements, etc.) from the keyboard. These keystrokes of the input stream can be divided into four classes, depending on their influence on the error rate. Correct keystrokes can be stated as the *Correct* (C). Errors which are not noticed by the user during composition, remain into the transcribed text which can be termed as *Incorrect and Not Fixed* (INF). Together, these C and INF keystrokes comprise all of the characters in the transcribed text. The errors corrected characters are grouped in the *Incorrect but Fixed* (IF) category. Similarly, the keystrokes performing the corrections are considered as the *Fix* (F). Given the presented text, input stream, and transcribed text, it is easy to classify keystrokes.

C and *INF*: It represents the characters in the transcribed text which belong to the corrected or not corrected classes, respectively. The *C* keystrokes are the correct characters in the transcribed text and *INF* contain wrongly entered keystrokes present in final transcribed stream.

F: The keystrokes belonging to the *F* class are mainly editing functions like backspace, delete, cursor movement as well as modifier keys (shift, alt, and control) etc.

IF: The *IF* keystrokes are those which are in the input stream, but not present in the transcribed text. Also editing keys are exempted from this class.

For classifying the particular characters in each group, a more detailed character-by-character analysis of errors is required [183]. However, in computing error rates, the particular characters are not useful in calculation. Rather, the size of the classes is important. Therefore, for convenience, let C, INF, IF, and F denote the number of keystrokes in each of their respective classes. The MSD error rate and KSPC statistic then can be defined in Eqn 3.4 and 3.5, respectively [184].

$$MSDErrorRate = \frac{INF}{C + INF} \times 100\% \tag{3.4}$$

$$KSPC \approx \frac{C + INF + IF + F}{C + INF} \tag{3.5}$$

Consider the following example for further classification of the classes stated above.

Presented text: the quick brown
Input stream: th qui**x**←ck br**p**own
Transcribed text: th quick br**p**own

In this example, there are three errors: an 'e' is omitted. There is an extra 'x' which further corrected with a backspace. Also, there exists an extra 'p' that remains uncorrected. The keystrokes are mapped into the several class depicted in Figure 3.8.

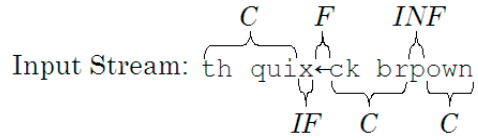


Figure 3.8: Classifying the keystrokes in an example

The example itself contains wrong information where a detail framed to be missing in Figure 3.8 which is the INF class also includes the missing 'e' keystroke. In this example, C = 14, INF = 2 (counting the extra 'p', and the missing 'e'), IF = 1, and F = 1.

Model-based evaluation: We use Keystroke Level Model (KLM) [101] along with Fitts'-digraph (FD) model [182] in our model-based evaluation. The parameters and metrics considered in our model-based evaluation is summarized as follows. The KLM is used to evaluate a user interface corresponding to the task completion time so far the lower level tasks are concerned. In addition to this, the KLM also takes care about cognitive load. The different lower level tasks, cognitive load and their completion times according to the KLM are summarized in Table 3.3. So far the evaluation of virtual keyboard interface is concerned, the KLM can be applied to evaluate mouse related task completion time plus the cognitive load. However, there is a modification that needs to be performed to evaluate mouse movement time properly in its present scope. This time would account for the time required for mouse movement from a currently selected key to the next key. To measure this component, we propose to use Fitts'-digraph model [182] (see Eqn. 3.2).

Thus, the time for typing a single character $T_{compose}$ can be evaluated by measuring the sum of the times taken to perform three important subtasks individually: mentally preparing for tapping next character (T_M) (which involves searching for the character in the keyboard by visual perception through eyes and finding the decision through cognition in human brain), moving or pointing (T_P) of the mouse pointer towards the newly decided characters ($\overline{MT}$) calculated using Fitts'-digraph model, and finally clicking on the character button by mouse (T_{BB}). In KLM model, it has been pointed out that

the system response operator (T_R) value depends on the performance of the system where the model is to be deployed. In our case, we take T_R as 0.1 second [101]. The tasks with their predicted values are mentioned in Table 3.3.

$$T_{compose} = T_{BB} + T_M + T_R + T_P(\overline{MT}) \tag{3.6}$$

In the calculation of $\overline{MT}$ (Eqn. 3.2), the values of a and b are decided empirically as 0.2 and 0.1, respectively [129]. Width W_j of all target keys are taken as one except *Space*, *Clear* and *Back*. For these three keys we take W_j as two. The digraph probability P_{ij}'s are calculated following the Wikipedia corpus of target *IL*. Thus, using KLM and Fitts'-digraph model, we can estimate the average time to type a character ($T_{compose}$) as stated in Eqn. 3.7a and 3.7b, respectively.

$$CPS = \frac{1}{T_{compose}} \tag{3.7a}$$

$$WPM = CPS \times (\frac{60}{\overline{w}}) \tag{3.7b}$$

Here, $\overline{w}$ denotes that the average word length of the language *IL*.

3.1.4.6 Designs in evaluation

Existing virtual keyboards in Indic languages like *iLeap* [196], *gate2home* [61], *LooKeys* [125], *Lipik* [124], *MyLanguage* [150], *Google* [69], *Guruji* [76], *Avro* [159] are considered for the experiment. In window-based system, Microsoft offers an on-screen keyboard in Indic languages which is also taken into account. On the other hand, keyboards developed based on the design principles of popular English keyboards

Table 3.3: Parameters in KLM

Operator type	Operator	Remarks	Time (in Second)
Physical motor operator	B BB	Pressing or releasing mouse button Clicking the mouse button	0.10 (Down or up) 0.20 (Click)
Mental operator	M	Mentally preparing for a physical action	1.35
System response operator	R	Waiting for system to respond	0.1

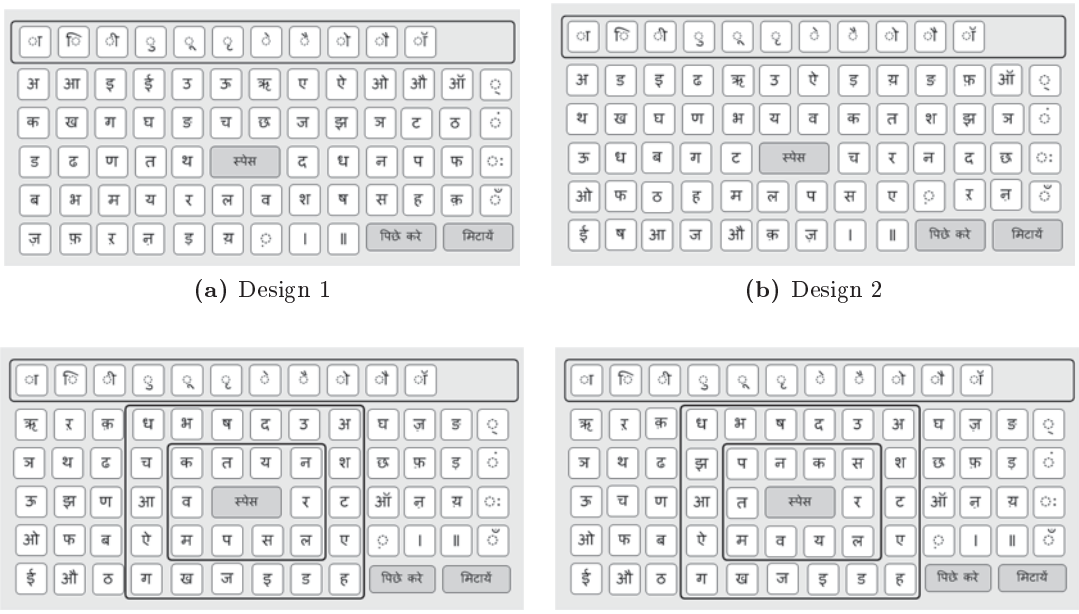
like *Dvorak*, *OPTI*, *FITALY*, *Lewis*, *Hooke*, *Metropolis*, *ATOMIK* in Hindi, Bengali and Telugu languages are also taken for experiment. Moreover, to prove the effectiveness of the proposed keyboard design, four different layouts with variating design principles have been exercised which are also accessed by users during experiments. Those alternate designs are stated as below.

Design 1 *(Layout in alphabetical order)*: This design considers alphabetical ordering of characters in the layout. The keyboard consists of distinct individual grouping of vowels, consonants and dependent vowels, and characters are spatially laid in their respective groups in alphabetical order (Fig. 3.9(a)).

Design 2 *(Layout with random arrangement of characters)*: In this design, we consider a random arrangement of characters except dependent vowels and punctuation symbols in the layout (see Fig. 3.9(b)).

Design 3 *(Layout of multiple zones)*: In this design, we arrange the characters into three different zones. The most frequently occurred characters are placed in the central zone followed by the next most frequently occurred characters in the outer zone surrounded by the central zone and so on (Fig. 3.9(c)).

Design 4 *(Multi-zonal with GA-based optimum key placement)*: This design is same as the Design 3 except that in each zone of the layout, characters are placed with optimal arrangement. The layout of Design 4 is shown in Fig. 3.9(d).



(a) Design 1 **(b)** Design 2

(c) Design 3 **(d)** Design 4

Figure 3.9: Alternate virtual keyboard design approaches in Hindi

3.1.4.7 Experimental results

The main objective of our experiments is to study the text entry rates of the existing English and Indic language virtual keyboards along with three alternate designs and proposed *iLiPi* keyboard through user as well as model-based evaluations. Also, we require to analyze layout area requirements for all the keyboards. The other targets are to analyze learning curve obtained based on longitudinal user studies on several keyboards as well as to investigate errors committed by users during text composition.

Further, to analyze the results of text entry rates getting from text typing sessions performed by users or computer programs, we have conducted an analysis of variance (ANOVA) test. The tests are conducted using *Statistical Package for the Social Sciences*, also called SPSS tool [89]. As the average word lengths of these three languages are different, the comparison may not be properly accomplished. So, to avoid the ambiguity, we take the lower valued Hindi word length as baseline and convert the other language's text entry rate into one standard by multiplying with a factor which is $\frac{4.69}{x}$, where x denotes the average word length of a target language.

Also, we consider weighted mean of text entry results in three languages (calculated in Eqn 3.8) for statistical analysis.

$$Weighted\ mean\ of\ WPM = \frac{\sum_{i=1}^{n}[WPM_i \times Average\ word\ length_i]}{\sum_{i=1}^{n} Average\ word\ length_i} \tag{3.8}$$

where i indicates the individual language.

To comprehend user acceptance more appropriately on alternate keyboard designs, we have conducted experiments with users through *Read and type*-based text composition method with selected text corpus (mentioned in Table 3.1) in Bengali, Hindi and Telugu languages. The measured mean text entry rates for those designs along with proposed *iLiPi-H* in Hindi are depicted graphically in Fig. 3.10.

From the result we see that the *Design* 2 gives 0.72% better text entry rate than *Design* 1. Similarly, *Design* 3 and *Design* 4 give 6.88% and 13.77% better text entry rates when we compare them with respect to Design 1, respectively. We note that 18.84% more text entry rate is possible with *iLiPi-H* compared to *Design* 1. Further, we note that the design concept of dependent vowel (inflexion) window (*iLiPi-H*) enables 4.46% better text entry rate compared to non-inflexion window-based design (Design 4). Similarly, Design 4 achieves 6.44% improvement in text entry rate over Design 3 and Design 3 achieves 6.12% better text entry rate than Design 2. More or less similar results have been observed with *iLiPi-B* and *iLiPi-T*.

Further, the analysis of variance on text entry speeds shows that there is a significant difference between the means of user's performance on different keyboard designs ($F_{4,35} = 18.92, p < 0.05$).

Text entry rate: We conduct both user evaluation and model-based simulation experiments. It may be noted that for each of the three languages, the inexperienced users are requested to spend sometime (may be one or two days) in composing text through virtual keyboards before starting of actual sessions to get familiarized. After accumulating the result from log files, we compare the means of text entry rates taken from several sessions performed by experienced or inexperienced users as well as automated systems of the proposed design approach with respect to some existing designs. The text entry performances in comparison with different keyboard design approaches are given below.

User evaluation: Participants have tried to compose selected texts (mentioned in Table 3.1) in three languages (Bengali, Hindi and Telugu) through Indic language-based existing virtual keyboards like *iLeap* [196], *gate2home* [61], *LooKeys* [125], *Lipik* [124], *MyLanguage* [150], *Google* [69], *Guruji* [76], *Avro* [159]. During analysis of the keyboards from their original source, we have found that some layouts are lacking in providing sufficient text area attached with the keyboard like *Google*. In that case, to increase the usability and overcome the current limitations, we have imitated the same design and

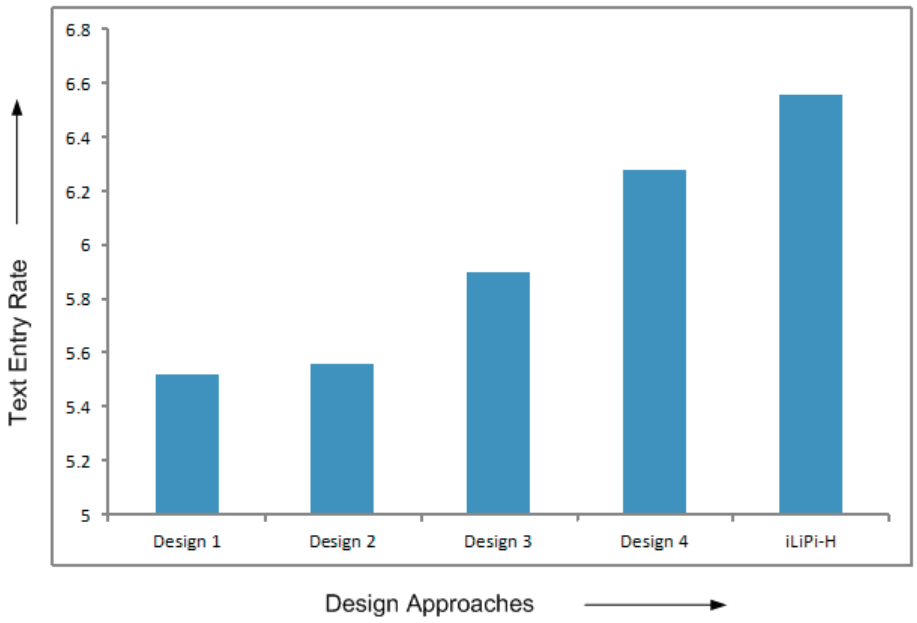


Figure 3.10: Comparison among different virtual keyboard designs

attached moderately large text area with each keyboard. The experiments on keyboards have been performed through several sessions and text entry rates are measured. In window-based system, Microsoft offers an on-screen keyboard in Indic languages which is also taken into account. The mean experimental user evaluation results on different test corpus in three Indic languages have been accumulated. The mean results of both experienced and inexperienced participants using both text composition methods (*Read and type* and *Type freely*) in three different languages are shown in Table 3.4.

Subsequent analysis of user results using *Read and type* method for typing reveals that the developed *iLiPi* design achieves more text entry rate than Google keyboard in Hindi, *Avro* keyboard in Bengali, Guruji keyboard in Telugu by 15.41%, 18.53% and 22.12% respectively for inexperienced users as well as 20.70%, 18.96%, 16.57% for experienced users. Similarly, for *Type freely*-based text entry, *iLiPi* outperforms same keyboards in three languages by 14.95%, 17.35% and 19.81% for inexperienced users side by side 19.13%, 17.05% and 18.90% in case of experienced users.

To further compare the two types of users' performance on different virtual keyboard-based text composition mechanisms, we have conducted the analysis of variance tests for Indic language and English-based virtual keyboards in three languages. While comparing the proposed *iLiPi* keyboard with existing Indic language keyboards, the analysis of variance tests (ANOVA) performed for experienced and inexperienced users on *Read and type*-based text entry reveal that the mean text entry rate of the keyboards are

Table 3.4: User evaluation result for existing Indic language-based keyboards

Language	Text entry method	Type of user	Text entry rate (in WPM)									
			iLeap	gate2home	Lookeys	Lipik	Google	Guruji	Microsoft	Avro	MyLanguage	iLiPi
Hindi	Read & Type	Inexp	4.30	4.24	4.29	5.04	5.32	5.25	4.53	-	4.34	6.14
		Exp	4.58	4.42	4.47	5.32	5.70	5.63	4.64	-	4.52	6.88
	Type Freely	Inexp	4.96	4.72	4.83	5.28	5.55	5.48	4.89	-	4.82	6.38
		Exp	5.24	5.18	5.12	5.96	6.22	6.14	5.29	-	5.19	7.41
Bengali	Read & Type	Inexp	4.03	3.90	3.94	4.40	4.64	-	4.05	4.69	3.96	5.50
		Exp	4.49	4.36	4.39	4.93	5.38	-	4.69	5.37	4.44	6.40
	Type Freely	Inexp	4.30	4.15	4.19	4.66	4.90	-	4.31	4.91	4.16	5.75
		Exp	4.84	4.61	4.65	5.19	5.61	-	4.95	5.63	4.67	6.59
Telugu	Read & Type	Inexp	2.66	2.61	2.59	2.76	3.05	3.12	2.76	-	2.63	3.81
		Exp	3.03	2.91	2.94	3.23	3.35	3.38	3.10	-	3.03	3.94
	Type Freely	Inexp	2.82	2.76	2.74	3.13	3.19	3.23	2.80	-	2.79	3.87
		Exp	3.19	3.05	3.11	3.40	3.44	3.51	3.25	-	3.19	4.09

*Inexp: Inexperience, Exp: Experience

not same ($F_{9,476} = 42.34, p < 0.05$). On the other side, while we compare the results obtained from users performance for composing text freely (*Type freely* method), we also observe that there is significant difference between performance of Indic virtual keyboards ($F_{9,476} = 37.96, p < 0.05$). The statistical test, that is analysis of variance test, has been conducted for comparing mean text entry rates of inexperienced and experienced users. The result substantiate that there is significant difference between performance of keyboards irrespective of users' proficiency (for inexperienced, $F_{9,233} = 33.29, p < 0.05$, for experienced $F_{9,233} = 22.54, p < 0.05$)

We have calculated the mean text entry rates for two kind of users on Indic language keyboards and analyzed the result through *Independent T-test*. It signifies there is a significant difference between the mean text entry rates (WPM) achieved by experienced and inexperienced users ($t = -3.12, df(18)$ and $p < 0.05$ for two tailed test). The result reflects that for a fixed type of text composition task and text to be typed, experienced users outperform people belong to inexperienced category. Similarly, we gather users' text composition results keeping other constraints fixed and analyze the tendency of data. The statistical test (*Independent T-test*) has been conducted which announces that there exists significant differences between the mean text entry rates achieved through *Read and type* and *Type freely* methods ($t = -2.122, df(18)$ and $p < 0.05$ for two tailed test). Experimental results follow the traditional fact that, for other condition unaltered, experienced users and *Type freely* method proved to be better than inexperienced users and *Read and type* method, respectively.

The keyboards based on the design principles of *Dvorak*, *OPTI*, *FITALY*, *Lewis*, *Hooke*, *Metropolis*, *ATOMIK* are adopted to design virtual keyboards in Hindi, Bengali and Telugu. We compare these designs with our designs through users' evaluation maintaining same condition and corpus. The mean user evaluation results for all users are shown in Table 3.13.

For *Read and type* method-based typing, the developed *iLiPi* design yields higher text entry rate than the existing highest text entry valued *OPTI* keyboard in Hindi, Bengali and Telugu language by 17.85%, 28.80% and 29.59% respectively for inexperienced users as well as 21.34%, 33.11% and 18.60% respectively for experienced users. Similar observations have been observed for the other method of composing text, namely *Type freely*, in Hindi, Bengali and Telugu language by 15.85%, 21.99% and 22.08% respectively for inexperienced users as well as 19.32%, 28.72% and 16.19% for experienced users.

We perform analysis of variance to compare the results of the proposed *iLiPi* keyboard with other English language-based keyboards. The test has been performed on experienced and inexperienced users' text composition rates achieved through *Read and*

type and *Type freely* methods. Result substantiates that for *Read and type* method, the mean text entry rate of 10 virtual keyboards are not same ($F_{9,530} = 45.79, p < 0.05$). We analyze the results accumulated from users performance for freely composed text and observe that there is also significant difference between performance of English virtual keyboards ($F_{9,530} = 22.87, p < 0.05$). The ANOVA test conducted to verify mean text entry rates in user base (i.e. for inexperienced and experienced users), also presents that there is significant difference between performance of keyboards irrespective of users' proficiency (for inexperienced, $F_{9,260} = 26.92, p < 0.05$, for experienced $F_{9,260} = 24.64, p < 0.05$

Also, further statistical result reveal that there are significant differences between the means of text entry rates accomplished by experienced and inexperienced users ($t = -2.31, df(18)$ and $p < 0.05$ for two tailed test) as well as *Read and type* and *Type freely*-based text composition methods ($t = -2.29, df(18)$ and $p < 0.05$ for two tailed test) keeping other constraints unchanged. The results inevitably support the normal circumstances.

Model-based evaluation: The performance of a computing system to compose text automatically can be measured through KLM model. Using the model, the system's text composition rate on previously mentioned corpus through virtual keyboards in three different languages are depicted in Table 3.6 and 3.7.

Analysis of system performed results signifies that the *iLiPi* design achieves better

Table 3.5: User evaluation result for English language-based keyboards in Indic languages

Language	Text entry method	Type of user	Text entry rate (in WPM)									
			Dvorak	Fitaly	Opti	Lewis	Hooke	Cirrin	Chubon	Metropolis	Atomik	iLiPi
Hindi	Read & Type	Inexp	5.08	5.12	5.21	4.93	4.36	4.67	4.42	4.84	4.99	6.14
		Exp	5.55	5.57	5.67	5.32	4.75	4.81	4.83	5.19	5.32	6.88
	Type Freely	Inexp	5.47	5.51	5.53	5.42	4.76	5.02	4.76	5.08	5.19	6.38
		Exp	6.06	6.13	6.21	6.08	5.37	5.56	5.89	5.95	5.93	7.41
Bengali	Read & Type	Inexp	4.03	4.23	4.27	4.14	3.84	3.92	3.86	3.91	3.99	5.50
		Exp	4.36	4.38	4.41	4.32	3.99	4.34	4.03	4.08	4.16	5.87
	Type Freely	Inexp	4.06	4.25	4.32	4.17	3.88	4.01	3.87	3.96	4.06	5.27
		Exp	4.58	4.63	4.70	4.65	4.28	4.64	4.30	4.37	4.41	6.05
Telugu	Read & Type	Inexp	2.90	2.93	2.94	2.87	2.73	2.82	2.77	2.85	2.85	3.81
		Exp	3.29	3.27	3.32	3.14	3.04	3.21	3.17	3.15	3.19	3.94
	Type Freely	Inexp	3.03	3.08	3.17	3.05	2.85	3.05	2.87	3.05	3.03	3.87
		Exp	3.43	3.45	3.52	3.38	3.26	3.40	3.28	3.31	3.37	4.09

*Inexp: Inexperience, Exp: Experience

text entry rate than Google keyboard in Hindi (21.98%), *Avro* keyboard in Bengali (21.34%) and Guruji keyboard in Telugu language (33.97%), respectively. Same observations have been achieved for English language-based keyboard for Hindi (26.22%), Bengali (28.99%) and Telugu (29.85%) languages.

Statistical tests have been conducted to scrutinize the trend of model-based evaluation results executed on different virtual keyboards. The comparison between weighted mean text entry rates carried out for existing Indic language as well as English language-based keyboards confirms significant difference between keyboards. The ANOVA test on Indic language keyboards produces significant difference ($F_{9,71} = 12.44, p < 0.05$) and English language-based keyboard performance results also follow the same trend ($F_{9,80} = 15.38, p < 0.05$)

Layout area: The keyboard occupied area for existing Indic language-based virtual keyboards, English language design principle-based keyboard layouts are shown in Table 3.8. Here we have considered the standard screen area as 1440×900 $pixel^2$ i.e. 36×22.5 cm^2.

Table 3.8 reflects the fact that the proposed *iLiPi* layout occupies on an average 514×185 $pixel^2$ area (13.07×4.705 cm^2) on the screen (36.62×22.89 cm^2) and it is 15% lower than any average English language-based layout occupancy area as well as on the average 12% less than an existing Indic language-based virtual keyboards.

Learning curve and Error proneness: We have also taken feedbacks regarding user friendliness and error proneness from the users after completion of evaluation sessions for every keyboard and analyze them thoroughly.

Learning curve: Due to the space limitation, we plot only Bengali language-based text entry performance results (for 60 consecutive sessions) of three virtual keyboards each representing one particular type of design approach. Those keyboards are: *OPTI*, *Avro* and an adhoc virtual keyboard (Design 1). We have taken the users' text composition performance results for 60 sessions spread over a duration of three weeks and *Type freely* text composition strategy has been followed.

The observed results are shown in Fig. 3.11. From Figure, it is evident that *iLiPi-B*

Table 3.6: Model-based evaluation result for existing Indic language-based keyboards

Language	Text entry rate (in WPM)									
	iLeap	gate2 home	Lookeys	Lipik	Google	Guruji	Microsoft	Avro	My Language	iLiPi
Hindi	5.55	5.53	5.54	5.55	5.96	5.70	5.56	-	5.53	7.27
Bengali	4.78	4.76	4.77	4.79	5.06	-	4.83	5.03	4.75	6.14
Telugu	2.75	2.78	2.77	2.74	3.15	3.08	2.78	-	2.76	4.22

Table 3.7: Model-based evaluation result with English keyboard design approaches

Language	Text entry rate (in WPM)									
	Dvorak	FITALY	OPTI	Lewis	Hooke	Cirrin	Chubon	Metro polis	ATOMIK	iLiPi
Hindi	5.72	5.74	5.76	5.53	5.26	5.66	5.22	5.35	5.23	7.27
Bengali	4.72	4.74	4.76	4.64	4.30	4.63	4.26	4.38	4.27	6.14
Telugu	3.18	3.19	3.25	3.09	3.01	3.06	2.97	2.99	3.06	4.22

Table 3.8: Layout area of different keyboards (*in* cm^2)

Indic language-based virtual keyboards		Existing keyboard design approaches of English applied to Indic languages	
Keyboards	Layout Area (in cm^2)	Keyboards	Layout Area (in cm^2)
iLeap	20.47×6.38	Dvorak	18.66×6.41
gate2home	14.98×5.21	FITALY	18.67×7.32
Lookeys	16.53×6.69	OPTI	19.12×7.43
Lipik	16.99×6.03	Lewis	18.36×5.72
Google	10.53×10.58	Hooke	16.43×5.52
Guruji	11.16×6.61	Cirrin	15.11×5.19
Microsoft	15.21×7.58	Chubon	15.41×6.18
Avro	14.04×5.62	Metropolis	17.24×5.98
My Language	13.53×5.24	ATOMIK	16.61×7.15
iLiPi	13.07×4.71	Single-finger	13.02×12.97

needs more initial effort to learn compared to *OPTI* and *Avro*, however, after twenty to twenty five sessions, *iLiPi-B* outperforms other two. The *iLiPi-B* layout reaches nearly 5.71 wpm by the 25^{th} session whereas the performance of the *OPTI* and *Avro* keyboard layouts achieve upto 5.23 wpm text entry speed in average.

For each layout, we derived standard regression models in the form of the power curve fitting as it follows *Power law of learning* [26]. The prediction equations and the squared correlation coefficients for the curves are illustrated in Fig. 3.11.

The high R^2 values signify that the learning curves are fitted more accurately with user behavior. In three cases over 99.3% of the variance is observed for in the models. Higher R^2 value interprets that participants are getting well accustomed with computers and they are familiar with the virtual keyboard-based text composition. The longitudinal study lasts for 60 sessions for each experienced and inexperienced users. The participants become “experienced" in context of composing text through virtual keyboards. The text typing performed with *iLiPi-B* layout lasts for several days. So, we plot the results and construct the learning curves which inevitably reflects the increasing efficiency of users after performing several sessions (see Fig. 3.11). The highest text entry rate achieved by

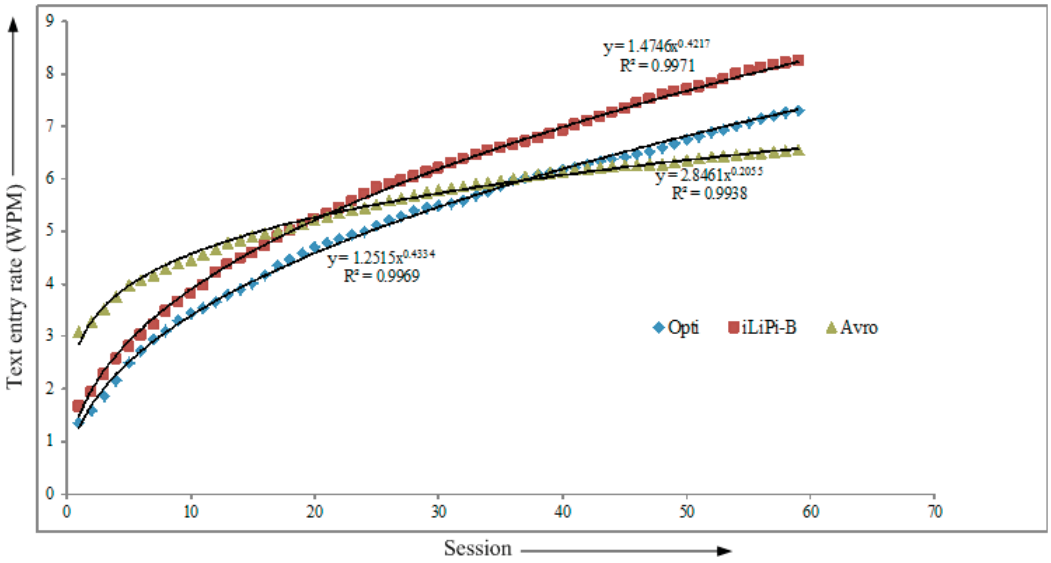


Figure 3.11: Learning curve

user on typing similar length texts for the *iLiPi-B* layout is 8.25 wpm and for the *OPTI* and *Avro* layouts, 7.31 and 6.55 wpm, respectively.

Error proneness: We have also observed the number of errors committed by users during text composition experiments. We have used the MSD/KSPC metric [183] (described in Section V(E)) for calculating error rate and considered the method sufficient toward fulfilling the requirement estimating the approximate error rate. Error corrections during the user evaluation experiments (through *Read and type* and *Type freely* tasks) are allowed. Two kinds of errors have occurred, those that are corrected and those that are not corrected. The minimum string distance (MSD) is used to count the errors which are not corrected (*INF* error). The keystrokes per character (KSPC) is used for calculating the number of keystrokes used to correct errors. If there are no extra keystrokes, KSPC is equal to one. If some errors are corrected (*IF*), KSPC is greater than one.

An analysis of variance reveals that there is no significant difference in error rates between the keyboard designs ($F_{14,269} = 19.74, p < 0.05$). Another observation shows that there is a significant difference in committing error rates between *Experienced* and *Inexperienced* users ($t = -3.27, df(18)$ and $p < 0.05$ for two tailed test).

3.2 Virtual keyboard design for mobile devices

We propose a user interface, named as *mLiPi* (text composition with **m**obile devices), which supports single-finger or thumb based text entry in touch-enabled mobile devices. There are three design tasks in our proposed approach:1) layout design of virtual keyboard, 2) augmenting character level prediction and c) implementing text entry rate enhancement strategy. Our solutions along with the rationale behind each solution are discussed in the following.

3.2.1 Designing a virtual keyboard layout

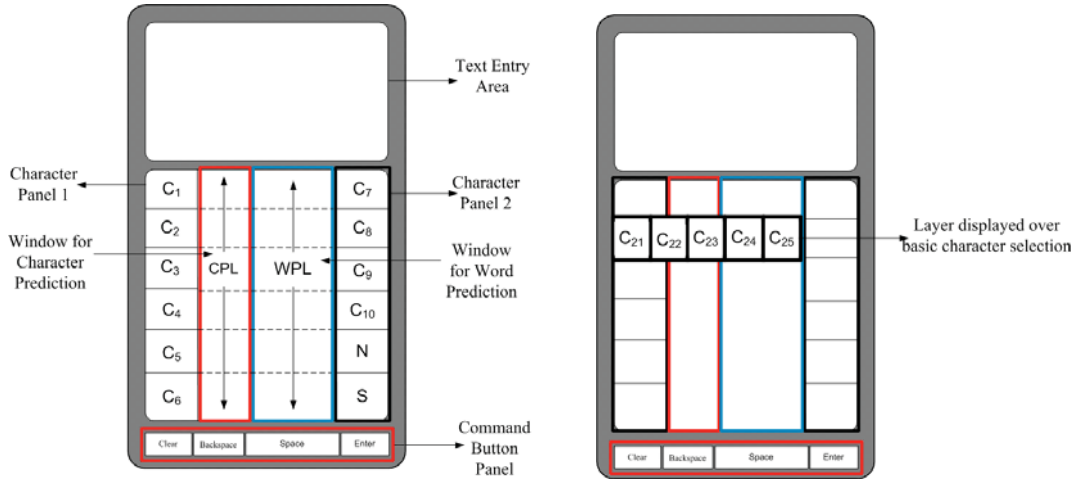
A virtual keyboard layout is developed to support text entry in touch-enabled mobile devices. The interface design paradigm consists of three principles, which are described below.

3.2.1.1 Basic layout design

A basic structure of the interface design for text composition is shown in Fig. 3.12(a). The character arrangement, in this layout, follows the characteristics of majority of the Indian languages which are categorized under *Brahmic script* [42]. In the layout, alphabetically arranged characters are grouped assigning a group leader for each (which becomes a first character of the group according to the character position in the alphabet). These characters are placed at the left side first (from top to bottom, $C_1 - C_5$) and then at the right side ($C_7 - C_9$) Except C_9, each group contains five characters, which are consonants. C_9 contains 10 characters as independent vowels. Group C_6 and C_{10} contain two special set of characters present in Indian languages under *Brahmic script*. These two groups contain 10 characters in each, which are dependent vowel (*matra*) and conjunct. Numerics and special symbols (10 in each group) in the language are represented by N and S, respectively. The layout also includes three command buttons (*Backspace*, *Clear* and *Enter*) along with space character at the bottom area of the interface. The layout also includes a word-level prediction list (*WPL*). Detailed methodology is described in Chapter 4.

3.2.1.2 Working with the basic layout

The working of the proposed layout is described below.
Selection of characters: On hovering a group leader character button, a horizontal character panel is displayed containing other characters in the group, (Fig. 3.12(b)).



(a) Keyboard layout **(b)** Layout with layering

Figure 3.12: Basic structure of interface design

When the leader character is tapped, all elements other than that panel get deactivated. Then, users can select the intended character in the group by moving the finger to the target and tapping it. Once tapped, the panels gets disappeared reactivating the whole interface. It may be noted that, if any user wants to select the leader character itself, he needs to tap the panel twice. The basic design of hovering layer and the layer developed in proposed interface are pictorially depicted in Fig. 3.12(b) where a panel containing five characters namely C_{21}, C_{22}, C_{23}, C_{24} and C_{25} gets displayed after selecting character C_2. It may be noted that using our system, user needs to spend two keystrokes.

Composition of glyph: The languages following *Brahmic script*, many combinational character called *glyph*s [96], need to be composed. This can be done with our system by combining consonant character (any character belongs to $C_1 - C_5$ and $C_7 - C_9$) with any member of the *matra* group (C_{10}).

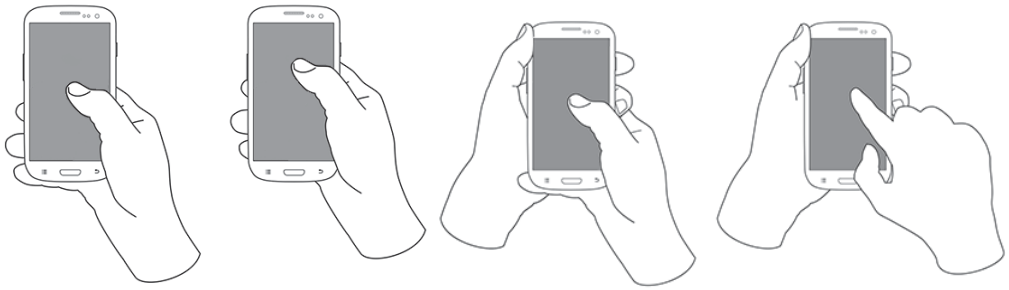
Composition of conjuncts: In *Brahmic* language scripts, joining two or more consonants creates conjuncts [96]. Conjuncts are particularly difficult to input with *mLiPi* interface; any conjunct possible in an Indic language can be composed keeping a joiner (called halant ্) between two consonants. Conjuncts are particularly difficult to input as well as takes two or more keystrokes for composing.

Command buttons: The interface accommodates three command buttons namely *Backspace*, *Clear* and *Enter* for performing string level operations while composing texts in *text entry area* of the interface (Fig. 3.12). Selecting *Backspace* button deletes the last composed character and *Clear* button deletes the whole composed text chunk from

the text box. Whereas activation of *Enter* command places the cursor into next line to allow user for paragraph breaking. Apart from these buttons, one *Space* character is also present allowing a user to enter blank character.

Design rationale: The user interface as stated above has been planned for touch-enabled mobile phones supporting two ways of phone use. One way is to grip the phone in one hand and access it by thumb of the same hand (49% of the world smartphone users hold their handsets in this way [85] (see Fig. 3.13(a))). The other way is holding the phone in one hand and accessing with other hand's finger or thumb (36% of the total users, where 72% of them use thumb and 28% use fingers [85] (Fig. 3.13(b))). The design of the basic layout is influenced by this fact of holding handset habits of users. Note that the basic design can be easily customized to left-handed users. In addition to this, the interface design is driven by solving certain linguistic and interaction issues enlisted below.

- Accommodating a large alphabet set in a small display devices: We call a mobile device is with small display, if its screen size is $4'' - 5''$ [218]. Entire alphabet set can not be comfortably accommodated in such a small displayed screen. Note that a typical Brahmic language consists of approximately 67 characters in alphabets (37 consonants, 15 vowels and dependent vowels each etc.). We categorize 25 regular consonants present in the language alphabets belonging to Brahmic script family into 5 groups called as *vargas* (consonants that stop air from moving out of the mouth [220]). This categorization is well-known and mentioned in language grammars [179]. The *vargas* are ordered according to where the tongue is in the mouth [179]. Each successive *varga* refers to a successively forward position of the tongue. The *vargas* are named as (with an example of a corresponding consonant) *Velar* (ordered as unaspirated, unvoiced), *Palatal* (aspirated, unvoiced), *Retroflex* (unaspirated, voiced), *Dental* (aspirated, voiced) and *Labial* (nasal). According



(a) Holding with one hand **(b)** Cradling in two hands

Figure 3.13: User interactions holding with phones

to the grammar, each *varga* group is named with the first character of it. So, we display first character representing each *varga* group in the basic keyboard layout. The remaining consonant characters are grouped following the same principle of *varga* (like four *semivowels* and *fricatives*) [179]. Our proposed interface includes leader character in five vargas at the first layer and other characters in a particular varga in the last layer. This avoids necessity of all characters or managing several layers/levels. Likewise, vowels and *matra*s are grouped together to provide an easy layering.

- Size of character button: We maintain standard width of each character button (48 to 50 pixels) and command button (60 to 65 pixels) (it is nearly to the study results conducted by MIT *Touch Lab* [43] as 45-57 pixels for index finger width and 72 pixels for thumb width (Fig. 3.14)), suitable for touch-enabled mobile phones having screen length of 10.16 cm or more,. This recommendation is to mitigate the *fat finger* or *fat thumb* problem.

- Avoiding occlusion during character selection: While tapping a leader character button in the keyboard, the hovering panel explores at a side of the position of user's finger or thumb (Fig. 3.15). The only occluded character, in this case, is the leader character which is already known to user. So, the scenario, overall, does not obstruct the view of users in finding the intended characters.

- Placing keyboard characters in appropriate locations supporting user comfort: Suppose, the proposed interface design supports right thumb oriented interaction

Figure 3.14: Pixel widths of index and thumb fingers

Figure 3.15: Avoiding occlusion effect

(that is, while holding the device with right hand only). In this situation, the lower right region of the interface becomes most difficult place to be considered within thumb's reach [24]. Keeping this problem in mind, we propose to place the less frequently used characters into that region (such as special symbols, numeric character group common buttons). This principle is driven by very less occurrence of thumb interaction in this area.

- Mitigating touch-level inaccuracy: In the proposed user interface, each character is inserted with touching key twice. This policy, although, increase the number of key presses in overall text typing, but it mitigates touch-level inaccuracy as after each initial selection, user gets a chance to withdraw a selection, if she thinks a wrong selection going to be happened.

- Supporting user in fixing up their concentration in target selection: After selecting a panel for a group, other characters in the keyboard are kept inactive which, in fact, facilitates less distraction of users.

3.2.2 Character level prediction

Character level prediction has been considered in our design to support faster text entry rate as well as to avoid character level errors. In fact, the character level prediction is a challenging task in touch sensitive small display devices. In such devices, the touched input is inaccurate as the size of the target (that is, button size) is smaller than thumb or index finger. In other words, if a current selection is erroneous then it induces the next character prediction. This uncertainty in character selection has been addressed by Bi and Zhai [19]. According to Bi et al. [17], the finger touch input can be interpreted as a sum of two independent normal distributions. Among them, one distribution reflects the relative precision governed by the speed-accuracy trade-off rule in the human motor system, and the other captures the absolute precision of finger touch independent of the speed-accuracy trade-off effect. Bi and Zhai [19] proposed a statistical criterion of finger touch selection namely *Bayesian touch criterion* (BTC) combining the basic Bayes' rule of probability with the generalized dual Gaussian distribution hypothesis of finger touch. The criterion states that a selected character is the candidate with the shortest *Bayesian touch distance* (BTD) to the touch point, which is computed from the touch point to target center distance and the size of the target. In our work, we use BTD score to decide the most probable candidate. Then we augment this BTD score with the trigram probability values obtained from language model. The aggregated result is then used to calculate the rank of candidate characters. In summary, our approach of predicting next

possible characters consists of three steps, candidate generation based on Bayesian touch criterion, assigning probabilistic language model to candidate and candidate ranking. These three tasks are discussed in details in the following.

3.2.2.1 Candidate generation based on Bayesian touch criterion

The BTD between a touch point s and a target candidate t is given in Eqn. 3.9 [19].

$$BTD(s,t) = \frac{(s-c)^2}{2(\alpha W^2 + \sigma_a^2)} + \frac{1}{2} ln(\alpha W^2 + \sigma_a^2) \tag{3.9}$$

where s is the finger touch coordinates reported by the device, W is the width of t, c is the center of t, α and σ_a are empirically decided constants.

For two dimensional touch input scenario, s_x and s_y would be the coordinates of touch point s, c_x and c_y are the coordinates of the center of a target candidate t, and d is the diameter. So, Eqn. 3.10 becomes

$$BTD_{2D}(s,t) = \frac{1}{2}\left[\frac{(s_x-c_x)^2}{\alpha_x d^2 + \sigma_{a_x}^2} + \frac{(s_y-c_y)^2}{\alpha_y d^2 + \sigma_{a_y}^2}\right] + \frac{1}{2} ln(\alpha_x d^2 + \sigma_{a_x}^2) + \frac{1}{2} ln(\alpha_y d^2 + \sigma_{a_y}^2) \tag{3.10}$$

The units of these variables are measured in millimeter scale. As in touch-enabled devices, index finger and thumb have different precision, with the index finger is more accurate than thumb [216]. Bi and Zhai [19] estimated the values of α_x, σ_{a_x}, α_y and σ_{a_y} for finger and thumb inputs, based on the user evaluation, as 0.0073, 1.35, 0.0113, 1.18 and 0.0075, 1.24, 0.0104 and 1.12, respectively.

In order to calculate BTD score, it is required to calculate center point location of each character present in the interface. As all the characters are not visible at a time, the exact locations of them are calculated by exploring all characters, symbols and number groups. When user touches a location on the mobile device, we obtain the x- and y-coordinates of that point. The system then calculates BTD for each character using Eqn 3.10. The smaller the result is, the preference of that particular character gets higher. The selection is then prompted in text field area showing the composed text.

3.2.2.2 Probabilistic language model for characters

Our approach to calculate the language model probability is as follows. An n-gram model is a probabilistic model for predicting the next item in a sequence of n characters. Based on the occurrence of previously typed characters, this technique predicts next possible

character(s). For this purpose, we need a character level trigram model. Such a language model provides a probability distribution $P(\sigma)$ defined for strings over an alphabet character set $\sigma \in \Sigma^*$ over a fixed alphabet of characters Σ. We begin with *Markovian language model* normalization as random processes, that is, the sum of the probabilities for strings of a fixed length is 1.0. We consider the chain rule factors as $P(\sigma c) = P(\sigma) \cdot P(c|\sigma)$ for a character c and string σ. The n-gram *Markovian assumption* [98] restricts the context to the previous $n-1$ characters, taking $P(c_n|\sigma c_1 \dots c_{n-1} = P(c_n|c_1 \dots c_{n-1})$.

To calculate $P(c|\sigma)$, we follow the maximum likelihood estimator [98] for n-grams as $\hat{P}_{ML}(c|\sigma) = count(\sigma c)/extCount(\sigma)$ where $count(\sigma)$ is the number of times the sequence σ was observed in a training data and $extCount(\sigma)$ is the number of single-character extensions of σ observed. In other words, $extCount(\sigma) = \sum_c count(\sigma c)$. Finally, we use generalized form of *Witten Bell smoothing* technique for an uniform estimation of $P(c|\sigma)$values. Witten-Bell smoothing uses linear interpolation to form a mixture model of all orders of maximum likelihood estimates down to the uniform estimate $P_U(c) = 1/|\Sigma|$ (Σ denotes whole bigram or trigram character probability starting with character chunk c). The interpolation ratio $\lambda(d\sigma)$ (where string σ occurring after character chunk d) ranges between 0 and 1 depending on the context (Eqn. 3.11b):

$$\hat{P}(c|d\sigma) = \lambda(d\sigma)P_{ML}P(c|d\sigma) + (1 - \lambda(d\sigma))\hat{P}(c|\sigma) \tag{3.11a}$$

$$\hat{P}(c) = \lambda()P_{ML}(c) + (1 - \lambda())(1 - |\Sigma|) \tag{3.11b}$$

where $\lambda()$ denotes the interpolation ratio of null string occurrence. Generalized Witten-Bell smoothing defines the interpolation ratio with a hyperparameter θ (Eqn. 3.12):

$$\lambda(\sigma) = \frac{extCount(\sigma)}{extCount(\sigma) + \theta \cdot numExts(\sigma)} \tag{3.12}$$

We calculate $numExts(\sigma)$ as $|\{c|count(\sigma c) > 0\}|$ to be the number of different symbols observed following σ in the training data. The original Witten-Bell estimator set the hyperparameter $\theta = 1$. For $n-$gram model, the value of θ is usually chosen as three [123].

For a given target language, we have to obtain a set of character sequences, from which each character will be assigned the probability values as mentioned above. This constitutes $n-$gram ($n = 3$) language model measuring unigram, bigram and trigram values for all characters in a target language.

3.2.2.3 Candidate ranking

The next step in our method is ranking of candidates. In this method, each candidate in the preference list has a) next character probabilities (bigram or trigram, based on previous characters) according to the language model and b) *Bayseian touch distances* among touch point and neighbourhood characters. We use these two results to calculate the rank of the candidate characters. We consider the measure $\frac{1}{BTD(s,t)}P(c_1c_2c_3)$ where $\frac{1}{BTD(s,t)}$ is the reciprocal of BTD value and $P(c_1c_2c_3)$ is the probability of the character trigam sequence (c_1 is the next probable character and c_1c_2 are previous two composed characters). Each candidate sequence value is calculated using ranking formula and characters are ranked based on their descending order of values. The first six characters in the ranked list are displayed to user (in the area CPL, Fig. 3.12(a)) as the most predictable characters.

Design rationale: In mobile device based interaction, the finger touch input modality is usually ambiguous. It is required for a text entry system running in the smartphone is to properly disambiguate the intended character based on user's touch location. The proposed text entry system offers character selection at the cost of two keystrokes. To reduce the keystroke by one, it is easy for user to directly select probable character(s) from a list through single keystroke. Beside solving two aforementioned problems, character prediction list can also help users to minimize the key searching time during text composition.

3.2.3 Case studies in three Indian languages

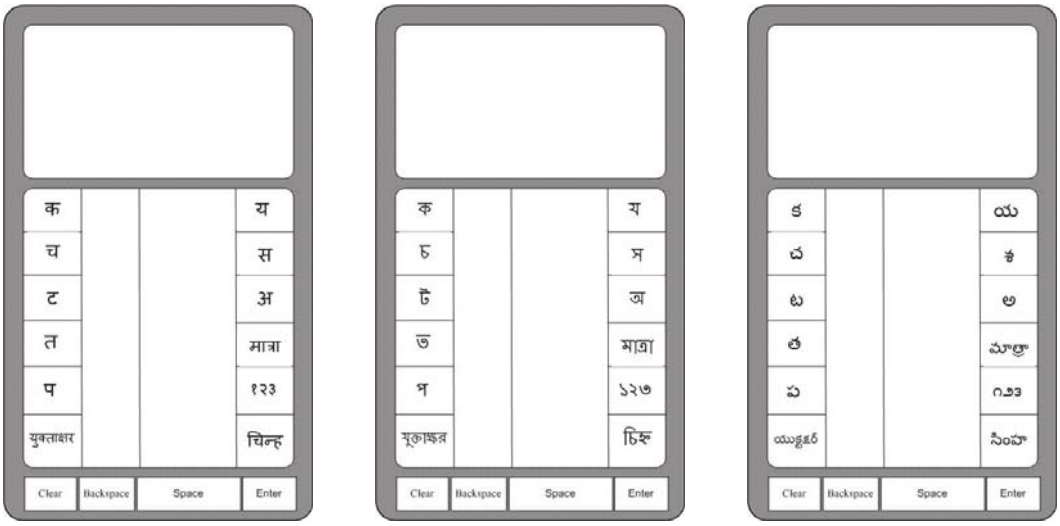
Typically, all Indian languages contain a large number of characters compared to English. The sets of characters and their classifications in three Indian languages namely Hindi, Bengali and Telugu are shown in Table 3.9. This table contains seven groups in consonants (five varga groups and two other constant groups), one each for independent vowel, dependent vowel (*matra*), conjuncts, numerics and symbols. It can be noted that there are approximately 12 vowel characters, 33 consonants, 13 dependent vowels and 10 numeric characters present in Hindi language [205], 12 number of vowels, 35 number of consonants, 11 dependent vowels and 10 numeric characters are used in Bengali written text [204] and 15 vowels, 35 consonants, 14 dependent vowels and 10 numeric characters in Telugu language [206]. The basic layout designs in three languages are shown in Fig. 3.16.

Table 3.9: Charcter grouping in Hindi, Bengali and Telugu languages

Language		Hindi	Bengali	Telugu
Independent vowel		अ आ इ ई उ ऊ ऋ ए ऐ ओ औ अं अः अँ	অ আ ই ঈ উ ঊ ঋ এ ঐ ও ঔ	అ ఆ ఇ ఈ ఉ ఊ ఋ ఎ ఏ ఐ ఒ ఓ ఔ
Consonants	Varga 1	क ख ग घ ङ	ক খ গ ঘ ঙ	క ఖ గ ఘ ఙ
	Varga 2	च छ ज झ ञ	চ ছ জ ঝ ঞ	చ ఛ జ ఝ ఞ
	Varga 3	ट ठ ड ढ ण	ট ঠ ড ঢ ণ	ట ఠ డ ఢ ణ
	Varga 4	त थ द ध न	ত থ দ ধ ন	త థ ద ధ న
	Varga 5	प फ ब भ म	প ফ ব ভ ম	ప ఫ బ భ మ
	Group 6	य र ल व श	য র ল ব	య ర ఱ ల ళ
	Group 7	ष स ह	শ ষ স হ	వ శ ష స హ
	Group 8	-	ড় ঢ় য় ৎ ঃ ঁ	-
Dependent vowel		ा ि ी ु ू ृ े ै ो ौ ं ः ्	া ি ী ু ূ ৃ ে ো ৌ	ా ి ీ ు ూ ృ ె ే ొ ో ౌ ్
Frequent conjuncts		क्ष त्र ज्ञ श्र ज़	ক্ষ ত্র জ্ঞ শ্র ঙ্গ	క్ష త్ర జ్ఞ శ్ర ఙ్గ
Numerics		१ २ ३ ४ ५ ६ ७ ८ ९ ०	১ ২ ৩ ৪ ৫ ৬ ৭ ৮ ৯ ০	౧ ౨ ౩ ౪ ౫ ౬ ౭ ౮ ౯ ౦
Symbols		| , ? - ;	| , ? - ;	| , ? - ;

3.2.4 Experiments and experimental results

Based on the approach proposed above, we have designed the interface for text composition in three Indian languages namely Bengali, Hindi and Telugu (designed layouts are shown in Fig 3.16). In order to validate the effectiveness of the proposed interface design, we have conducted a number of experiments with keyboards in Hindi, Bengali and Telugu. The experiments were conducted in two phases, first-time usability study and longitudinal study with both experienced and inexperienced users in order to observe the short and long term effects in text entry rate and error rate results on existing and proposed interfaces in touch-enabled mobile phones. Also, the interfaces were qualitatively evaluated by users to judge the usability, user-friendliness etc. through analyzing the user responses to a set of questions asked after each typing session.



(a) Hindi keyboard **(b)** Bengali keyboard **(c)** Telugu keyboard

Figure 3.16: Keyboards in Hindi, Bengali and Telugu languages

3.2.4.1 Apparatus

The experiments were conducted with a number of mobile handsets such as Nokia Lumia 525 smartphone (Windows Mobile), Windows Mobile 7 and Windows phone 8 operating system, Microsoft.NET framework 4.0 for Visual C#.NET 2010 as development framework.

3.2.4.2 Participants

Total 33 users participated in our experiments. 10, 13 and 10 users were for testing on-screen keyboards in Bengali, Hindi and Telugu languages, respectively. Details of the participants are summarized in Table 3.10. We chose the participants with different educational background and level of mobile accessibility. Their educational background varied from secondary level to post graduate level education. Thus, all participants were language literate. Participants' mobile accessing proficiency were also within a wide range, which can be categorized at three broad levels: (a) having at least five years of experience with mobile devices and two years experience with smartphones, know operating systems, programming and application software (Level 1) in mobile, (b) having $1-3$ years experience and mainly familiar with browsing different menus, game playing, messaging etc. in older mobile phones (mobiles with 12-key telephone keypad) and less than one year experience with touchscreen phones in performing same task to

Table 3.10: Description of participants

Participant's profile				Number of participants		
Occupation	Age group	Education	Computer proficiency	Bengali	Hindi	Telugu
Student	19-27	Under Graduate	Level 1	2	2	2
		Higher Secondary	Level 2	1	1	1
		Secondary	Level 2	1	2	1
Office staff member	31-48	Graduate	Level 1	2	2	1
		Higher Secondary	Level 2	1	1	2
Business person	34-53	Post Graduate	Level 1	1	2	1
		Graduate	Level 2	1	2	1
		Higher Secondary	Level 3	1	1	1

some extent (Level 2) (c) having less than one year experience with older mobiles and know only to receive and make phone calls, besides no experience with smartphones (by entering number and dial, not by searching from phone list) (Level 3). There are 10 female and 23 male participants and the average age is 26.37 years (SD=4.22). Out of these 33 participants, 21 participants are familiar in typing text with mobile keypad and touchscreen keyboards in English or Indian languages. We call them as "*Experienced*". The rest of the participants, we call "*Inexperienced*", who are not familiar to any text entry system and thus new to composing text in any language.

3.2.4.3 Designs in evaluation

We consider overall seven keyboards including proposed *mLiPi* layout for user evaluation. The candidate keyboard layouts in Hindi, Bengali and Telugu cover popular keyboard design principles like Indic script (Inscript), alphabetic, phonetic and frequency-based designs. The keyboards are namely, Indic language (Inscript) on-screen keyboard layout [195], three alphabetic keyboards, a) *Swarachakra* proposed by Joshi et al. [96], b) keyboard proposed by Jung et al. [97] in Hindi and c) *Sparsha* Telugu keyboard [8] and frequency-based layouts which are a) *Panini* keypad developed by *Luna Ergonomics* [53] (we have mimicked the original layout without attaching the word completion facility) and b) Bengali two layer based keyboard layouts, proposed by Bhattacharya and Laha [14]. We could not include the efficient keyboard layout proposed by Hinkle et al. [82] into designs to be tested because they had not provided structure of any layout other than Assamese language.

3.2.4.4 Text under test

The texts for testing were collected from various sources such as popular newspapers, books, formal conversations (taken from chats), idiomatic expression taken from Internet, movie dialogues etc. to form the corpus. The most frequently occurred words and sentences from the above mentioned sources are considered for our text for testing in Bengali, Hindi and Telugu. The words and sentences were grouped into *Easy*, *Medium* and *Hard* level according to the usage of the glyphs and number of the conjunct characters. We marked *Easy* words which contain only characters and glyphs (average word length 3.47 characters), *Medium* words which are having one conjunct and *Hard* words containing more than one conjunct. Similarly, we designated *Easy* sentences which hold $4-5$ words having no conjunct, *Medium* sentences containing $5-6$ words having a word with conjunct and *Hard* sentences comprising $7-8$ words having at least two words with conjuncts. The detail categorizations of words and sentences used for testing in three languages are shown in Table 3.11.

3.2.4.5 Dependent measures

The dependent measures used in this experiment are words per minute (WPM) [128] and the total error rate. The formula to calculate text entry rate (in WPM) is already described in previous section.

Errors left in the final typed text are being considered to calculate the error rate. The low error rate signifies either user has typed correct characters or corrected them after committing errors. To avoid the confusion, we consider total error which constitutes of two error types, *Corrected* and *Uncorrected*. *Corrected* indicates those characters which, after wrongly typed, are corrected by user and *Uncorrected* means the characters which left erroneous in the final composed text [185,226]. A log file records every activity of the user in text composition from which we can easily quantify the corrected and uncorrected error types.

Table 3.11: Testing text categorization in Hindi, Bengali and Telugu

Language		Easy	Medium	Hard
Hindi	Word	40	40	40
	Sentence	35	35	35
Bengali	Word	42	25	30
	Sentence	30	30	30
Telugu	Word	32	25	22
	Sentence	27	23	20

3.2.4.6 Procedure

The user evaluation sessions, for each *Experienced* or *Inexperienced* users, were executed in three steps, Training, First time usability study task and longitudinal study task.

The training sessions were conducted with the objective to increase the familiarity of the participants with the device as well as task performing ability in those devices. Initially, they were asked to play one to two popular games and a text typing game in their mobiles. Beside those, we gave participants moderate time (mostly a day or so) to get familiarized with the devices given to them. After the user got familiarity with the devices, we ask them to use keyboard and editor. The steps of the training phase were as follows. We informally followed *Text Transcription* method [112] for this phase. First, the instructor gave a demonstration on typing of few words through the assigned keyboard. After that, user was instructed to select the same word following the same way. Once participant had committed an error in typing, instructor pointed out it and asked him to correct that. If user still was not able to correct it, instructor repeated the same word composition procedure by his own. This process was continued until the user composes the word correctly. Once the word was typed correctly, instructor asked the user to enter the next word. The training was continued until all training words are completed. All users were shown the words in the same order during training. When all words were typed, the user had the choice to type whatever he likes in the typing window. This phase ended with user's confirmation about completion of their practice. Each user was given training at least once. However, if instructor thought that user needs extra training, he conducted the phase once again.

First time usability task began immediately after the training session gets over. Here, users were asked to input 20 words (randomly selected 8 *Easy*, 6 *Medium* and 6 *Hard* words from the testing corpus) one by one as fast as possible where words were provided in printed form. If any of the words was typed incorrectly and the user overlooked it, instructor marked the error, might repeated the help instruction, and then asked the user to correct it. This help was given once for each word and after that, we gave a chance to type the word for a second time. Words were shown to all participants in the same order during this task. After all words were typed, the user was shown how well he had done (what he was supposed to type, what he had typed on the first attempt, what he typed on the second attempt if any, the errors if any, and the speed). Along with this, we offered an option where user can also type the text by filling up the form (*Form fill-in* method [177]). When the session got over, each user was asked to rate the task on a *Likert* scale of $1-5$ for difficulty.

We planned to conduct two ways of performing longitudinal study experiments, a)

Read and type (read the text from the paper then type it) and Type freely (type a text of user's own). After analyzing the collected user evaluation results in first-time usability test from the log file, we comprehended that users did not select some characters in the language as those characters did not appear in their typed texts. That problem hindered us to decide the text entry rate in an optimal way. To overcome the sub-optimality of the result, we also followed the *Read and type* method in which texts are decided so that all language characters present in the texts with uniform probability. Here, we considered a number of texts for performing *Read and type* method by users, chosen from the testing corpus. For *Read and type* method, without providing any prior training, users were asked to compose 10 sentences (randomly selected five *Easy*, three *Medium* and two *Hard* sentences from the testing corpus) as fast as they can do through both study design keyboards with existing layouts along with proposed *mLiPi* keyboard.

For *Type freely* testing method, a form consisting of 25 entries (like name, address, occupations, hobbies etc.), was given to users and thus are required to fill up the form. Unlike the first time usability task, no feedback, help or second attempt was provided for the longitudinal task between phrases. This task allowed only one attempt per phrase. Once the user had finished typing all the phrases or filled up the form, we have summarized his performance.

The testing corpus, used for *Read and Type* and the keyboard for typing were chosen randomly (from the pool consists of both study design and existing keyboards) in each session to minimize the memorizing effect of users. The keyboard order and typing method selection for a particular session were counterbalanced across all the participants. Each study lasted approximately 70 to 80 minutes. On an average, a user spent time in performing $2-3$ experiments per day. Most of the users had performed text entry with all the mobile keyboards in their mother languages. Only a few have not completed all the experiments successfully. A user, who had evaluated all keyboards in different conditions, approximately took on an average three months to finish. Each user session was automatically logged into a file. After completion of a session, the session log file was analyzed to calculate text entry rate. After typing completion, users were asked to rate the task on a $1-5$ *Likert* scale and then requested to leave with a remainder to come for the next session.

3.2.4.7 Experimental Results

The main objective of our experiments is to study the text entry rates of the existing virtual keyboards along with two alternate designs and proposed *mLiPi* keyboard through user based evaluations. Also, our objective is to analyze corrected and uncorrected errors

occurred during text composition. The user experiments were conducted in two stages namely first-time usability task followed by longitudinal study. Also, users performed experiments with existing keyboard layouts. The observed results are summarized below.

Further, we analyze successfully completed within subject user experiment results statistically. For this, we conducted analysis of variance test with SPSS tool [89]. As the average word lengths of these three languages are different, we performed weighted mean of all three language results using the formula described in previous Section 3.1.

First-Time Usability Test: Participants, for existing keyboard layouts, completed a total of two sessions × seven designs × two typing methods = 28 sessions. So, with 33 participants in three languages, the entire study comprised of 2872 trials.

Analyzing user evaluation results in Hindi, Bengali and Telugu, we calculated users' typing speed along with corrected (number of backspaces pressed) and uncorrected (error remaining in the typed text) error rate for all the keyboards. In our experiments, all users composed words in each keyboard at least once. For the first-time usability test, we gathered all users (both *experienced* and *inexperienced* category people) and keyboard based results and averaged them. The average text entry rates achieved by participants with existing keyboard along with the proposed *mLiPi* design in three languages are shown in Table 3.12.

We analyzed total error rate of users which is a combination of corrected and uncorrected error rate. Over the sessions, the average total error rate for *mLiPi* is 2.88%. On the other hand, for *Indic*, *Panini*, *Swarachakra*, Jung et al's keyboard, Bhattacharya and Laha's Bengali keyboard and *Sparsh* Telugu keyboard, total error rates are 14.02%, 9.11%, 11.95%, 13.53%, 12.80% and 8.98%, respectively. However, for all keyboards, total error rates are dropped significantly over sessions ($F(1, 1434) = 6.33$, $p < 0.05$). The observation of the experiments reveals that using the proposed *mLiPi* interface, users left very few errors uncorrected than other designs, that is, the number of corrected errors is much more in case of proposed interface. Also, *Post-hoc Tukey test* analysis of variance reveals that there is significant differences in error rates between the existing keyboard designs and proposed interface ($p < 0.05$).

Table 3.12: First time usability study results

Language	Text entry rate (in WPM)						
	Indic [65]	Swarachakra [96]	Panini [53]	Jung [97]	Bhattacharya [14]	Sparsh [8]	mLiPi
Hindi	3.58	3.86	3.53	3.91	–	–	3.96
Bengali	3.65	3.82	3.67	–	3.88	–	3.89
Telugu	3.23	3.68	3.16	–	–	3.73	3.77

Results of longitudinal study: Within subject experiments were performed with 7 keyboards measuring text typing rate and corrected, uncorrected and total error rate. Also, after completion of each session, we collected subjective feedback from each participant. Data for each user were averaged in each session to form single measure per participant per session on a variety of metrics. In this study, Participants completed a total of two trials × seven keyboards × eight sessions = 112 trials. With 33 participants, the entire study comprised of 3696 trials. Also, for testing sessions, we randomly selected a keyboard for a participant in each session. But the overall keyboard order was kept counterbalanced across participants. Maximum three sessions were performed per day by each participant. The whole study lasted for approximately six months.

Average experienced and inexperienced user text entry rate results with *Read and type* and *Type freely* methods in Hindi, Bengali and Telugu are given in Table 3.13.

While comparing the proposed *mLiPi* keyboard with six existing language keyboards, the analysis of variance tests (ANOVA) performed for experienced and inexperienced users on *Read and type*-based text entry reveal that the mean text entry rate of the keyboards are not same ($p < 0.05$). On the other side, while we compare the results obtained from users performance for composing text freely (*Type freely* method), we also observe that there is significant difference between performance of mobile on-screen keyboards ($p < 0.05$). The statistical test, that is analysis of variance test, has been conducted for comparing mean text entry rates of inexperienced and experienced users

Table 3.13: User evaluation result with longitudinal study

Language	Text entry method	Type of user	Text entry rate (in WPM)						
			Inscript	Panini	Swarachakra	Jung's KB	Bhattacharya and Laha's KB	Sparsh KB	mLiPi
Hindi	Read & Type	Inexp	4.98	6.34	5.63	5.72			6.13
		Exp	5.13	5.45	5.81	5.59			6.17
	Type Freely	Inexp	5.07	5.32	5.67	5.78			6.09
		Exp	5.27	5.59	5.85	5.74			6.24
Bengali	Read & Type	Inexp	4.93	5.28	5.69		5.66		6.21
		Exp	5.06	5.51	5.87		5.82		6.28
	Type Freely	Inexp	5.09	5.37	5.61		5.71		6.12
		Exp	5.22	5.46	5.75		5.78		6.25
Telugu	Read & Type	Inexp	4.65	4.68	4.72			4.75	5.26
		Exp	4.78	4.81	4.86			4.83	5.39
	Type Freely	Inexp	4.82	4.84	4.91			4.79	5.43
		Exp	4.91	4.96	5.04			4.94	5.56

*Inexp: Inexperience, Exp: Experience, KB: Keyboard

in case of *Type freely* method. The result substantiate that there is significant difference between performance of keyboards irrespective of users' proficiency (for inexperienced, $p < 0.05$, and experienced $p < 0.05$ also).

Further the *Independent T-test* result signifies there is a significant difference between the mean text entry rates (WPM) achieved by experienced and inexperienced users ($t = -2.84, df(12)$ and $p < 0.05$ for two tailed test). The result reflects that for a fixed type of text composition task and text to be typed, experienced users outperform people belong to inexperienced category. Similarly, we gather users' text composition results keeping other constraints fixed and analyze the tendency of data. The statistical test (*Independent T-test*) has been conducted which announces that there exists significant differences between the mean text entry rates achieved through *Read and type* and *Type freely* methods ($t = -2.27, df(8)$ and $p < 0.05$ for two tailed test). Experimental results follow the traditional fact that, for other condition unaltered, experienced users and *Type freely* method proved to be better than inexperienced users and *Read and type* method, respectively.

Total error rate: Soukoreff and MacKenzie [184] define total error rate to be the sum of uncorrected and corrected errors. Over eight sessions, the total error rates, averaged over all the participants, in Hindi, Bengali and Telugu are given in Table 3.14.

The results we observed from the above error analysis do not strictly reflect better performance of the proposed keyboard than other designs. In contrast, the observation reveals that using the proposed *mLiPi* keypad, users left a few errors uncorrected than other designs, that is, the number of corrected errors is more in case of proposed interface. Further, to get a clear picture, we analyze number of errors left in the transcribed text which is being indicated as accuracy measure, for all the seven designs. An analysis of variance reveals that there is significant difference in total error rates between the keyboard designs ($p < 0.05$). Further study (*Post-hoc* using Tukey HSD test) confirms difference of *mLiPi* design from others ($p < 0.05$).

Subjective evaluation: Participants completed a subjective questionnaire after each

Table 3.14: Longitudinal study error rates (in %) for keyboards in Hindi, Bengali and Telugu

Language	Text entry error rate (in %)						
	Indic	Swarachakra	Panini	Jung	Bhattacharya	Sparsh	mLiPi
Hindi	23.05	18.96	24.48	24.82	–	–	21.73
Bengali	23.78	19.65	24.76	–	23.71	–	21.24
Telugu	22.91	19.10	25.97	–	–	25.71	21.62

session. We collected the overall ratings from the participants and analyzed the data with nonparametric *Wilcoxon Matched Pairs Signed Ranks Test* as Likert-scale data did not often conform to the assumptions required for ANOVA procedures.

Participants preferred *mLiPi* significantly over the other keyboard interfaces. They felt that *mLiPi* was easier to use ($z = 46.00, p < .001$), faster ($z = 45.00, p < .01$), and less fatiguing ($z = -53.00, p < .001$) than other interfaces while typing both in *Read and type* and *Type freely* methods.

Participants also expressed that the concept of double layer activation for character selection helped them to select their intended characters most of the time (overcoming frequent wrong character selection occurred due to *fat finger* problem). Grouping of vowel (all into one group) and consonant (all *varga* characters in a group) characters initially made user confused about the intended character existence within the specific group. Users mostly agreed upon the fact that next probable character displaying scenario reduced the character searching time. It has also been observed as well as understood by participants that first few sessions were spent to get familiarized with all the on-screen keyboards. *mLiPi* keyboard was learned quickly than other keyboards and after that, the text typing speed, according to users, was significantly increased session by session.

Regarding Swarachakra interface [96], participants expressed positive feedback about the simplicity, intuitiveness and design uniqueness of it. But many of them said that they had felt difficulty in selecting a *matra* after the consonant as the finger occlusion hinders in understanding characters placed at the bottom in the "*Chakra*". An interaction level scenario with *Swarachakra* keypad is that after selecting *halant*, it combines all consonants with character selected before *halant* and displays them in the keyboard. This situation initially made most of the participants confused which, after spending some sessions, got minimized with a little effect remained. *Swarachakra* keyboard contains 30 soft key buttons having smaller sizes and no gap is maintained between two consecutive keys. Consequently, users faced problems in tapping the intended character as the key area became smaller (neighboring key press due to *fat finger*). Similar key selection problem had occurred with Jung et al.'s keyboard during user experiment as it displays vowel group (both dependent and independent, displays one depending on last typed character) characters in addition to *Swarachakra* layout. The *Panini* keypad displayed 12 characters in a layer where participants spent huge time to search a character which is less frequent. *Indic* on-screen keyboard and *Sparsh* Telugu keyboard layouts consist of two character layers where number of characters in a layer is more than *Swarachakra*. This scenario, in fact, increases the visual search time to find a character as well as chance of committing errors due to neighboring character press (*fat finger* problem). The design of Bengali

keyboard layout proposed by Bhattacharya and Laha [14] is similar to *Swarachakra* keyboard layout with a difference of frequency based arrangement of character keys. Similar to *Swarachakra* layout, it suffered with *Fat finger* problem. It was difficult for the participants to remember character location in two leyer, specially while toggling. This scenario increases participants' character searching time in the interface.

3.3 Summary

In this chapter, we present two virtual keyboard-based text entry mechanisms in Indic languages, namely mouse-based for desktop devices and finger or touch-based for touch-enabled mobile devices. The concept of multi-zonal layout and optimal arrangement of characters in each zone proves to be an effective virtual keyboard design solution suitable for mouse-based text entry in desktop devices, so far the reduced hand (and eye) movements and hence a better text entry rate is concerned. In fact, our proposed keyboard achieves more text entry rate than existing virtual keyboards for Indic language text entry, such as *Google* (for Hindi language), *Avro* (for Bengali), *Guruji* (for Telugu) etc. Further, inclusion of dependent vowel window reduces display space overhead required for on-screen graphics keyboard. Proposed *iLiPi* keyboard design is found to be superior even compared to the virtual keyboards developed following popular English language keyboard design principles.

The proposed design for touch-enabled mobile devices yields moderately better text entry rate compared to the existing Indic language solutions. Less characters are arranged in the proposed keyboards in such a way that the chance of occurring wrong character (neighboring) selection problem due to *fat finger* gets reduced. Whereas, other characters are displayed dynamically, based on already selected characters in the layout. Beside providing an effective solution toward accommodating a large character set within space-constrained display area, the design provides a character-level prediction with error correction support. The proposed *mLiPi* interface also provides facility to minimize character searching time in the keyboard. The proposed interface design principle can be extended to many other languages in India as well as outside. In the next chapter, we design word-level prediction strategy to enhance text entry rate as well as to reduce user errors while typing through Indic language virtual keyboards in desktop as well as touch-enabled mobile devices.

Chapter 4

Word Level Prediction

In this chapter, we describe a word-level prediction method as a text entry rate enhancement strategy in Indic languages. Due to linguistic complexities and large variation in character combinations, the character by character text entry rate remains slow compare to text entry in English. Further, Indic language text entry often suffers with user-level errors which occur due to many reasons like confusion between graphically and phonetically similar characters, improper use of input sequence to form texts, lack in knowledge about morphological variance of words, system-level errors due to many reasons like absence of different representational forms of words in dictionary etc. Moreover, sometimes system cannot interpret user input like wrong touch location selection due to *fat finger* or finger occlusion when performing touch and drag gesture etc. which results wrong character or text chunk entry. We aim to implement text entry rate enhancement mechanism which also enhances the quality of the composed texts. Our proposed prediction approach addresses many of the aforementioned problems. Apart from dealing with linguistic intricacies, the method suggests correct spelling of the predicted words even those are under composition. So, using this method, users need not to erase the character for correcting words which saves the number of keystroke and time for composing texts in Indic languages.

The rest of the chapter is organized as follows. Section 4.1 describes the word=level prediction methodology for large as well as small screen devices. Experimental setup, results and analysis of our proposed prediction system augmented with virtual keyboard design for desktop and mobile devices are presented in Section 4.2. Section 4.3 summarizes the chapter.

4.1 Prediction methodology

In this section, we discuss our approach to develop a word prediction system in the context of Indic languages. The major objectives of the proposed prediction system include the following.

- Developing a mechanism for the fast and accurate text entry with a lesser number of key presses.
- Dealing with the complexities of text entry in Indic languages.
- Handling several types of error which may occur at any position(s) in words under composition.
- Implementing an efficient multi-variate method for ranking to select the best candidates for the prediction list.

An overview of our proposed prediction methodology is shown in Fig. 4.1. There are three main tasks in our approach: a) resource creation b) generation of candidate list and c) ranking of candidates. These tasks are discussed in details in the following sub sections.

4.1.1 Resource creation

Our proposed word prediction algorithm requires some resources which are to be preprocessed once prior to the execution of the algorithm. Our resource consists of a language model, list of graphical/phonetically similar character) and a set of confusable words (see Fig. 4.1). We need the probabilities of occurrences of words. This can be done by developing a language model. As similar characters (in terms of graphical/phonetic similarity) cause confusion in Indic language text entry, a list is required to identify those characters in order to mitigate the typing error further. There are various valid words which are confusable with other words present in vocabulary in the context of spelling similarity. In this work, we utilize a set of confusable words to handle first character error in typing. In fact, this helps in identifying and generating a potential list of candidate words for prediction system.

4.1.1.1 Development of language model

A language model contains words and their probabilities in a given context. To create an effective language model, we would rely on a large written text corpus. A detail description of developing the training corpus and language model is given below.

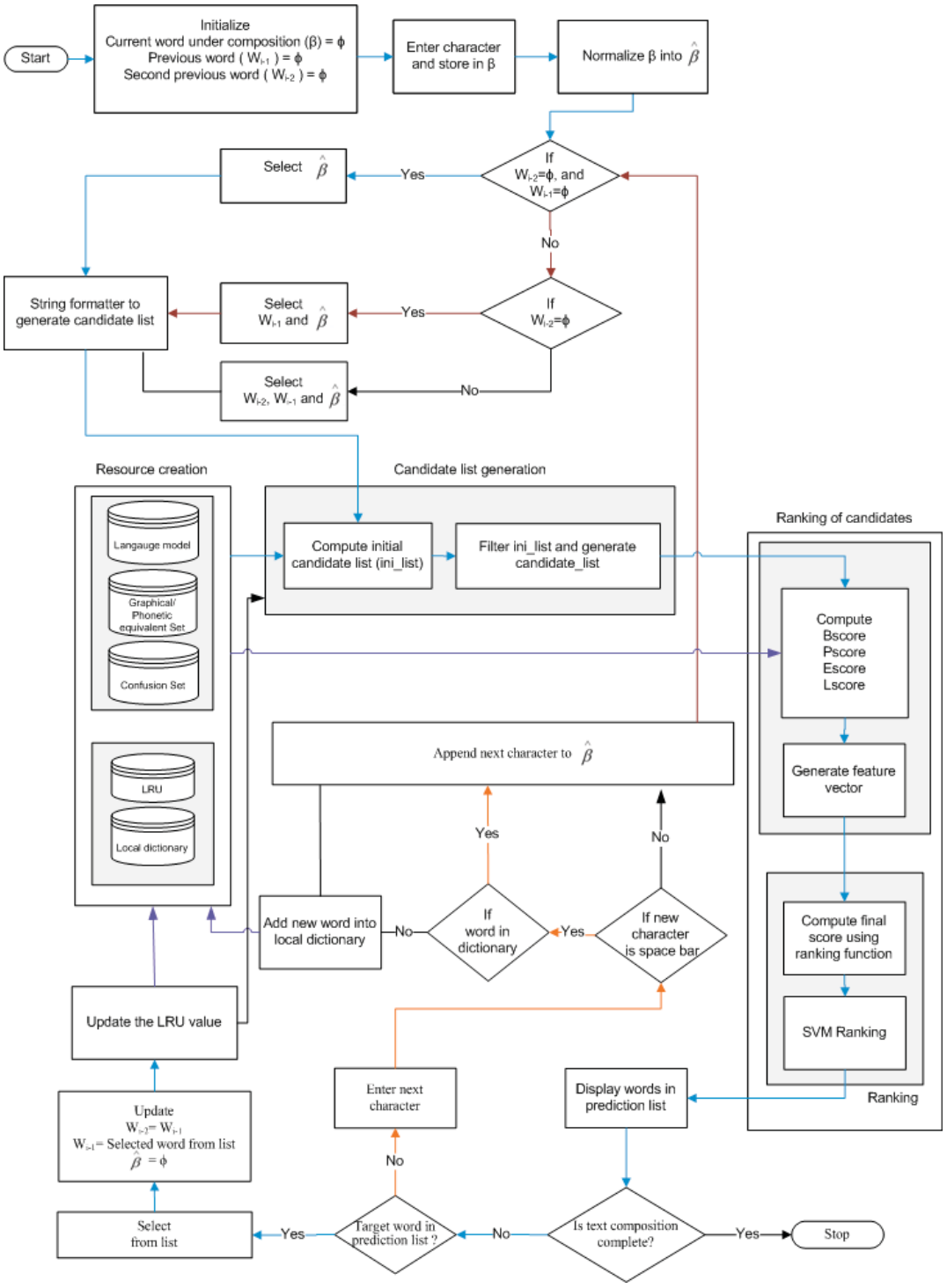


Figure 4.1: An overview of the proposed prediction system

i) Development of training corpus: There are several sources such as newspapers, novels, literatures etc. can be considered for the purpose. In this work, we consider the Internet repository, since it is huge, complete and easily available. We have taken the texts from *Wikipedia* for corpus creation. The content of *Wikipedia* can be downloaded from archive file. A link to few such archives is shown in Table 4.1.

The *Wikipedia* pages contain information in the forms of text, images and HTML tags. Since only text part is required, we filter the pages to remove all non-relevant elements. This is done using XML parser namely "WP2TXT" [1]. The filtered data are saved into a text file. Some redundant data (such as word having more then 20 characters, other language contents etc.) may still exist in the processed file. Those are removed by writing a program. The extracted texts are then "normalised" [38, 189] and removed the occurrence of "Zero-Width Joiner" (ZWJ) and "Zero-Width Non Joiner" (JWNJ) [40], if any.

ii) Computation of probabilities of occurrences: The processed text corpus is used to generate the "Backoff n-gram language" model [98, 99, 171], containing unigram, bigram, trigram and probability mass distribution. Let w_i be the current word under composition, w_{i-1} and w_{i-2} are the last two words entered. According to "Backoff n-gram modeling approach", if we see no counts (unseen event) for a trigram sequence say $w_{i-2}w_{i-1}w_i$ (also written as w_{i-2}^i) to determine $P(w_i|w_{i-1}w_{i-2})$, then we can estimate its likelihood by using the bigram probability $P(w_i|w_{i-1})$. If we do not see counts to calculate $P(w_i|w_{i-1})$, then we can use unigram probability $P(w_i)$. However, if we have a non-zero count for trigram (seen event), we depend on trigram counts and do not interpolate the bigram and unigram counts. We only "back off" to a lower order *n-gram*, if we have zero evidence for a higher order *n-gram*. Backoff *n-gram* modeling is formally expressed in Eqn: 4.1.

Table 4.1: Wikipedia archives

Language	Download Link	File Name	Size
English(en)	http://dumps.wikimedia.org/enwiki/latest/	enwiki-latest-pages-articles.xml	3.4G
French(fr)	http://dumps.wikimedia.org/frwiki/latest/	frwiki-latest-pages-articles.xml	1.3G
Hindi(hi)	http://dumps.wikimedia.org/hiwiki/latest/	hiwiki-latest-pages-articles.xml	193.7M
Bengali(bn)	http://dumps.wikimedia.org/bnwiki/latest/	bnwiki-latest-pages-articles.xml	51.1M

[1] it can be downloaded from `http://wp2txt.rubyforge.org/index-old.html`

$$\widehat{P}\left(w_i|w_{i-1}w_{i-2}\right) = \begin{cases} \widetilde{P}\left(w_i|w_{i-1}w_{i-2}\right), & \text{if } C\left(w_{i-2}w_{i-1}w_i\right) > 0 \\ \lambda\left(w_{i-2}^{i-1}\right) \times \widetilde{P}\left(w_i|w_{i-1}\right), & \text{if } C\left(w_{i-2}w_{i-1}w_i\right) = 0 \\ & \text{and } C\left(w_{i-1}w_i\right) > 0 \\ \lambda\left(w_{i-1}\right) \times \widetilde{P}\left(w_i\right), & otherwise. \end{cases} \tag{4.1}$$

Here, $\widehat{P}\left(w_i|w_{i-1}w_{i-2}\right)$ represents trigram backoff probability and $\widetilde{P}$ is discounting probability computed using Eqn: 4.2. λ is a function of the preceding word string and gives the probability mass distribution from an *n-gram* to $(n-1)$-*gram* (estimated according to Eqn 4.4) based on Good-Turing discounting method [98,136,214].

$$\widetilde{P}\left(w_n|w_{n-N+1}^{n-1}\right) = \frac{C^*\left(w_{n-N+1}^{n}\right)}{C\left(w_{n-N+1}^{n-1}\right)} \tag{4.2}$$

Here, C is the number of counts for seen word sequence. C^* represents discounted count which is used to reserved a certain amount of probability mass for unseen *n-gram* (w_{n-N+2}^{n-1}) and N represents order of *n-gram*. The discounted count estimate (C^*) [98] for large count (where $C > k$, k is threshold) is defined as

$$C^* = \frac{(c+1)\frac{N_{c+1}}{N_c} - c\frac{(k+1)N_{k+1}}{N_1}}{1 - \frac{(k+1)N_{k+1}}{N_1}}, for 1 \leq c \leq k \tag{4.3}$$

Here, N_k and N_c are the number of *n-gram* that occur exactly k and c times in corpus, respectively. With Good-Turing discounting method, it is usual to treat *n-gram* with low counts (especially count of 1) as if the count was 0.

$$\lambda\left(w_{n-N+1}^{n-1}\right) = \frac{1 - \sum_{w_n:c\left(w_{n-N+1}^{n}\right)>0} \widetilde{P}\left(w_n|w_{n-N+1}^{n-1}\right)}{1 - \sum_{w_n:c\left(w_{n-N+1}^{n}\right)>0} \widetilde{P}\left(w_n|w_{n-N+2}^{n-1}\right)} \tag{4.4}$$

We use CMU-SLM toolkit [35] to calculate backoff probabilities for *Wikipedia* resource. The language model is created and stored in ARPA file format [35]. It contains unigram, bigram and trigram probabilities; it also contains the probability mass distribution (used in a situation when one of the higher order *n-gram* sequence does not exist).

4.1.1.2 Graphical/Phonetic equivalent set

In Indic language writing, character level confusion may occur due to graphical/phonetic similarity [173, 174]. We create a list of confusable characters in terms of graphical/phonetic similarity following the structure presented in Table 4.2. In this table, C_1, C_1^1, C_1^2 and C_1^3 are characters either phonetically or graphically similar to each other and all are mapped to some specific character C_1, which is termed as *Character Set IDentifier* (CSID). As an example, $CSID(C_1^2) = C_1$ and $CSID(C_1^3C_2^1) = C_1C_2$. In this table, S represents special character present, if any, in a language.

A sequence of *CSIDs* is required for word level similarity, for a given character sequence α (and it is denoted as $CSID_{seq}(\alpha)$). It can be computed using Algorithm 4.1. We illustrate the algorithm with an example as follows. Let α be a word consists of characters $C_1^1C_2^2S^1C_2^2$. To compute $CSID_{seq}$ for α, we first convert each character in α into its respective $CSID$ (hence, α becomes $C_1C_2SC_2$) and then remove occurrence of S (a special character) if present (see Table 4.2). We then remove the consecutive duplicate character(s) from this string. Thus, $CSID_{seq}(\alpha)$ becomes C_1C_2.

Table 4.2: Graphical or phonetic similar sets of characters

CSID	Similarity set				
C_1	C_1	C_1^1	C_1^2	C_1^3	C_1^4
C_2	C_2	C_2^1	C_2^2		
C_3	C_3	C_3^1	C_3^2	C_3^3	
C_4	C_4				
C_5	C_5	C_5^1	C_5^2		
...	...	...	...	...	...
S	S	S^1	S^2		
...	...	...	...	...	...

Algorithm 4.1 $CSID_{seq}$ computation

Input: Given word α

Output: Returns $CSID_{seq}$ for α

1: $\delta \leftarrow$ Convert each character in α into its respective $CSID$
2: $CSID_{seq} \leftarrow$ Remove S, special $CSID$, if present, followed by deletion of consecutive duplicate *CSIDs* in δ
3: Return $CSID_{seq}$

4.1.1.3 Identification of confusable words

In a basic word level prediction system, prefix based matching of vocabulary words is performed to generate a probable candidate list. When an error occurs during text entry, the basic word prediction system is unable to predict the correct intended candidate in a given context, even if it is present in the dictionary. Such an error is due to the confusion between two or more words with spelling similarity. Spelling errors are frequently taken care by spell checker. It uses predefined set of words that are considered to be confusable such as (wright, right) and (cite, sight, site) etc. While user composes one of the words present in the set, spell checker uses certain rules to decide whether one of the other member of the set would be more appropriate in the given context [161]. After composing a text, spell checker detects the occurrence of error. In this process, it checks the confusable words as potential candidates to decide the appropriate word in a given context. Various works have been reported in regarding the creation of confusion set [142, 161]. According to [161], confusion set can be used both to detect and correct errors and is appropriate for both syntactic and semantic errors. One method of creating confusion sets by Mays et al. [142] is to group together words that differ from each other by a single edit distance.

In our work, to avoid errors occurred due to confusion with other words present in vocabulary, a set of confusable word pairs need to be properly identified, processed and plugged in the initial candidate list. We identify these word pairs (termed as confusion pair) and process them to generate *Confusion Set Identifier* at the preprocessing stage. This *Confusion Set Identifier* is invoked at runtime to generate initial candidate list (discussed further in Section 4.1.2). The procedure to compute *Confusion Set Identifiers* and their corresponding sets is precisely stated in Algorithm 4.2 and illustrated in the following.

1. Listing of confusable pairs: For each word (we call it as *Head Word*) in vocabulary (V), we compute a list of words which differ in one edit operation from that word. Each such word pair is termed as confusable pair. Suppose, *ABD*, *MBC*, *NBC*, *BC* and *ABCD* are five words having *Edit Distance* one with respect to *Head Word ABC*. Hence, confusion pairs are: (ABC, ABD), (ABC, MBC), (ABC, NBC), (ABC, BC) and $(ABC, ABCD)$. Similarly, confusion pairs for *ABD* are (ABD, ABC), (ABD, NBD) and $(ABD, ABCD)$.

2. From pairs to sets: Confusion set is created by taking all elements in a confusion pair for a specific *Head Word*. For example, confusion set for *Head Word ABC* is represented as {*ABD*, *MBC*, *NBC*, *BC*, *ABCD*}. The set for *Head Word ABD*

Algorithm 4.2 Computation of confusion sets

Input: list of vocabulary, Θ is predefined threshold and list L initially empty
Input: list of vocabulary V
Output: Confusion set L, initially empty

```
1: for each x ∈ V do
2:   γ ← SubString(x, 1)   /* 1st character of x */
3:   ID ← γ
4:   HeadWord ← x
5:   for each y ∈ V do
6:     count ← Edit Distance(x, y)
7:     δ ← SubString(y, 1)   /* 1st character of y */
8:     if ((γ ≠ δ) AND (count = 1) then
9:       if (y is not present in L) then
10:        Store y into L
11:      end if
12:    end if
13:  end for
14: end for
```

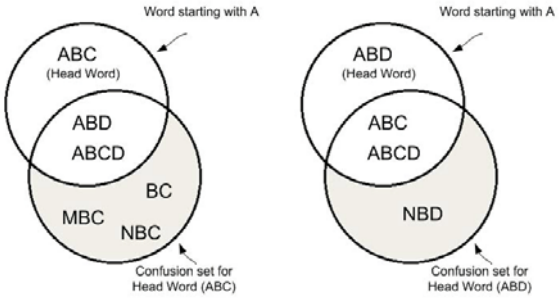
contains {*ABC*, *NBD*, *ABCD*} (see Fig. 4.2). Note that, *Head Word* is considered to be confusable with each word present in that set. However, it is not necessary that word in a given set is confusable with each other [161].

3. Filtering: For a specific *Head Word*, all words present in confusion set which do not start with 1^{st} character of *Head Word* are taken out. As an example, for *Head Word* *ABC* and *ABD* filtered lists are {*MBC*, *NBC*, *BC*} and {*NBD*}, respectively (see Fig. 4.2(a)). It is to be noted that, 1^{st} character of *Head Word* (that is, *Confusion Set Identifier*) is *A* for both words.

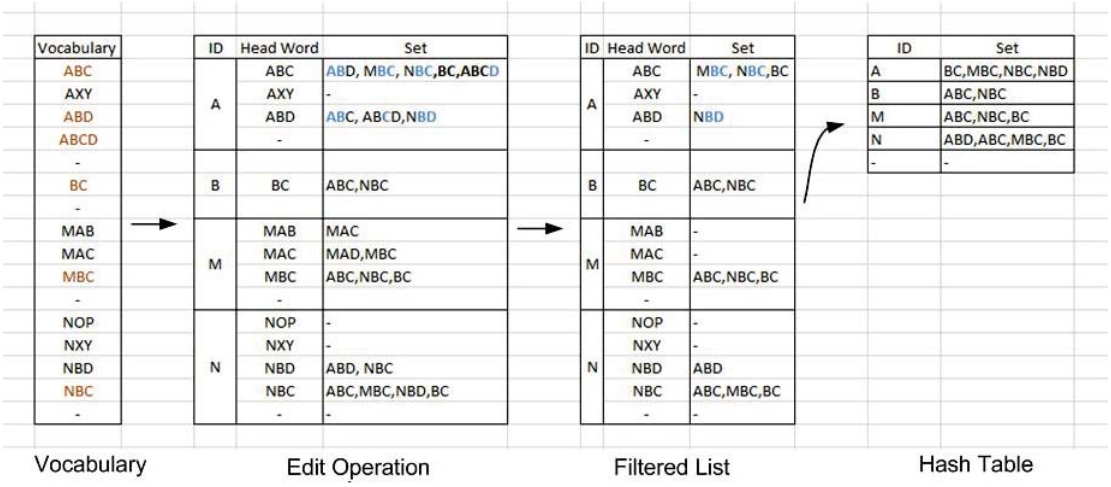
4. Merging: We merge the sets which have the same *Confusion Set Identifier*. Hence, *Confusion Set Identifier A* contains {*MBC*, *NBC*, *BC*, *NBD*} (see Fig. 4.2(b)).

5. Storage: For all words present in the final list, we insert them into a hash table, where "key" is *Confusion Set Identifier* and "value" is words in concatenated form separated by tab.

A merged list in confusion set contains a list of words where error in the first character in some words (those which starts with character pointed by *Confusion Set Identifier*) leads to other valid words in vocabulary. This list is utilized to generate the candidate list for predicting words.



(a) Confusion set for *Head Word* ABC and ABD



(b) Confusion set identifier and its element

Figure 4.2: Illustration of creating of confusion set

4.1.2 Generation of candidate list

The core task of any prediction system is to predict the most relevant words for a word under composition. To do this, we generate a list of candidate words followed by ranking them to consider for the prediction list. In this section, we discuss our approach to generate candidate list. Generation of a list of candidate words can be divided in two stages: i) completion of current word and ii) prediction of next word.

i) Completion of current word: Candidate list generation consults the vocabulary V (which is already known to us, as Section 4.1.1). Note that vocabulary is a list of all unigram words. Suppose, there are η number of such words present in V (see Fig. 4.3(a)). After entering 1^{st} character (say γ) prediction system searches the vocabulary to extract words starting with γ. We assume n_1 number of such words are present in vocabulary which start with γ (see Fig. 4.3(b)). When 2^{nd} character of a word is entered, the chunk is searched within the list of words containing n_1 number of words instead of complete

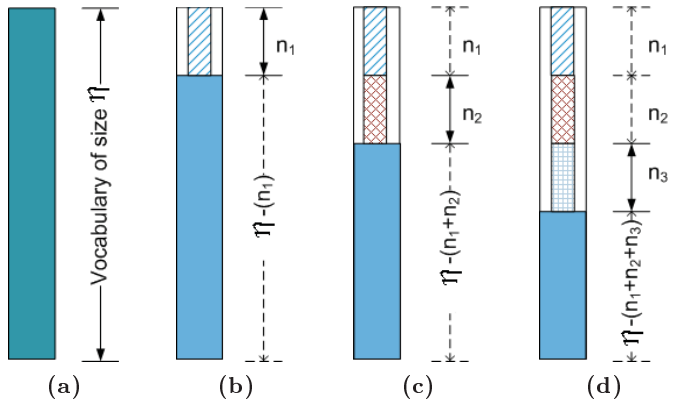


Figure 4.3: Candidate list generation

vocabulary V. This eventually reduces the search time of our method. The above mentioned step retrieves all words which start with γ, however, those words which start with phonetically/graphically similar to γ (let n_2 number of such words exist (Fig. 4.3(c)) are not included into the candidate list generated so far. We identify and add those words into candidate list.

There are situations when a typing error is occurred while entering the 1^{st} character. This can occur as a result of insertion, omission or substitution of a character. It is reported that, 11% real-word and 4% non-word of total errors are occurred at first character for English [161]). Finding suitable candidate words when error occurred at the 1^{st} position require huge computation time as we would need to search with every possible word present in vocabulary. In order to reduce the search time, we use the mechanism as discussed in Section 4.1.1.3 to identify those words which become valid entry in vocabulary. Suppose, n_3 number of such word exists (see Fig. 4.3(d)). We include all n_1, n_2 and n_3 number of words to decide the final list of candidate words. Note that with this approach, the initial search space becomes $n_1 + n_2 + n_3$ number of words instead of η where $\eta \geq n_1 + n_2 + n_3$.

Our approach to generate candidate list is formally stated as Algorithm: 4.3 and its working is described in the following. An initial list (represented as *ini_list*) is used to store candidate words, which is initially empty. We store β (the part of word under composition) as a sequence of Unicode characters. This sequence may lead to a number of ambiguity [30]. Presence of ambiguous character sequence, in fact, increases the miss ratio in word prediction, although words are present in vocabulary. To alleviate this problem, we use Unicode normalization [38]. In addition to this, we remove the occurrence of “Zero-Width Joiner" (ZWJ) and “Zero-Width Non Joiner" (ZWNJ), if any, in the word.

Algorithm 4.3 Candidate words generation

Input: Typed string (β), previous two word (w_{i-1} and w_{i-2}) and language model.
Output: Returns the list of candidates *ini_list*, initially empty.

1: $\widehat{\beta} \leftarrow UnicodeNormalization\,(\beta)$ /* Unicode normalization for β and removal of all the occurrence of "Zero-Width Joiner" (ZWJ) and "Zero-Width Non Joiner" (ZWNJ) */
2: **if** ($Length(\widehat{\beta}) = 1$) **then** /* Case 1: $\widehat{\beta}$ contains only one character */
3: $\gamma_2 \leftarrow CSID(\widehat{\beta})$
4: **for** each $\alpha \in$ *Vocabulary* **do**
5: $\delta \leftarrow CSID(SubString(\alpha, 1))$ /* *CSID* for the 1^{st} character of α */
6: **if** ($\gamma_2 = \delta$) **then**
7: Add α into *ini_list*
8: **end if**
9: **end for**
10: *ini_list* $\leftarrow$ Retrieve and store all words in *Confusion Set Identifier* for $key = \widehat{\beta}$
11: *candidate_list* $\leftarrow$ *ini_list*
12: **end if**
13: **if** ($Length(\widehat{\beta}) > 1$) **then** /* Case 2: $\widehat{\beta}$ contains more then one character */
14: *candidate_list* $\leftarrow \phi$
15: **for** each $\alpha \in$ *ini_list* **do**
16: $\alpha_1 \leftarrow CSID_{seq}(\alpha)$
17: $\beta_1 \leftarrow CSID_{seq}(\widehat{\beta})$
18: **if** ($length.\alpha_1 \geq length.\beta_1$) **then**
19: Store α into *candidate_list*
20: **end if**
21: **end for**
22: *ini_list* $\leftarrow$ *candidate_list*
23: **end if**
24: **if** (Prediction list is selected) **then** /* Case 3: Length of $\widehat{\beta}$ is 0 */
25: Update β, w_{i-1} and w_{i-2}
26: Store words from *Vocabulary* into *candidate_list*
27: **end if**

Let $\widehat{\beta}$ be the Unicode normalized version of β. First, we check for the number of characters present in $\widehat{\beta}$. There can be two cases: $\widehat{\beta}$ contains only one character (Case 1) and more than one characters (Case 2). In Case 1, we extract the first character from α (where α represents a word present in the *Vocabulary* V) and convert to its respective Character Set IDentifer (*CSID*). Similarly, content of $\widehat{\beta}$ is also converted into its respective *CSID*. These converted data are stored as δ and γ_2, respectively. If both γ_2 and δ are the same, then we add α to the *ini_list* as a probable candidate word (see Step $2-7$ in Algorithm 4.3).

Next step is to include words where error in the first position may make a valid word in the vocabulary. Here, we put all the words which belong to the confusion set whose *Confusion Set Identifier*) is same as $\widehat{\beta}$. This step is proposed to reduce the computation overhead at run time. Otherwise, we have to process all words in vocabulary to get desired result. In Case 2, that is, when typed text ($\widehat{\beta}$) contains more than one character, then the candidate list already obtained (that is, *ini_list*) is needed to be further filtered so that most relevant words only can be included. This filtration is done by containing only those words in *ini_list* whose lengths after converting them into $CSID_{Seq}$ are larger than the text entered by the user (See Step $13-23$, Algorithm 4.3). Note that filtering candidate words, in this way, reduces the size of the candidate list.

ii) Prediction of next word: When a target word appears in the prediction list and the user selects the word from this list, we are to predict the next word. In this situation, the value of β, w_{i-1} and w_{i-2} get updated (see Fig. 4.1). That is, content of w_{i-1} is stored into w_{i-2}, and selected word from the list is stored in w_{i-1} followed by adding space into the composed text. The value of β in this case is set to *null* (and hence $\widehat{\beta}$). The *CandidateList* is initialized with all words from *Vocabulary* V. This is because any word present in *Vocabulary* can be a candidate.

Whatever be the case, the *CandidateList* so obtained is passed through a ranking procedure, which is discussed in the next sub section.

4.1.3 Ranking of candidates

The candidate words so obtained should be shortlisted before displaying in the prediction list. This shortlisting need to be done based on various features such as how much the words are similar to each other in terms of graphical and phonetic similarity, presence of typographical error (no error, one error or more than one error), probability of words used in a given context and how recently the word is used. We finally need a mechanism to use these features for prediction. This, in fact, is a non-trivial task. To achieve this, we propose to use Support Vector Machine (SVM) [95] based ranking approach (see Fig 4.5). We accomplish this with two tasks namely i) vector representation of candidates and ii) ranking the candidate vectors.

4.1.3.1 Vector representation

For the vector representation of each candidate word, we are to decide the components with which such a word can be specified. To do this, we propose measurement of four different scores of a candidate word. The scores in our approach are graphical/phonetic

similarity, typographical error, word probability and how recently a word is used. In the following, we define the score metrics and also discuss the methods of calculating them.

1) ED_{Equal}*:* Given two words α and β, the score ED_{Equal} (α, β) is defined as the minimum number of edit operations needed to transform β into a part of α, with the allowed edit-operations being insertion, deletion, or substitution of a character. It may be noted that the calculation of ED_{Equal} follows *Levenshtein Edit Distance* [41] but they are not necessarily same. For example, ED_{Equal} between *"similarity"* and *"semi"* is one whereas, Levenshtein Edit Distance is six (see Table: 4.3). Note that ED_{Equal} and Levenshtein Edit Distance between two words is same, if both words are of equal length.

2) E_{Score}*:* Given a typed word (β) and target word (α), $E_{Score}(\alpha, \beta)$ is defined as the error within the typed word (β) and target word (α) in a scale of $[0-1]$. Lower E_{Score} means word are more similar.

$$E_{Score}(\alpha, \beta) = \left(\frac{ED_{Equal}(\alpha, \beta)}{Length(\beta)} \right) \tag{4.5}$$

3) B_{Score}*:* Given a target word (α) and previous word sequence (w_{i-1}^{i}), B_{Score} is defined as the absolute value of trigram backoff probability in logarithmic (base 10) scale of occurrence of α from the given word sequence w_{i-1} and w_{i-2}.

$$B_{Score} = \left| Log_{10}\left(\widehat{P}\left(\alpha | w_{i-1} w_{i-2}\right) \right) \right| \tag{4.6}$$

4) P_{Score}*:* It is defined as a character similarity between typed word (β) and desired word (α) in a scale of $[0-1]$ and can be computed using Eqn. 4.7 where character similarity means phonetically or graphically similarity. Lower value of P_{Score} indicates more similarity.

$$P_{Score} = E_{Score}(\alpha_1, \beta_1) \tag{4.7}$$
$$where, \alpha_1 = CSID_{seq}(\alpha) \; and \; \beta_1 = CSID_{seq}(\beta)$$

5) L_{Score}*:* Given a target word (α), L_{Score} is defined as a score indicating how recently the word is used. It can be calculated using Eqn 4.8, where T_{cts} denotes current time

Table 4.3: ED_{Equal} calculation between *similarity* and *semi*

	-	S	I	M	I	L	A	R	I	T	Y
-	0	1	2	3	4	5	6	7	8	9	10
S	1	0	1	2	3	4	5	6	7	8	9
E	2	1	1	2	3	4	5	6	7	8	9
M	3	2	2	1	2	3	4	5	6	7	8
I	4	3	2	2	1	2	3	4	4	5	6

stamp and T_{pts} is previously used time stamp.

$$L_{Score} = \left| Log_{10} \left(\frac{T_{cts} - T_{pts}}{T_{cts}} \right) \right| \tag{4.8}$$

The feature vector $\overrightarrow{z}$ consisting of P_{Score}, E_{Score}, B_{Score} and L_{Score} for a candidate word α with reference to a current word under composition $\widehat{\beta}$ can be represented as shown in Eqn 4.9. We use the feature vectors of all candidate words for ranking.

$$\overrightarrow{z_{\alpha,\widehat{\beta}}} = (P_{Score})\,\widehat{i} + (E_{Score})\,\widehat{j} + (B_{Score})\,\widehat{k} + (L_{Score})\,\widehat{l} \tag{4.9}$$

4.1.3.2 Ranking

Learning a ranking function is used in various applications in information retrieval and machine learning. It is distinguished from that of learning classification problem [94,234] as follows. A training set in classification problem is a set of data objects (i.e. feature vector) and their class labels and outputs a distinct class for a given feature vector. Whereas, *ranking function* is an ordering of feature vectors and outputs a score for each feature vector, from which a global ordering of feature vectors is constructed. For example, let "A is preferred to B" be specified as $A \succ B$. A training set for ranking function is denoted as $R = (\overrightarrow{x_i}, y_i), ..., (\overrightarrow{x_m}, y_m)$ where y_i is the ranking of $\overrightarrow{x_i}$, that is, $y_i < y_j$ if $\overrightarrow{x_i} \succ \overrightarrow{x_j}$. The objective is to find a function F which output scores such that $F(\overrightarrow{x_i}) > F(\overrightarrow{x_j})$ for $\overrightarrow{x_i} \succ \overrightarrow{x_j}$.

Ranking support vector machine improves the generalization performance by maximizing the minimal ranking difference [94, 234]. For example, consider the two linear ranking functions $F_{\overrightarrow{w_2}}$ and $F_{\overrightarrow{w_3}}$. The linear ranking function $F_{\overrightarrow{w}}$ projects feature vectors onto a weight vector $\overrightarrow{w}$. For instance, Fig.4.4 illustrates linear projections of five feature vectors $\overrightarrow{x_1}$, $\overrightarrow{x_2}$, $\overrightarrow{x_3}$, $\overrightarrow{x_4}$ and $\overrightarrow{x_5}$ onto three different weight vectors $\overrightarrow{w_1}$, $\overrightarrow{w_2}$ and $\overrightarrow{w_3}$, in a two-dimensional space. In this example, both $F_{\overrightarrow{w_2}}$ and $F_{\overrightarrow{w_3}}$ make the same ordering for the five vectors (i.e. $x_1 \succ x_2 \succ x_3 \succ x_4 \succ x_5$) whereas, ordering of $F_{\overrightarrow{w_1}}$ is ($x_1 \succ x_3 \succ x_2 \succ x_5 \succ x_4$) different from $F_{\overrightarrow{w_2}}$ and $F_{\overrightarrow{w_3}}$. Here, an ordering implies the sequence of points of projections from vectors on the weight vector. In this example, δ_1, δ_2 and δ_3 represent distance between closest vector on $\overrightarrow{w_1}$, $\overrightarrow{w_2}$ and $\overrightarrow{w_3}$, respectively (see Fig. 4.4). Although the two weight vectors $\overrightarrow{w_2}$ and $\overrightarrow{w_3}$ make the same ordering, but $\overrightarrow{w_3}$ generalizes better than $\overrightarrow{w_2}$ because the distance between the closest vectors on $\overrightarrow{w_3}$ (i.e. δ_3) is larger than that on $\overrightarrow{w_2}$ (i.e. δ_2). This way, ranking support vector machine computes the weight vector $\overrightarrow{w}$ that maximizes the differences of closest data pairs in ranking.

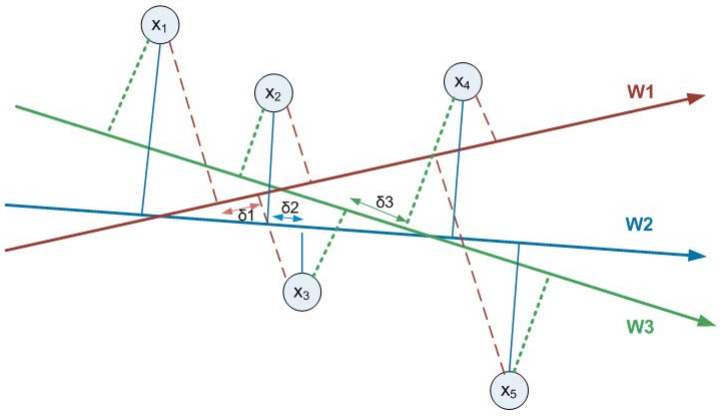


Figure 4.4: Vector representation for generalization

The ranking function F on a new vector z can be computed using Eqn 4.10 and 4.11, where $\overrightarrow{w}$ is weight vector. In Eqn 4.11, ϕ_1, ϕ_2, ϕ_3, ϕ_4 are the components of weight vector $\overrightarrow{w}$. These values are learned with training data set.

$$F(\overrightarrow{z_{\alpha,\beta}}) = \overrightarrow{w}.\overrightarrow{z_{\alpha,\beta}} \tag{4.10}$$

$$\overrightarrow{w} = (\phi_1)\,\widehat{i} + (\phi_2)\,\widehat{j} + (\phi_3)\,\widehat{k} + (\phi_4)\,\widehat{l} \tag{4.11}$$

In our system, we compute feature vector $\overrightarrow{z_{\alpha,\beta}}$ for each word (α) present in the candidate list. A block diagram illustrating ranking steps is shown in Fig. 4.5. An algorithm to rank the candidates in the candidate list is precisely stated in Algorithm:4.4.

According to Algorithm 4.4, for each candidate word α, we calculate its score values and store them into *final_list* (Step 1−10 in Algorithm 4.4). For the situation when user selects a word from the prediction list, the proposed prediction methodology predicts the next word (i.e. prediction system is in the stage of next word prediction, based on B_{Score} of the candidates only (See between Step 3 and Step 12 in Algorithm 4.4). The "ordering of candidate list" is then done on this *final_list*. In other words, ranking is performed by sorting the *final_list* in ascending order and select top ψ words (α only) from this list and display them in the prediction window. Here, ψ denotes the size of the prediction window.

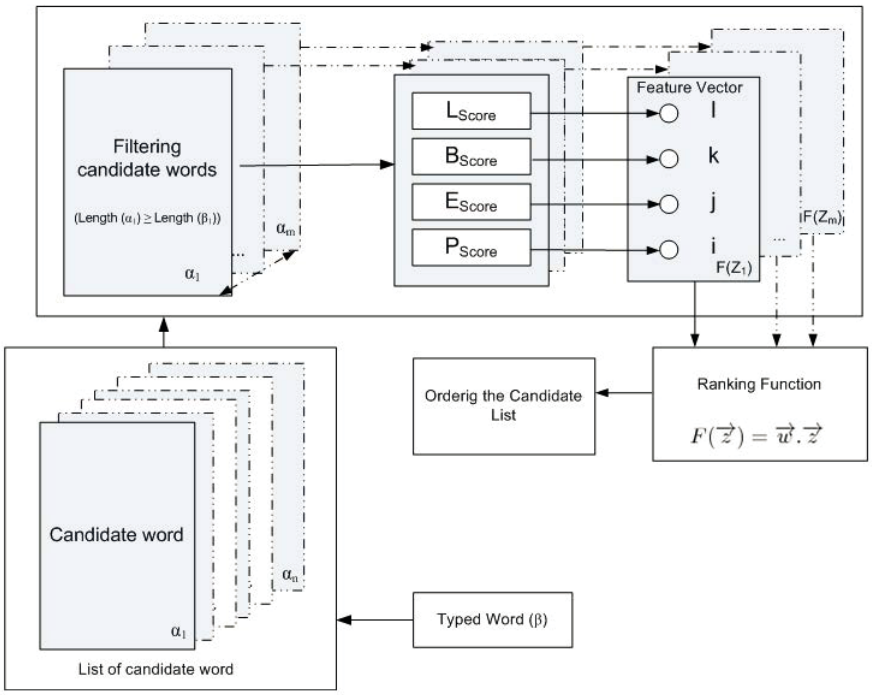


Figure 4.5: Overview of the proposed ranking method

4.1.4 Development of prediction systems in Hindi, Bengali and Telugu

To test the efficacy of our approach, two word prediction systems for Hindi. Bengali and Telugu have been developed. Bengali language has a number of similarity with other Indic languages namely Odiya, Assamese, Manipuri while Hindi is with Marathi, Gujrati, Punjabi, Bhojpuri etc. More significantly, a vast population (around 3.11% and 4.70% of the total population [213]) use these three languages for their reading/writing. In this section, we discuss the development of prediction systems for these three languages. Initially, we develop our resources from *Wikipedia* corpus (written in Unicode; the download links are provided in Table 4.1). We use "hiwiki-latest-pages-articles.xml", "bnwiki-latest-pages-articles.xml" and "tewiki-latest-pages-articles.xml" files for Hindi, Bengali and Telugu text corpus, respectively. We filter the *Wikipedia* corpus to remove all non-relevant elements. The extracted texts are then "Normalised" [38] and we remove the occurrences of "Zero-Width Joiner" and "Zero-Width Non Joiner" [40]. We develop language models for Hindi, Bengali and Telugu using CMU Toolkit [35] for both desktop and mobile text entry method. or Hindi, our language model contains 65001 unigram (i.e. 65000 vocabulary words and one symbol for out-of-vocabulary (OOV) word; the size of file for desktop and mobile are same as 2.92 MB), 1 million bigrams (for desktop device,

Algorithm 4.4 Score calculation and generation of feature vector

Input: Normalized typed word ($\widehat{\beta}$), previous typed words w_{i-1} and w_{i-2}, language model and a ranking function F
Output: Returns scores of candidate words as *final_list*

1: **for** each $\alpha \in$ *candidate_list* **do**
2: $B_{Score} = \left| Log_{10}\left(\widehat{P}\left(\alpha | w_{i-1} w_{i-2}\right)\right)\right|$ /* Calculate B_{Score} */
3: **if** $(Length(\widehat{\beta}) > 0)$ **then** /* Case 1: Prediction while word completion*/
4: Compute P_{Score} between α and $\widehat{\beta}$
5: Compute E_{Score} between α and $\widehat{\beta}$
6: Compute L_{Score} between α and $\widehat{\beta}$
7: /* Formulation of Feature Vector */
8: $\overrightarrow{z} \leftarrow (P_{Score})\,\widehat{i} + (E_{Score})\,\widehat{j} + (B_{Score})\,\widehat{k} + (L_{Score})\,\widehat{l}$
9: $\theta \leftarrow$ Calculate $F(\overrightarrow{z})$ for α and $\widehat{\beta}$
10: Add θ and α into *final_list*
11: Add B_{Score} and α into *final_list* /* Case 2: Next word prediction*/
12: **end if**
13: **end for**

52.5 MB and mobile system, 35.7 MB) and 2.1 million trigrams backoff probability (143 MB versus 52 MB). Bengali resource model consists of 65001 unigram (file size same, 2.51 MB), 1.3 million bigrams (49.2 MB for desktop device and 33.9 MB for mobile) and 2.4 million trigrams backoff probability (104 MB for desktop and 55 MB for mobile). Telugu resource model contains 65001 unigram (file size 2.66 MB), 1.2 million bigrams (38.2 MB) and 2.2 million trigrams back-off probability (60MB). The language models also contain their back-off weights.

We have developed the lists of graphical and phonetic similar characters for Hindi, Bengali and Telugu (see Hindi and Bengali language results in Table 4.4). In Table 4.4, C, $iTrans$ and $CSID$ represent the “characters", “iTrans-encoded text” and “Character Set IDentifier”, respectively. Note that *halant* is considered as the special character in both the languages.

Following the procedure described in Section 4.1.1.3, we have computed the confusion sets for Hindi and Bengali. The intermediate results of confusion pair identification, their filtration and final confusion set generation are illustrated in Table 4.5.

Calculation of score values of a given candidate list and subsequent ranking with reference to Hindi is illustrated in the following (also see Table 4.6).

CSID		Similarity set							
C	iTrans	C	iTrans	C	iTrans	C	iTrans	C	iTrans
অ	[a]	অ	[a]	আ	[aa]	া	[a]		
ই	[i]	ই	[i]	ঈ	[ii]	ি	[i]	ী	[ii]
উ	[u]	উ	[u]	ঊ	[uu]	ু	[u]	ূ	[uu]
ঋ	[Rî]	ঋ	[Rî]		[Rî]				
এ	[e]	এ	[e]	ঐ	[ai]	ে	[e]	ৈ	[ai]
ও	[o]	ও	[o]	ঔ	[au]	ো	[o]	ৌ	[au]
ক	[ka]	ক	[k]	খ	[kha]				
গ	[ga]	গ	[ga]	ঘ	[gha]				
চ	[ca]	চ	[c]	ছ	[cha]				
জ	[ja]	জ	[ja]	য	[J]	ঝ	[jha]		
ঙ	[nga]	ঙ	[nga]	ঞ	[nja]				
ট	[Ta]	ট	[Ta]	ঠ	[Tha]				
ড	[Da]	ড	[Da]	ড়	[Dhha]	ঢ	[Dha]	ঢ়	[Dhha]
ত	[ta]	ত	[ta]	থ	[tha]				
দ	[da]	দ	[da]	ধ	[dha]				
ন	[na]	ন	[na]	ণ	[Na]	ং	anusvāra		
প	[pa]	প	[pa]	ফ	[pha]				
ব	[ba]	ব	[ba]	র	[ra]	ভ	[bha]		
ম	[ma]	ম	[ma]						
য	[ja]	য	[ja]	য়	[ya]				
ল	[la]	ল	[la]						
শ	[sha]	শ	[sha]	ষ	[Sha]	স	[sa]		
হ	[ha]	হ	[ha]						
়	nukta	়	nukta						
্	*halant*	্	*halant*						

(a) Bengali

CSID		Similarity set							
C	iTrans	C	iTrans	C	iTrans	C	iTrans	C	iTrans
अ	[a]	अ	[a]	आ	[aa]	ा	[a]		
इ	[i]	इ	[i]	ई	[ii]	ि	[i]	ी	[ii]
उ	[u]	उ	[u]	ऊ	[uu]	ु	[u]	ू	[uu]
ऋ	[Rî]	ऋ	[Rî]		[Rî]				
ए	[e]	ए	[e]	ऐ	[ai]	ॅ	[e]	ै	[ai]
ओ	[o]	ओ	[o]	औ	[au]	ॉ	[o]	ौ	[au]
अं	[a.n]	अं	[a.n]	अँ	[a.n]				
क	[ka]	क	[k]	ख	[kha]				
ग	[ga]	ग	[ga]	घ	[gha]				
च	[ca]	च	[c]	छ	[cha]				
ज	[ja]	ज	[ja]	झ	[jha]				
ट	[Ta]	ट	[Ta]	ठ	[Tha]				
ड	[Da]	ड	[Da]	ङ	[Ña]	ढ	[Dha]		
त	[ta]	त	[ta]	थ	[tha]				
द	[da]	द	[da]	ध	[dha]				
न	[na]	न	[na]	स	[ñ]	ण	[Na]	ं	anusvāra
प	[pa]	प	[pa]	फ	[pha]				
ब	[ba]	ब	[ba]	व	[va]	भ	[bha]		
म	[ma]	म	[ma]						
य	[ya]	य	[ya]						
र	[ra]	र	[ra]						
ल	[la]	ल	[la]						
श	[sha]	श	[sha]	ष	[Sha]	स	[sa]		
ह	[ha]	ह	[ha]						
़	nukta	़	nukta						
्	*halant*	्	*halant*						

(b) Hindi

Table 4.4: Graphical or phonetic similar sets of characters

ID	Head Word	List of confusion pair				
उ [u]	T_1 = उतार [utaara]	उतारा [utaaraa]	उतारी [utaarii]	उतारू [utaaruu]	उतारे [utaare]	...
		उधार [udhaara]	उभार [ubhaara]	कतार [kataara]	तार [taara]	...
	T_2 = उगाने [ugaane]	उगने [ugane]	उगाते [ugaane]	उगाये [ugaaye]	उठाने [uThaane]	उडाने [uDaane]
		गाने [gaane]	जगाने [jagaane]	भगाने [bhagaane]	लगाने [lagaane]	...
	T_3 = उठा [uThaa]	उठाई [uThaaii]	उठाए [uThaae]	उठाना [uThaanaa]	उठानी [uThaanii]	उठाने [uThaane]
		उडान [uDaana]	ठान [Thaana]	पठान [paThaana	...	...

(a) Identification of confusion pair

ID	Head Word	Confusion set				
उ [u]	T_1	कतार [kataara]	तार [taara]	...		
	T_2	गाने [gaane]	जगाने [jagaane]	भगाने [bhagaane]	लगाने [lagaane]	...
	T_3	ठान [Thaana]	पठान [paThaana]	...		

(b) Filtration of confusion pair

ID	Confusion set				
उ [u]	कतार [kataara]	तार [taara]	गाने [gaane]	जगाने [jagaane]	भगाने [bhagaane]
	लगाने [lagaane]	ठान [Thaana]	पठान [paThaana]	...	

(c) Final confusion set

Table 4.5: Generation of confusion set: An illustration

A part of *CandidateList* is shown in Table 4.6(a). Calculation of different scores on these words is shown in Table 4.6(b). Let us call this list as *ScoreList*. This list stores the calculated scores and is used by the ranking module. Let α_i is the i^{th} word belonging to *CandidateList*. Suppose, at any instant, normalized input sequence ($\widehat{\beta}$) is सनगरह [sanagaraha] for which candidate words are: α_i is संगठित [sa.ngaThita], α_{i+1} is संगृहीत [sa.ngRîhiita], α_{i+3} as संग्रहालय [sa.ngrahaalaya] etc. (see Table 4.6a) and user wants to compose the word α_{i+3} as संग्रहालय [sa.ngrahaalaya]. It may be noted that errors between composed word chunk सनगरह [sanagaraha] (composed as स + न + ग + र + ह) and required word संग्रहालय [sa.ngrahaalaya] (composed as स + ं + ग + ् + र + ह + ा + ल + य) are misuse of न [na] for anusvāra (ं) and omission of halant (्).

We calculate E_{Score} between α_i and $\widehat{\beta}$ (using Eqn 4.5). To do this, we need to compute ED_{Equal} between α_i and $\widehat{\beta}$. The value of E_{Score} is 0.6 for α_i. Similarly, E_{Score} values with $\widehat{\beta}$ and α_{i+1}, α_{i+2} ... α_{i+9} are computed as shown in Table 4.6(b). Next, we calculate P_{score} between α_i and $\widehat{\beta}$, which uses the $CSID_{Seq}$ (see Table 4.6). The P_{score} value between α_i and $\widehat{\beta}$ is 0.4. Likewise, P_{score} values between $\widehat{\beta}$ and α_{i+1}, α_{i+2} ... α_{i+9} are computed. B_{Score} is calculated taking the backoff probability of α_i, $1.410E-05$ in this case (using Eqn: 4.6). Similarly, we compute the L_{Score} for α_i which is $1.580E-03$. The different score metrics for α_{i+1}, α_{i+2} ... α_{i+9} are shown in Table 4.6b. Further, we compute $F(\overrightarrow{z})$ using Eqn. 4.10. The different values of $F(\overrightarrow{z})$ are given in Table 4.6(c).

α	Candidate word	Length(α_i)	$CSID_{Seq}$
α_i	संगठित [sa.ngaThita]	6	शनगटइत [shanagaTaita]
α_{i+1}	संगृहीत [sa.ngRîhiita]	7	शनगऋहइत [shanagaRîhaita]
α_{i+2}	संग्रह [sa.ngraha]	6	शनगरह [shanagaraha]
α_{i+3}	संग्रहालय [sa.ngrahaalaya]	9	शनगरहअलय [shanagarahaalaya]
α_{i+4}	संग्रहालयों [sa.ngrahaalayo.n]	11	शनगरहअलयओन [shanagarahaalayaona]
α_{i+5}	संग्राम [sa.ngraama]	7	शनगरअम [shanagaraama]
α_{i+6}	संग्राहक [sa.ngraahaka]	8	शनगरअहक [shanagaraahaka]
α_{i+7}	संघर्षविराम [sa.ngharShaviraama]	11	शनगरशबइरअम [shanagarashabairaama
α_{i+8}	संदर्भों [sa.ndarbho.n]	8	शनदरबओ [shanadarabao]
α_{i+9}	सागरतटों [saagarataTo.n]	8	शागरतटओन [shaagarataTaona]

(a) A part of candidate list and their $CSID_{Seq}$

α	Candidate word	ED_{Equal}	P_{Score}	E_{Score}	B_{Score}	L_{Score}
α_i	संगठित [sa.ngaThita]	3	0.4	0.6	4.851	7.199
α_{i+1}	संगृहीत [sa.ngRîhiita]	2	0.2	0.4	6.135	6.599
α_{i+2}	संग्रह [sa.ngraha]	3	0.0	0.6	4.124	1.517
α_{i+3}	संग्रहालय [sa.ngrahaalaya]	3	0.0	0.6	4.549	0.512
α_{i+4}	संग्रहालयों [sa.ngrahaalayo.n]	3	0.0	0.6	6.343	0.786
α_{i+5}	संग्राम [sa.ngraama]	3	0.2	0.6	4.772	1.178
α_{i+6}	संग्राहक [sa.ngraahaka]	3	0.2	0.6	6.867	7.199
α_{i+7}	संघर्षविराम [sa.ngharShaviraama]	3	0.2	0.6	5.198	7.169
α_{i+8}	संदर्भों [sa.ndarbho.n]	3	0.4	0.6	4.665	0.941
α_{i+9}	सागरतटों [saagarataTo.n]	2	0.4	0.4	7.167	8.198

(b) Calculating score for $\widehat{\beta}$= सनगरह (sanagaraha) and w_i=संग्रहालय (sa.ngrahaalaya)

α	P_{Score}	E_{Score}	B_{Score}	L_{Score}	$F(\overrightarrow{z})$	*Ranked Word*
α_{i+3}	0.000	0.600	4.549	0.512	260.221	संग्रहालय [sa.ngrahaalaya]
α_{i+2}	0.000	0.600	4.124	1.517	263.507	संग्रह [sa.ngraha]
α_{i+4}	0.000	0.600	6.343	0.786	289.445	संग्रहालयों [sa.ngrahaalayo.n]
α_{i+5}	0.200	0.600	4.772	1.178	453.676	संग्राम [sa.ngraama]
α_{i+1}	0.200	0.400	6.135	6.599	462.997	संगृहीत [sa.ngRîhiita]
α_{i+7}	0.200	0.600	5.198	7.196	517.159	संघर्षविराम [sa.ngharShaviraama]
α_{i+6}	0.200	0.600	6.867	7.199	542.200	संग्राहक [sa.ngraahaka]
α_{i+8}	0.400	0.600	4.665	0.941	633.620	संदर्भों [sa.ndarbho.n]
α_{i+9}	0.400	0.400	7.167	8.198	677.353	सागरतटों [saagarataTo.n]
α_i	0.400	0.600	4.851	7.199	696.091	संगठित [sa.ngaThita]

(c) Ranking words for $\widehat{\beta}$= सनगरह (sanagaraha) and w_i=संग्रहालय (sa.ngrahaalaya)

Table 4.6: Generation of confusion set: An illustration (in Hindi)

In Table 4.6(c), we can observe that final score for the word संग्रहालय [sa.ngrahaalaya] is 36.9144 (ranked 2) whereas for the word संगृहीत [sa.ngRîhiita] and संगठित [sa.ngaThita] are 63.5125 (ranked 4) and 114.7380 (ranked 10), respectively. The desired word संग्रहालय [sa.ngrahaalaya], ranked at top, which is compared to other words in vocabulary for β= सनगरह [sanagaraha]. Finally, to rank the most probable words, we sort the *ScoreList* in ascending order and select top ψ words (α only) from this list and display them in the prediction window. Here, ψ denotes the size of the prediction window.

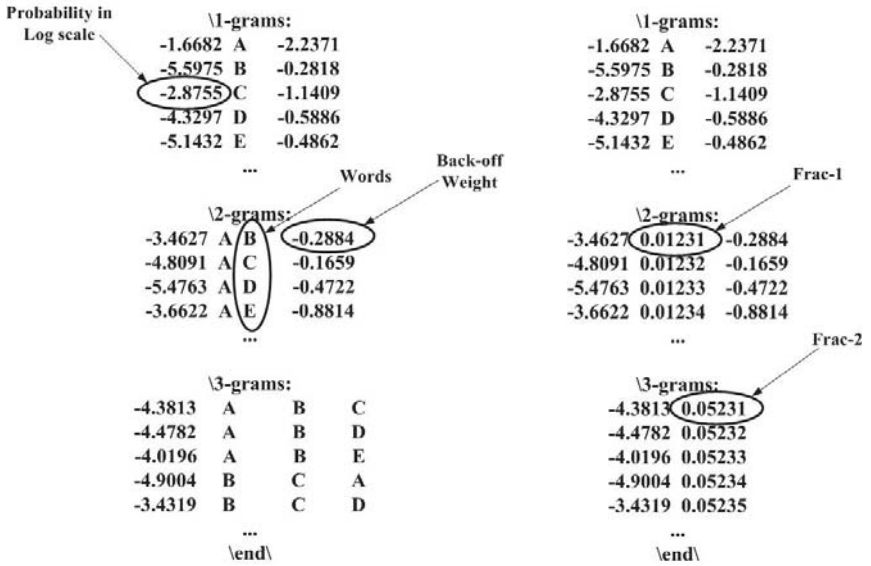
Our proposed word prediction method with error correction support for Indic languages cannot be directly deployed in a mobile device with limited configuration. We have adapted the technique with necessary modification so that the technique can work in mobile devices as well.

As a solution, we implement *Arithmetic coding* [224], an efficient data compression technique, to compress the size of the language files containing word-level unigram, bigram and trigram entries. *Arithmetic coding* is a form of entropy encoding used in lossless data compression. Normally, a string of characters such as the words "hello world" is represented using a fixed number of bits per character, as in the ASCII code. When a string is converted to arithmetic encoding, frequently used characters are stored with fewer bits and not-so-frequently occurring characters are stored with more bits, resulting in fewer bits used altogether [224]. Note that Arithmetic coding encodes the entire input data into a single number, which is a rational number between $0.0-1.0$ [224].

How the Arithmetic coding can be applied to our approach is explained with an illustration. A sample snippets of a language model file (stored in Arpa style [35]) containing unigram, bigram and trigram word sequences are shown in Fig. 4.6(a). It may be noted that the unigram and bigram file structure contain three entries, the probability values in log scale, the character (for unigram) or pair of characters (for bigram) and the back-off probability scores. Whereas, the trigram entry has two parts, the probability and sequence chunks. In Fig. 4.6(a), usually text is stored in UTF-16 format, where it requires two bytes of memory space for each character whereas for a number, it is one byte only). So, if we apply *Arithmetic coding* to each bigram and trigram character chunk and convert them into numbers (e.g. see *Frac-1* and *Frac-2* in Fig. 4.6(b)), we can save enough memory space.

A *Hash table* can be maintained to retrieve exact word sequence for a given code value. In other words, after compression, a word sequence say $w_1w_2w_3$ as a trigram sequence is represented by a rational number with 64 bits (eight bytes) instead of $(3 * 5.1 * 16 + 2 = 247$ bits, that is, ≈ 31 bytes. The structure of file we propose to store a language model is shown in Table 4.7. Here, *prob* and *wt* represents the backoff probability in log scale and their corresponding backoff weights (*wt*). The structure of *Dictionary class* (for Hash table) is also mentioned in Table 4.7 (val is calculated as $< backoffprobability > +"/t" + wt$).

Following the proposed prediction method, for each word in candidate word list and last two typed words $w_{i-2}w_{i-1}$ (identified from interface *text entry area*), we transform them into arithmetic code and search them into modified dictionary when required. The data in dictionary is already converted into arithmetic code at preprocessing level.



(a) Arpa original **(b)** Arpa compressed

Figure 4.6: Basic structure of *arpa* file

Table 4.7: Structure of Language Model

Description	Without compression	With compression	Dictionary structure
Unigram	<prob word wt>	<prob word wt>	$< [word], [val] >$
Bigram	<prob w_1 w_2 wt>	<prob $frac_1$ wt>	$< [frac_1], [val] >$
Trigram	<prob w_1 w_2 w_3 >	<prob $frac_2$ >	$< [frac_2], [val] >$

Design rationale: We have aimed to run an efficient word prediction system meant for Indic languages in a memory and speed constrained mobile handset. In our work, we propose to implement the word-level text prediction strategy proposed in [174]. However, the implementation of [174] requires at least $\approx$ 100 MB to store the language resource model. Typically, the mobile devices are equipped with memory in the order of $1-4$ GB. To alleviate this limitation, we propose the encoded form of the language resource model encoding with arithmetic coding. Our encoded language model files, for example, in Hindi the bigram and trigram files require only 44 MB and 60 MB, respectively. Thus, these files will be easily loaded into primary memory for the use in prediction program.

Proposed word-level prediction methods are augmented with *iLiPi* keyboard suitable for desktop device (Fig. 4.7) and *mLiPi* keyboard (Fig. 4.8) for mobile device (keyboard descriptions are provided in Chapter 3). Here, *iLiPi* and *mLiPi* are developed as front end of text entry systems and the proposed word-level prediction methods are implemented at the back end.

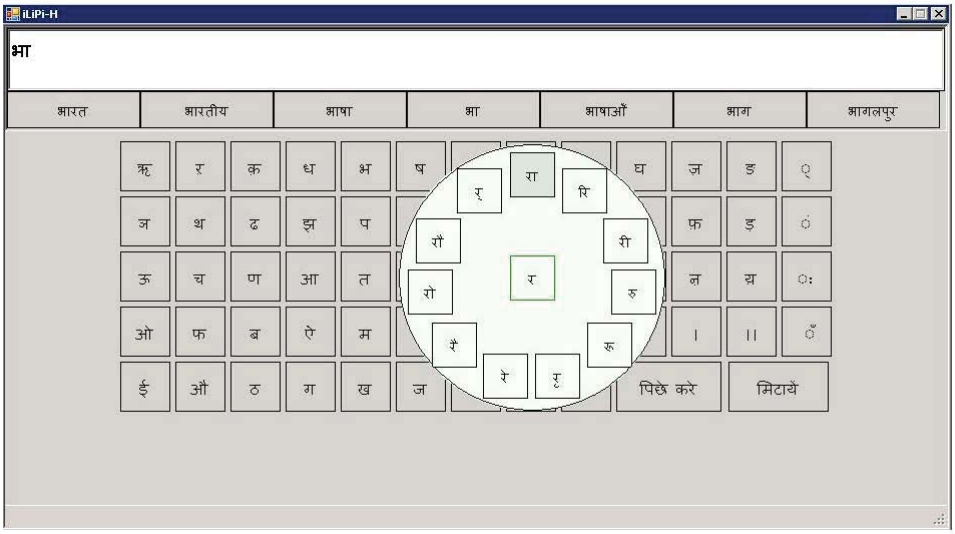


Figure 4.7: *iLiPi* keyboard layout augmented with word prediction

The text entry task through our proposed mobile interface can be illustrated with a practical example (see Fig. 4.8). Suppose user wants to compose ভারত [bharat] with the *mLiPi* Bengali interface. First, he selects the panel where character ভ [bh] resides (exploration of that panel is as shown in Fig. 4.8(a)). This selection costs two keystrokes. Then, user observes that the next intended character া [a] matra is in the next character list (Fig. 4.8(b)), which he selects then. This selection saves a keystroke (instead of exploring the *matra* panel and then pressing the character). After composing ভা [bha] properly, it can be observed that intended word ভারত [bharat] is displayed in the list (Fig. 4.8(c)). Selection of the word takes one keystrokes whereas composing it character by character takes another four keystrokes (two for corresponding panel selection and two for character selection). So, this optimum way of selecting word ভারত [bharat] saves $(8 - 4 = 4)$ keystrokes.

4.2 Experimental setup and results

To judge the efficacy of the proposed approach, we have done both user evaluation and simulated evaluation for desktop device and user evaluation for mobile devices. In this section, we discuss the experiments and experimental setup, evaluation procedure and results observed for both the interfaces.

(a) Selecting panel where ভ resides

(b) After the selection of ভ , next character া is displayed in the character list

(c) After composing ভা, intended word ভারত is displayed in the prediction list

Figure 4.8: Composing word ভারত through *mLiPi* Bengali interface

4.2.1 Apparatus

In case of desktop device, we conducted experiments in 2.0 GHz - 2.6 GHz Pentium Core2Duo machines with HP w17e 17-inch wide screen LCD color monitors. The different screen resolutions are 1440×900 and 1280×768. The proposed word prediction system is developed in C# using Visual Studio 2010. All experiments are carried out in Windows 7 or Windows XP environment. Whereas, mobile device based experiments were conducted with a number of mobile handsets such as Nokia Lumia 525 smartphone (Windows Mobile), Windows Mobile 7 and Windows phone 8 operating system, Microsoft.NET framework 4.0 for Visual C#.NET 2010 as development framework.

4.2.2 Metrics for performance measure

Text entry rate (wpm) and error rate: The formula to calculate text entry rate (in WPM) and total error rate [128] as addition of corrected and uncorrected error rate are already described in Chapter 3.

Keystroke savings (KS): Keystroke savings refer to the percentage of the keystrokes that a user can save using a word prediction system [55,230]. According to this measure, the keystroke savings can be expressed as shown in Eqn. 4.12.

$$KS = \left(1 - \frac{(K_w + K_s)}{T_{total}}\right) \times 100 \tag{4.12}$$

Here, T_{total} denotes the total texts, K_w is the number of keystrokes required to type a given text and K_s indicates number of times words are selected from prediction window.

Potential keystroke savings (PKS): PKS is defined as the keystroke savings obtained by users if they fully utilize the word prediction system [55]. That is, for every word present in a trial text, the user selects the word as soon as it appears in the prediction list. It may be noted that *KS* returns the keystroke savings actually accomplished by the user, and it varies from user to user. Further, the *KS* is less than or equal to *PKS*, because a user may or may not check the prediction window after every character pressing.

Prediction utilization (PU): This metric defines as how much a user depends on the word prediction system at the time of the text composition [55]. It is calculated as the ratio of *KS* and *PKS*.

Hit rate (HR): It is the percentage of times target words appear in the prediction list while composing the text.

4.2.3 Participants

To evaluate the *iLiPi* and *mLiPi* word prediction systems as well as the existing word prediction systems in Hindi, Bengali and Telugu, 32 (8 female and 24 male, average age 26.79 years), 30 (11 female and 19 male, average age 26.77 years) and 28 (6 female and 22 male, average age 25.20 years) users for Hindi, Bengali and Telugu respectively, are recruited from the locality surrounding the university campus. The participants are having different educational backgrounds and computer proficiency. Their educational backgrounds vary from higher secondary to post graduate level. Consequently, all participants are language literate. 27 participants in Hindi, 19 in Bengali and 18 in Telugu are having familiarity in typing text in English as well as their mother languages.

4.2.4 Text under test

We selected five different benchmarked texts in Hindi, four in Bengali and three in Telugu for testing. The details about these benchmark texts are presented in Table 4.8. Here, $TH_1 - TH_5$ represent Hindi texts, $TB_1 - TB_4$ denote Bengali benchmark texts and $TT_1 - TT_4$ indicate Telugu texts. These texts are classified as *in-domain* and *out-of-domain* data. As we have taken *Wikipedia* corpus to train our systems (discussed in

Table 4.8: Description of benchmark texts

Text under test	Reference text	Type of text	Number of sentences	Number of words
TH_1	*Satya ke prayog* by M. K. Gandhi	Autobiography	19	257
TH_2	*Patrakarita: ek parichay* by Sandip Kumar Srivastava	Journalism article	18	243
TH_3	*Phoolbaari* by Vijaydan Detha	Story book	22	312
TH_4	*Rangbhoomi* by Munsi Premchand	Novel	20	279
TH_5	*Wikipedia*	Miscellaneous	21	298
TB_1	*Bhuto* by Satyajit Ray	Short story	22	284
TB_2	*Bangal Nama* by Tapan Roy Choudhury	Autobiography	24	288
TB_3	*Pather Dabi* by Sharat Chandra Chattopadhyay	Novel	20	263
TB_4	*Wikipedia*	Miscellaneous	19	259
TT_1	*Bhagavatha Purana* by Bammera Pothana	Novel	22	273
TT_2	*Vishwambhara* by C. Narayanareddy	Story	19	251
TT_3	*Kanyaasulkamu* by Gurajada Apparao	Drama	18	245
TT_4	*Wikipedia*	Miscellaneous	17	235

Section 4.1.1), benchmark texts which are the parts of *Wikipedia* are treated as *in-domain* texts. All other benchmark texts are treated as *out-of-domain* texts.

4.2.5 Procedure

Two different approaches for evaluation have been considered namely user evaluation and simulated evaluation [59, 105, 200, 214].

4.2.5.1 User evaluation

For this type of evaluation, we followed two approaches for typing text namely *Read and Type* (RT) and *Listen and Type* (LT). Thirteen benchmark texts were selected from popular literatures of three languages for evaluation. For the *RT* method, we provided the benchmark texts in printed form and asked participants to type them as quickly and accurately as possible. In *LT* based evaluation, users first listened words uttered by instructor and then typed those words. Four systems were selected for evaluation in case of desktop devices, *iLiPi* system with and without prediction support, *Lipik* [124] and *IME* [69]. On the other hand, *Panini* and *mLiPi* system with prediction are taken for

user experiment in case of mobile text entry methods. In user evaluation procedure, we considered session definition as a user performing text entry with a benchmark text on a single text input system following a particular typing method. Hence, total 40 sessions (four systems × two methods × five benchmark texts) were performed by each user in Hindi, 24 (three systems × two methods × four benchmark texts) for Bengali and 24 (three systems × two methods × four benchmark texts). The testing corpus (Table 4.8) and system for typing were chosen randomly in each session to minimize the memorizing effect of users. Each session lasted, on an average, varied from approximately 27 to 70 minutes (lower values for typing with word prediction systems and higher values with systems having no prediction support). On an average, a user performed two to three sessions per day. Most of the users completed all the sessions assigned to them. A user approximately took two to six months to finish all the sessions. Throughout the evaluation phase, the system and text ordering were counterbalanced over the users. The time taken to compose texts and the keys which were pressed by users are recorded into a log file. After completion of each session, feedback was collected from each user.

4.2.5.2 Simulated evaluation

In this method, a simulation program has been developed which reads sentence from texts and passes them character by character to the proposed word prediction system. It processes and then populates words in the prediction list. Whenever the prediction list contains the target word, it accepts the word and processed for next word until the text is complete. If the word is presented in the prediction list and selected by the simulated component, it automatically increments the hit count and appends space in the composed text. It may be noted that this evaluation method is applied on desktop based text entry interfaces.

4.2.5.3 Experimental results

The performance of *iLiPi* system with and without prediction, *mLiPi* system with prediction, *Lipik* and *IME*, *Panini* are evaluated and presented below.
User evaluation: We analyze the effectiveness of *iLiPi* by comparing average text entry rate and error reduction rate (calculated as percentage of erroneous character left out of the total composed text, averaged over all users) With Prediction (*WP*) and WithOut Prediction (*WoP*) support for different benchmark texts. We observe mean text entry rates of 10.52, 9.82 and 9.38 WPM on different benchmark text in Hindi, Bengali and Telugu, respectively. On the other hand, *mLiPi* text entry interface achieves 10.03, 10.01

and 9.91 WPM text entry rates for Hindi, Bengali and Telugu languages with respect to *Panini* interface as 7.91, 7.85 and 7.14 WPM text entry rates in Hindi, Bengali and Telugu. We also observe significant difference in mean text entry rate between *WoP* and *WP* using *Paired t* test for both language. For Hindi, $t = -8.290$, mean difference $\Delta\mu$ (WoP-WP) is -5.222, $p < 0.05$ whereas for Bengali, $t = -6.766$ and mean difference is -5.12 and for Telugu, it is $t = -6.153$ and mean difference is -5.04. User experiment result reveals that keyboard augmented with word prediction support performs 98.57%, 97.52% and 99.74% better from keyboard without word prediction support for Hindi, Bengali and Telugu, respectively. We also observe that performance of In-domain text (that is, TH_5, TB_4 TT_4 and for Hindi, Bengali and Telugu) is better compare to Out-of-domain text ($TH_1 - TH_4$, $TB_1 - TB_3$ and $TT_1 - TB_3$) when word prediction is used. Analysis of variance for benchmark texts for *WoP* reveals that there is no significant difference between performance of different benchmark texts. In other words, when word prediction is not used, the content of the benchmark texts may not be different from one another within a language. Hence we can conclude that the performance of virtual keyboard augmented with word prediction is better compared to without prediction. We also observe that, text composition using proposed approach contains less error in three languages when word prediction is used. Analysis of paired t test reveals that, there is a significant difference in the mean of typing errors without using prediction and using proposed word prediction $t = 18.423$, $\Delta\mu(WoP - WP) = 12.32$, $t = 19.698$, $\Delta\mu(WoP - WP) = 10.47$ and $t = 19.944$, $\Delta\mu(WoP - WP) = 10.18$, and $p < 0.05$ using 2-tail test for Hindi, Bengali and Telugu, respectively. The results of user evaluation show that the proposed *iLiPi* system corrects on the average 86.58%, 82.62% and 79.78% typing errors for Hindi, Bengali and Telugu compare to the text entry without prediction on the same benchmark texts. On the other hand, according to user evaluation result, *mLiPi* system corrects on an average 84.27%, 82.73% and 79.97% typing errors with respect to 76.11%, 72.15% and 71.18%, respectively for *Panini* mobile text entry system.

Performance of the proposed system was further analyzed by comparing the efficiency of our *iLiPi* system with the existing approaches like *Lipik* [124] and *IME* [68] summarized in Table. 4.9. These two systems resemble with our system so far as the input and output are concerned (detail described in Chapter 2). From Table. 4.9, we observe that *iLiPi*, *Lipik* and *IME* achieve average text entry rates of 9.82, 6.21 and 7.22 WPM, keystroke savings of 42.90%, 30.07% and 26.38% and prediction utilization of 94.65%, 86.28% and 85.22% for Bengali, respectively.

User experiment results of *Panini* and *mLiPi* text entry interfaces are shown in Table. 4.10.

Table 4.9: Quantitative comparison with existing word prediction systems

-	Language	Lipik	IME	iLiPi
WPM	Hindi	06.09	07.43	10.52
	Bengali	06.21	07.22	09.82
	Telugu	06.08	06.99	09.38
Error (%)	Hindi	08.72	09.08	01.91
	Bengali	08.46	09.96	02.20
	Telugu	08.68	10.09	02.42
Keystroke saving (%)	Hindi	30.10	26.47	43.49
	Bengali	30.07	26.38	42.90
	Telugu	30.06	26.12	42.64
Potential keystroke saving (%)	Hindi	34.29	31.24	45.17
	Bengali	34.85	30.95	45.32
	Telugu	35.11	31.13	45.91
Prediction utilization (%)	Hindi	87.79	84.73	96.29
	Bengali	86.28	85.22	94.65
	Telugu	86.41	85.33	94.54

Table 4.10: Quantitative results of *mLiPi* and *Panini* interface

-	Language	Panini	mLiPi
WPM	Hindi	07.91	10.03
	Bengali	07.85	10.01
	Telugu	07.14	9.91
Error (%)	Hindi	23.89	15.73
	Bengali	27.85	17.27
	Telugu	28.82	20.03

We conducted a small survey with the participants after completion of their experimental sessions. Questions relevant to this article, which were asked to participants, are summarized in Table 4.11. A set of six questionnaires was given to each participant to rate the system in a $1-5$ *Likert-scale*. Response from the participants were then consolidated, which was presented in Table 4.11. In this table, a comparison was made between different available word prediction systems with *iLiPi* in Hindi, Bengali and Telugu from users satisfaction point of view. In addition to this, we also compared the performance of our proposed system with and without prediction. When asked about the ease of use, users expressed that *WP* (with prediction) was much easier to use than the *WoP* (without prediction). Similarly, users felt system with prediction support was more comfortable, less tiring, faster, more accurate and useful over without prediction system, *Lipik* and *IME*.

Table 4.11: Average scores for survey responses in the range 1-5

Question	Lipik		IME		iLiPi			
					WP		WoP	
	H	B	H	B	H	B	H	B
How easy did you find the word prediction system ? (5- much easier, 1- much harder)	3.12	3.07	3.32	3.37	4.41	4.38	2.86	2.88
How tiring did you find using the text entry method ? (5- least tiring, 1- most tiring)	2.57	2.66	3.06	3.09	4.03	3.98	2.27	2.29
How faster did you compose the text ? (5- very fast, 1- very slow)	3.65	3.69	3.14	3.11	4.48	4.43	2.72	2.75
How accurate was the composed text ? (5- accurate, 1- inaccurate)	3.26	3.22	2.58	2.61	4.17	4.19	2.04	2.07
How useful was the word prediction system ? (5- very useful, 1- less useful)	3.75	3.79	3.88	3.82	4.23	4.26	-	-
How distracting did you find the word prediction ? (5- very distracting, 1- less distracting)	2.14	2.17	2.12	2.16	1.69	1.73	-	-

simulated evaluation: We further analyze the text entry performance of different prediction techniques by taking the help of simulation. We consider existing techniques based on unigram, bigram, trigram, last recently used (LRU) and our proposed approach. In our proposed approach, performance of unigram, bigram and trigram based prediction methods are also evaluated. The input text of the simulation, for all the cases, is user typed benchmark texts through *iLiPi* system without prediction support in Bengali language. The performance of Potential keystroke savings (PKS), Hit rate (HR) and the percentage of correctly composed text are summarized in Table 4.12. We observe that performance of trigram is better compared to unigram, bigram and LRU based methods. However, proposed prediction performs better than the existing approaches in terms of PKS, HR and percentage of correctly typed text.

Table 4.12: Performance analysis of different prediction approaches

Approach		Potential keystroke savings	Hit rate	Correctly composed text (%)
Traditional	Unigram	29.89	80.03	70.43
	Bigram	35.12	83.18	73.89
	Trigram	37.08	85.12	80.11
	LRU	10.14	79.11	61.05
Proposed approach	Unigram	35.86	90.23	90.81
	Bigram	42.15	92.94	92.74
	Trigram	44.53	94.11	94.37

4.3 Summary

In this chapter, we describe an efficient Indic language based word-level prediction method which suggests probable words based on the previous context and currently typed text. It can be augmented with virtual keyboard developed in any language, specifically Indic languages which are having typical linguistic features and complexities. Our experiments with Hindi, Bengali and Telugu substantiate the efficacy of the proposed approach. The proposed Hindi, Bengali and Telugu text entry systems are comparable to its counterparts namely Lipik, Google and *SulekhA*. The word-level prediction mechanism based on multi-variate ranking proposed in this prediction method is an indigenous approach. The proposed system takes care of typographical error, confusions due to phonetic or graphical similarity between characters, trigram probability and recency of word etc. which make the text composition fast and accurate. More significantly, the prediction system works even if the initial entry is mis-spelled unlike the existing approaches. Experimental results and empirical studies reveal that our prediction system is able to detect and correct typing errors efficiently.

Chapter 5

Gaze-Based Text Entry System

Of late, eye gaze has become an important modality of text entry in large and small display digital devices covering people with disabilities beside the able-bodied. Despite of many tools being developed, issues like reducing dwell time and visual search time, enhancing accuracy in composed text and eye-controlled mouse movement stability etc. are still points of concern in making any gaze typing interface more user friendly, accurate and robust. Moreover, eye typing interface having a large number of keys suffers with many problems like selecting wrong characters due to *Midas Touch* problem, more character searching time etc. Some linguistic issues often decline in mitigating dwell time incurred for character by character based eye typing. We propose an efficient gaze based text entry mechanisms for English and Indic languages. We design a single-tap keyboard layout and arrange the keys in such a way that it achieves optimum eye movement during eye typing. Then, we develop two efficient mechanisms to diminish the eye fixation time while selecting a character in eye typing task. We also develop a hybrid visual feedback which helps user to find desired character quickly in the keyboard. The proposed dwell time and visual search time diminishing strategies increase the eye typing rate with less error committed during text entry. On the other hand, most of the user difficulties faced during eye typing through Indic language keyboards have been addressed in our proposed interfaces.

The rest of the chapter is organized as follows. Section 5.1 describes on-screen English keyboard design strategy with optimum eye movement feature. The proposed method for mitigating visual search time to find a key is described in Section 5.2. Section 5.3 states two dwell-time diminishing techniques in eye typing. Experimental setup, results and analysis of proposed and existing English eye typing interfaces are reported in Section 5.4. Section 5.5 summarizes the chapter.

5.1 Developing an on-screen keyboard layout with optimum eye movement

We have observed the resemblance between mouse- and eye-based text entry tasks and explored several design principles of mouse-based keyboard to be applied at eye typing system. After selecting effective design from user study, we have further modified it to support less saccadic eye movement. Also, we have analyzed the keyboard design parameters which affect the eye movement and visual search tasks and set the optimum values got from user experiments.

5.1.1 Analysis of mouse- and gaze-based text entry tasks

We conduct an initial pilot study with sufficiently a large number of users on text entry using mouse- and gaze-based interfaces. Times required for performing each and every task of participants have been recorded in a log file. Exploring the log file, we analyzed user followed steps for composing a single character which are described below.
For a text composition task, after knowing what next character needs to be typed (either by reading text to be typed or remembering texts through listen before typing experiment), users' followed steps which are broadly categorized into three phases.

Perception: While perceiving, a user sees the region of interest on screen. In case of text composition through on-screen keyboards in mouse- and gaze-based interface, users converge their search from full keyboard to the intended character. We take the perception time as T_p which includes saccadic eye movement for both the text entry interface.

Cognition: After seeing through eyes, user processes the received image in brain and matches it with pre-stored character image. If it is matched with the intended one, then brain sends signal to appropriate organ responsible to perform the key selection task. Otherwise, user continues to perceive. Let the total time taken to finish cognition task for a single iteration is T_c for both the text entry modes.

Motor movement: After fixing the location of the characters on-screen, brain transmits signals to corresponding organ to get activated. For mouse-based interaction, brain gives signals to hand for moving the mouse pointer to the desired location. Performing the eye typing task, when eye points on the required key button, brain gives signal to eye to fix for a while. The time required for performing this task is T_m. Actually, for mouse-based text entry, $T_m << T_c << T_p$. In case of eye typing, T_m becomes eye fixation time. So, as a whole, the time required to enter a text of n characters denoted as T_{tm}) is

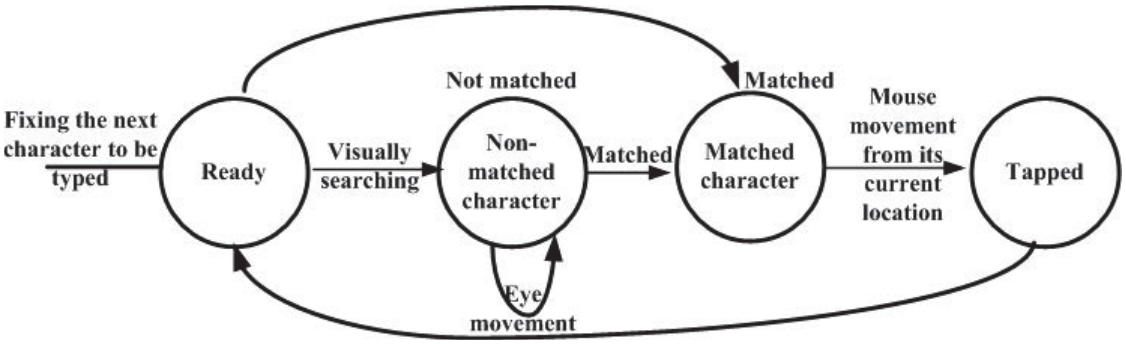
$$T_{tm} = n(T_p + T_c + T_m) \tag{5.1}$$

In other way of representation, visually searching the intended characters becomes an essential task in both mouse and eye gaze-based text entry through on-screen keyboard. The searching task is a combination of both perception and cognition task described above. The state diagram of the whole process is depicted in Fig. 5.1(a) and described as follows.

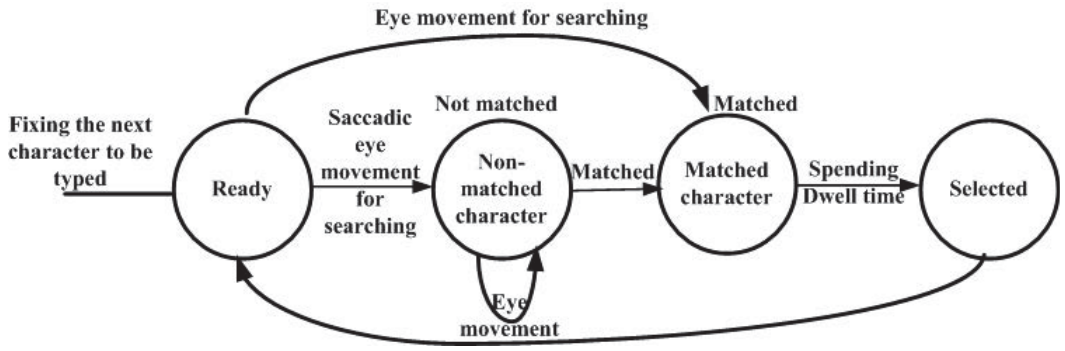
Suppose the total time spent for accomplishing the task is T_m and time for fixing the next character to be typed in user's mind is δt_1. Time for searching the desired character is t_2 which consists of two subtasks, total time spent for reaching to unsuccessful characters (t_{21}) and average time to find out a button is (t_{22}).

If a keyboard is having N characters visible from where user needs to find out a character in an iteration, then $0 \leq t_{21} \leq (N-1)t_{22}$ and $t_2 = t_{21} + t_{22}$

Time is zero when first hovered character is the desired one and maximum when upto $(N-1)$th characters user searches to find the intended character.



(a) State diagram of mouse-based text entry task



(b) State diagram of gaze-based text entry task

Figure 5.1: State diagrams of mouse- and gaze-based text entry tasks

After the character is found, user locates the mouse where he left (δt_3) and then move the mouse to the desired character (t_4) and tapped it (δt_5).

So, total time spent to select a character is $T_m = \delta t_1 + t_2 + \delta t_3 + t_4 + \delta t_5$

The δt_1, δt_3 and δt_5 times are moderately small for users who got habituated with the interface.

On the other hand, the state diagram of performing eye typing task is depicted in Fig. 5.1(b). Suppose, the total time spent for accomplishing the task is T_e. Also, time for fixing the next character to be typed in user's mind is δt_1 (same as with the mouse-based interaction). The time for searching the desired character through saccadic eye movement is t_2 which consists of two subtasks, total time spent for unsuccessful character finding (t_{21}) and time to find out required button is t_{22}.

Similar to mouse-based interaction, time is zero when first hovered character is the desired one and maximum when upto $(N-1)$th characters user searches to find the desired one. If t_2 and t_3 are the times for finding the target key and eye fixation, then total time spent to select a character is $T = \delta t_1 + t_2 + t_3$.

After analyzing, we conclude that the gaze-based interaction spends moderate amount of time in fixation (depends on the duration of dwell time) which may be equivalent to character selection time for mouse-based interface (locating the mouse pointer (δt_3) + placing the mouse pointer on intended character (t_4) + tapping the character (δt_5)). On the other hand, the eye movement time in gaze-based interface, in view of experimental results is found to be more or less similar with mouse-based interface.

5.1.2 Virtual keyboard design exploration for *eye typing*

We implement popular virtual keyboard design methodologies [128] for eye typing and compare them with respect to user's eye typing rate, eye fatigue, stressfulness etc. For that, we construct a pool of keyboard designs where mouse movement optimized virtual keyboard layouts are also considered beside the representative designs used in eye typing interfaces. We conduct a pilot study to select the suitable keyboard layout with respect to eye typing speed and accuracy.

Apparatus: Standard desktop computing environment with moderate sized screen and processing speed was used to conduct experiment. Sony *PlayStation Eye* webcam, a self-designed IR Lamp (Fig. 5.6) and a *Gaze-Tracker* software were used for experiments.

A detailed description about our experiment are given in the following.

Keyboard layouts: We considered five representative virtual keyboard layouts [236] which were having moderately higher text entry speed in single pointer based typing. These are namely, *QWERTY Dvorak*, *Opti*, *Fitaly* and *Lewis*. Also, four eye typing interface based keyboard layouts were taken for the experiments namely optimized *scrollable* keyboard which saves the screen space proposed by Špakov et al. [186], keyboard designed by Majaranta et al. [132] maintaining adjustable dwell time, a design called *Iwrite*, which is a square shaped interface keeping characters at outer side and text area in middle for gaze typing [208]) and a keyboard attached with the interface, proposed by Morimoto and Amir following *Context Switching* (CS) as a new activation mechanism for gaze controlled interfaces [147]. During the experiments, we kept all keyboard dimensions same like both key width and height as 70 pixels, horizontal distance between two keys is eight pixels and magnification is 30% of original size.

Dependent measures: As with the pilot study, the dependent measures used in this experiment were words per minute and the total error rate (addition of corrected and uncorrected error rate). Data collected during the experiment was analyzed with *StreamAnalyzer* utility [226].

Participants: We perform expert user-based evaluation [48] with every candidate keyboard layout in our eye tracking setup. The evaluations have been conducted by five expert participants (UG/PG students) having previous experience in participating user evaluation. The details are furnished in Section 5.4.3.

Procedure: We discuss with participants about the experimental setup and procedure to be followed. Next, the procedure was started. First, the calibration with eye tracker was done by each participant using nine calibration points. Participants were then given 20 minutes of free practice time for typing comfort with eye tracker. Participants then practiced typing sentences with a keyboard interface, randomly chosen from the design pool, for ten minutes. After that, participants took another five minute break, recalibrated their eyes and began the typing session. The session consists of typing one practice sentence followed by five randomly selected test sentences. For each experimental session, the details about the task performing along with the calculated result are stored in a log file.

Observation: Each participant performed experiment with nine designs. With five participants, the entire study comprised of 45 trials. The analysis of variance (ANOVA)

on text entry speeds shows that there is a significant difference between the means of keyboard's text entry performance on users ($F(8, 36) = 6.32$, $p < 0.05$). Further, The Post-hoc using *Tukey HSD* test reveals significant difference between performance of *Fitaly* and other keyboard designs ($p < 0.05$). The results also reveal that the *Fitaly* layout, with the participants, achieves on an average 21.26% improvement in eye typing rate than next best design.

After each session, we collected user feedback by formally asking some questions regarding eye fatigue, visual searching, eye typing error committed etc in $1 - 7$ Likert-scale. The result reveals that users like the *Fitaly* than other keyboard designs for ease of use through gaze ($z = 56.00, p < .001$) and less stressfulness ($z = -48.00, p < .001$). Also, irrespective of user's capability, the searching was less to find the intended key and number of typing errors committed was also decreased than other gaze-based keyboards. Participants also felt that *Fitaly* combines the default size of the key and space between keys as in such a perfect way that supports proper fixation and less saccadic movement.

5.1.3 Proposal of a virtual keyboard for eye typing

Fitaly, a frequency based keyboard layout contains two spacebars. Here characters are arranged in row-major order according to their unigram frequencies. Also, there is no need of two spacebars because more than one spacebar causes more eye gaze movement. To understand the pattern of saccadic eye movement in character by character text entry, several eye typing experiments have been performed on popular *QWERTY* layout with expert users and different standard sentences, taken from Mackenzie's phrase set [128]. We analyzed eye positions on different characters and paths of moving eyes during experiment. We observed that users saw in a relatively broad region around the keys. This may be caused by inaccuracies in the eye-tracker or by users not gazing directly at the center of the keys. Analysis of the trace data show that 63% of the total time participants activated a key by gazing inside it. For the remaining time, participants' gaze passed within the 1.5 key radii threshold but never went inside the key. Most of the time, user always started moving around the middle portion of the layout. Searching for next character was always started from the centre and spread over even on the character placed at the outer region of the keyboard. This is illustrated as the heat map for typing English sentence "A QUICK BROWN FOX JUMPS OVER THE LAZY DOG" as shown in Fig. 5.2.

Taking those design decisions into account, We propose to modify *Fitaly* layout for ryr typing interface. Here, we can reduce the search time, if frequently occurring characters are placed on or around the central region of the keyboard. If it is possible to

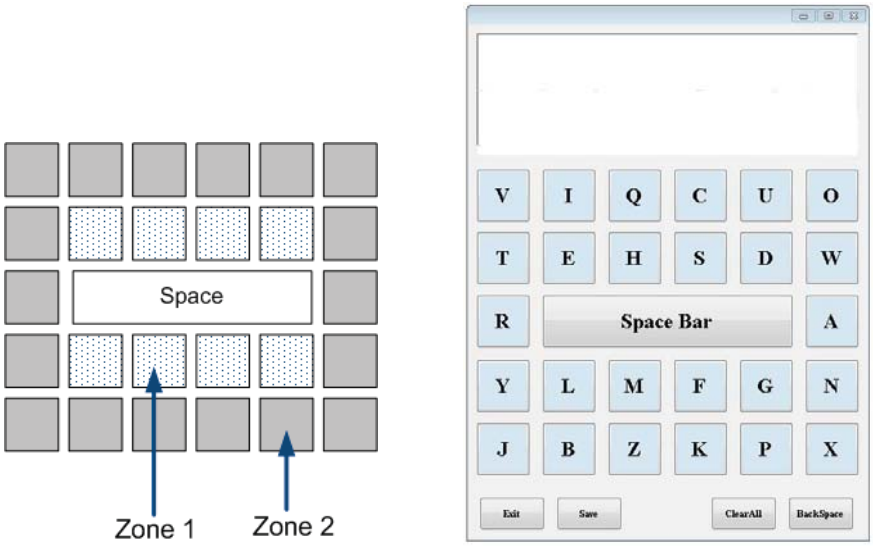
Figure 5.2: Heat map of gaze locations during the experiment

accommodate all frequently occurred characters in central region, then the characters may be placed into two zones, that is, the most frequently occurring characters in the central zone (zone 1) and next most frequently occurring characters in the outer zone surrounding to centre zone (zone 2). We use Mayzner and Tresselt's table [143] which provides unigram as well as bigram frequencies of English characters. Initially, we set the space character at the middle of the central zone (as space is the highest probable character among any English characters) following most frequently occurred characters around it (zone 1) and other less probable characters can be spatially arranged in other zone (zone 2) depending on their frequency of occurrence (Fig. 5.3(a)). Also each character, on hovering, is spoken out for better understandability of users.

After modifying the existing *Fitaly* keyboard and forming proposed layout (we call it *EyeBoard*, keyboard supporting eye gaze behavior), we fix values of three keyboard design parameters which affect eye movement: value or range of size of the key, distance between keys and magnification of the keys (magnification is suitable for eye typing interface and it provides benefit for pointing to and selecting target [180]).

5.2 Reducing visual search time

Majaranta et al. [133] conducted experiments with visual feedback on buttons and audio feedback while accessing the eye typing interface with less dwell time associated. She had drawn certain guidelines on feedback on eye typing. We follow three among them: a) using a short non-speech sound to confirm selection which is a simple 'click', b) combining speech with visual feedback where the selected character is spoken out and next probable character buttons get selected with red color and c) using simple one-level feedback with short dwell times, that is, the color changes to the buttons is done in one-go. We introduce



(a) Multi-zonal layout for virtual keyboard

(b) Proposed interface

Figure 5.3: Schematic and implemented diagram of proposed *EyeBoard* interface

a new way of providing visual feedback on key button which can be best fitted to short dwell time spent while typing. To judge the efficacy of the feedback mode with respect to user performance, we perform an user experiment with two feedback modes.

5.2.1 User study on feedback mode

Eight users (five male and three female, mean age 27.5 years) from the local area participated in the experiment. All were having normal eye vision. None had previous experience with eye tracking or eye typing, but all were familiar with desktop computers and QWERTY keyboard layout. Two feedback modes were tested.

- Two-level visual: The key is highlighted on focus. On selection, the key background turns red [133].
- Visual on next probable character(s): After selecting a character, next probable character buttons (calculated from character level bigram analysis) will be highlighted in red.

The dwell time for selection was 450 ms. For the two-level visual mode, the delay before highlight was 150 ms. Thus, the highlighting phenomena (before selection) lasted for 300 ms. Based on the experience from pilot studies, the dwell time to re-select the

current letter was increased by 120 ms to avoid erroneous double entries (e.g., 'aa'). Thus, the dwell time for two consecutive letters was $450 + 120 = 570$ ms. On the other hand, feedback mode in terms of next character highlighting spent 450 ms for selecting a character key. There, two same character key selection took $450 + 450 = 900$ ms. The experiment was a repeated measures design using a counterbalanced order of presentation. The results are based on the total of 320 phrases (eight participants × two feedback modes × two keyboard designs ×10 phrases).

Previously described apparatus were used for experiments. For both type of modes, we considered QWERTY and *EyeBoard* keyboard layout.

For all experiments, participants need to type short phrases of text. They were instructed to first read and memorize the source phrase and then, after calibration, to eye type it as quickly and accurately as possible. The distance between the monitor and participant's eye remains 70–80 cm. The participants were instructed not to move their head during experiment. It is also said that if movement is happened, calibration got disturbed and then users need to start afresh right from recalibration. All the phrases used in experiments were in English. While typing, participants could correct the phrase by pressing the Backspace button of keyboard layouts. The following is the observation from the user study.

- *Typing speed*: The average typing speed of participants is 7.32 wpm ($SD = 1.23$). A faster entry speed was expected, since the dwell time was less and the participants were experienced. The first feedback mode had a significant effect on text entry speed ($F(1, 14) = 1.28$, $p < 0.05$). The *two*−level visual mode was significantly slower (mean 6.83 wpm, $SD = 1.58$) than the proposed mode, that is, visual on next probable characters (mean 7.59 wpm, $SD = 1.18$).

- *Total error rate*: Eye typing with two-level visual feedback produced overall more errors than visual feedback on next characters. Probably, to do task in a faster way, user often spent very less time on first layer which actually chose wrong characters (mean error rate 1.15%). On the other hand, after getting complete understandability what the proposed feedback was given on probable characters, participants felt comfort in using this without fumbling many (mean error rate 0.41%). Interestingly, one observation we gathered from the experiment that in eye typing, users commit errors (especially with short dwell times) and immediately correct them.

- *Subjective evaluation*: After each session, we collected participants' impression by formally asking some questions to them regarding eye fatigue, visual searching,

coloring effect, cognitive load while typing and comfort in eye typing in $1-7$ Likert-scale. The result reveals that users like the next character highlighting mode than *two*− layer visual mode for ease of use through gaze ($z = 47.00$; $p < .001$) and less stressful ($z = 39.00$; $p < .001$). Also, irrespective of user's capability, the searching was less to find the intended key (because the proposed mode helped them) and cognitive load incurred by users were less for the newer mode ($z = 42.00$; $p < .001$). As each key selection in *two*−layer visual mode needs more concentration, the eye typing task became more stressful than the other ($z = 35.00$; $p < .001$). Also, user felt that the coloring effect after selection somehow delays the actual key selection task completion and as a result, it causes less comfort and eye typing speed. Participants expressed that initially the two-layer mode was fruitful to them as it marked each selection. After sometime, when they were feeling comfortable, they wanted faster entry speed to finish the typing as quickly as possible which couldn't be supported by the 2-layer visual mode. In other words, next probable character highlighter is found more preferable to user if the next target characters are readily available in the interface.

5.2.2 Proposed hybrid method

After next character button coloring was evolved as a preferred visual feedback mode, the immediate problem that arise is how to highlight the most probable characters so that it would reduce next character searching time significantly. For this, we propose two ways of displaying characters which can give advantage in text typing namely a) visual on next probable characters related to the last entered character based on bi- and trigram analysis and b) the character(s) selected from the trigram vocabulary entries holding previous two words in context and character level $k-$gram context for current character chunk under composition. Both the methodologies support characters or words present in the corpus, but not for OOV (out-of-vocabulary) entries. Although the training corpus is huge with full of character or word variation, some word formation (specially nouns) still unable to provide any visual feedback. We aim to address this problem by combining the previous two methods, which are activated in different times. Till the text composition matched with previous word level and current $k-$gram character level context, the second method remains active and provides next characters suitable after present typed character chunk. The time when no next character is found out from dictionary and user needs further typing, first method is initiated. In this way, visual feedback remains alive even for non-vocabulary words.

To judge the efficacy of aforementioned three methods and choose the most effective

method during eye typing, we conducted a small pilot study with users sampled from expert computer professional community.

Five users (four male and one female, mean age 25.3 years) from the university campus participated in the experiment. All were having normal eye vision. None had previous experience with eye tracking or eye typing, but all were proficient with computers and text messaging in mobile. Three aforementioned methods on QWERTY and *EyeBoard* keyboard layout were tested. Dwell time is taken as 450 ms. The experiment similar to the user study stated in Section 5.2.1 was a repeated measures design using a counterbalanced order of presentation. A total of 150 phrases (five participants $\times$ three feedback modes $\times$ two keyboard designs $\times$ five phrases) was typed by each user.

We followed similar procedures with same setup as in Section 5.2.1. We observe the following outcome.

A one-way ANOVA indicates significant differences between the three modes implemented on keyboards in average words per minute ($F(2,7) = 1.08$, $p < 0.05$). Further, The Post-hoc using Tukey HSD test reveals significant difference between performance of third method and others ($p < 0.05$). The third method achieves significantly faster typing rate (averaged on two keyboards), which is 7.14 wpm than others (5.83 wpm and 6.42 wpm). Also, no significant effect was observed of individual modes on keyboard designs ($\mathrm{F}(1,13) = 0.772$, n.s.).

We gathered user feedback after every experimental session through asking some questions to them regarding eye fatigue, visual searching, cognitive load while typing etc in $1-7$ *Likert*-scale. The result reveals that users like the hybrid third mode than two other modes for achieving less fatigue ($z = 46.00$; $p < .001$). Also, irrespective of user's capability, less searching and cognitive load were incurred for the hybrid third mode ($z = 41.00$; $p < .001$). Praticipants expressed their feeling that initially they were happy with simple bigram character highlighting mode, but they got habituated with the keyboard interface, they wanted to finish the typing task more quickly which could not be supported by first or second method. Alternatively, this hybrid method was timely being adapted with the situation. While aligning with prediction, it considered more deep context with $k-$gram character level suggestion to get filtered output while on the other hand, unavailability of next character series, it followed the common bigram level next character selection which many times helped user in constructing unknown word quite easily. Participants also admitted that initially, this third method was tough for learning and once learned, it was smooth for eye typing. On the other hand, participants gave the feedback that, with six characters highlighted as visual stimuli, the search space and probability of hitting the user intended character trade-off got reduced. The results

support two hypothesis: a) participants using the combined method perform the eye-typing task faster than participants using the other two individual methods and b) the hybrid method based visual feedback incurred less cognitive load than other two methods. So, visual feedback for the proposed interface was based on two principles, i) following the next character button coloring mode and ii) a suitable hybrid method supporting both dictionary and partially non-dictionary word creation depending on the context. As an example, after composing character chunk 'SC', system chooses next probable characters as 'E', 'A', 'O', 'H' and 'K' and colors them with red (Fig. 5.4).

5.3 Mechanism supporting dwell-free eye typing

Dwell-time is an important concern in any eye typing interaction. Several efforts have been made to diminish the dwell time to increase the eye typing rate. In general, according to the existing approaches, users need to follow a specific eye movement path to select a button (character, suggestion button etc.). In this work, we propose a dwell-free interaction. According to our proposed approach, users need not to follow any eye gestures [228] as well as they will not perform continuous eye writing in a single direction like *pEYEwrite* [208], during the interaction. Taking the humans' eye gaze behavior into consideration, *Continuous writing* matches best to the requirements of interfaces that utilize eye gaze for input control. Human eye gaze is always moving and always 'on'

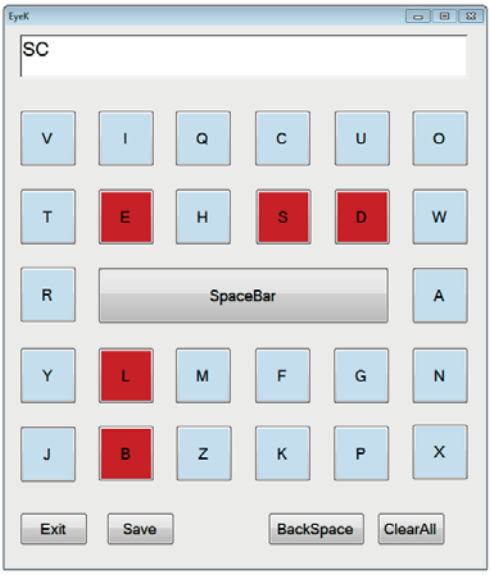
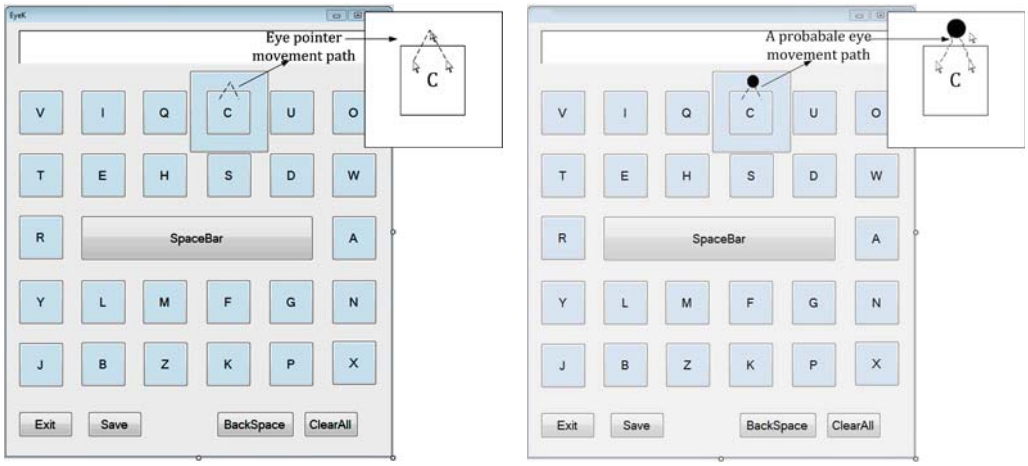


Figure 5.4: Next character highlighting in proposed interface

that can not be simply switched off [11]. Thus, instead of *Continuous eye writing*, it could be better to support eye writing for particular region to perform specific task. Our interaction is framed based on 'inside' and 'outside' continuous movement of eye through the key button area of the interface. The eye movement what user needs to perform is very natural and intuitive, so the memorability enhances and cognitive load in performing the interaction becomes less than *eye gesturing* and *continuous eye writing*. Although the interaction is found dissimilar than normal fixed gesture, it uses crossing in spite of pointing. Crossing refers to crossing a cursor over a line. According to *Fitts' law*, crossing is easier than pointing for large, proximate targets [225]. Selection of letters on the on-screen keyboard follows Fitts' law for pointing, meaning the larger the button, the easier it will be to select. But larger button size is at the cost of screen area.

In the eye typing keyboard, every key area is divided into two parts, inside actual key area and outside overlay area. A short "self-study" was conducted by the designers to determine subjectively optimal interface parameters, including height and width of key and length of outside area. Both height and width of the actual keys are taken empirically as 1.6 cm. We check the length of overlay area starting from 0.4 cm with an increment of 0.2 cm and found 0.8 cm as the comfortable length of overlay area where all of our participants performed the character selection task very smoothly. So, we keep the length of overlay area from the boundary of actual key area as 0.8 cm in our design. The interaction actually activates while user moves the eye pointer through the areas in inside-outside-inside fashion, shown in Fig. 5.5(a). With this oriented movement, keys will be selected automatically. As eye movement is faster than mouse or finger movement, this interaction takes minimum effort and time with respect to the dwell time [169]. A pictorial example clarifies the scenario further. Suppose, user wants to select character 'C'. While hovering on the character, key and outside area are visible (if it has been highlighted previously, then after hovering, highlighting will be off). At any instance, user starts moving the eye pointer from the key area, goes to outside area and again comes back to key area (follows almost a trajectory) to complete the interaction phase (Fig. 5.5(a), enlarged portion). After coming back on the key, character is entered. If users need to enter same characters twice, they have to get out of the character initially and enter it again in similar manner. This policy sometimes goes against user's natural eye movement naturalness and costs moderate user concentration to perform the interaction. But, after practicing, users find it less hectic and take moderate less time to complete it. As the interaction completes after finishing both 'going outside' and 'coming back' eye movements, it is up to user who can anytime discontinue to follow the path in order to cancel the selection process as incomplete to cancel the whole selection process.



(a) Eye movement on a key - Method 1 **(b)** Eye movement on a key - Method 2

Figure 5.5: Dwell time free eye typing through two methods in the *EyeBoard* interface

A further variation of the aforementioned interaction is proposed by Chakraborty et al. [32]. The new version supports more controlled eye movement toward selecting a key by tracing eye movement pattern. In the outer key area (activated after hovering), a black circular disc appears at upper side of the key. After hovering on the intended key, user requires to "go out" from the inner key, reach to that prominent point and after looking, "come back" inside the inner key area (Fig. 5.5(b)). Feedback system, same as the previous method, is applied providing selection confirmation to the user. Suppose, at an instance, user wants to select character 'C'. Then, on hovering 'C', the outer layer becomes visible and user "goes out" from the upper side, sees the point and "comes back" into the inner key from the same side. For double selection of a single key, same procedure as the previous method is to be followed.

5.4 Experiments and experimental results

In order to prove the effectiveness of the proposed interface, we have conducted a number of experiments. The judging is done through two perspectives, a) comparing with other study designs considering/not considering certain facilities and b) comparing with existing eye typing methods with respect to both quantitative (like *Text Entry Rate*, *Error Rate* etc.) and qualitative evaluation (subjective evaluations involving of measurement of cognitive load, usability etc.) measures. In this section, we discuss our longitudinal experiments and the results are obtained.

5.4.1 Apparatus

All experiments were conducted using two Desktop computers each having 2.2GHz Intel Core2Duo processor with 15″ wide screen LCD color monitor having 1440×900 resolution. We created a hand-made setup to perform gaze-based typing experiments. Two essential devices were required to complete the setup, a camera without infrared filter to catch the gaze position online and an infrared light source to properly distinguish the black gaze from the black surrounding portions of the eye. We first modified *Sony PlayStation Eye* webcam where original lens was replaced by manual focus and Infrared (IR) filter removed lens (Fig. 5.6). Next, we developed a circuit used as *IR Lamp*, consisting a matrix of 10 IR LED and a circuit of restricting the current source maximum upto five volt (the environment can posses varying power source) (Fig. 5.6). Also, an open source *ITU GazeTracker* software [1], developed by IT University of Copenhagen, were used for experiments. To achieve good eye tracking accuracy with the software, the camera was placed on the center path from eye to screen and infrared circuits were placed beneath of the screen (Fig. 5.6).

For better detection of eye gaze, we built a setup where camera was placed attached to a stand by hanging wires in front of eye at the center area of the screen. The distance between user's eye from camera and screen were approximately eight cm and 60 cm, respectively. The developed and other interfaces were developed using *C#* Visual Studio 2010. The key press events were recorded automatically and stored in a log file using a separate event hooking program. Another window hook program was developed to track gaze positions, also written in *C#*. All experiments were performed in Windows 7 environment. Controlled light conditions and positioning of the setup were maintained.

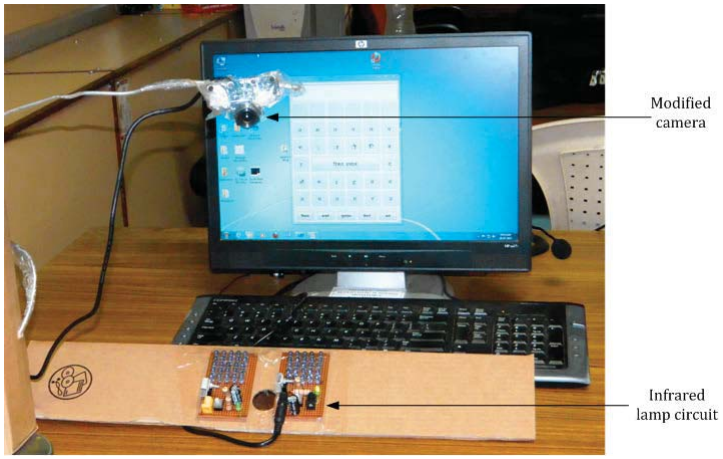


Figure 5.6: Experimental setup with camera and infrared lamps

5.4.2 Designs under evaluation

Two types of designs were considered in this longitudinal study, one for judging the effectiveness of the incremental intermediate designs, whether exist or not and other to compare the efficacy of the complete proposed interface with respect to other existing designs. For the first one, three study designs were considered given below.

Design 1 *(Multi–zonal character key arrangement for less eye movement)*: In this design, we initially arranged the characters into two different concentric zones surrounding center based on their unigram and bigram frequency associations. Moreover, we arranged the zonal characters based on trial and error method to achieve less eye movement while searching characters. The layout of Design 1 is shown in Fig. 5.3.

Design 2 *(Visual feedback augmented with Design* 1*)*: This design added visual feedback in terms of highlighting next probable character(s) in red, after selecting a character (Fig. 5.4).

Design 3 *(EyeBoard++ Dwell time diminishing strategy augmented with Design* 2*)*: This design, over Design 2, augmented a method that diminished the dwell time concept in finding keys (Fig. 5.5).

5.4.2.1 *EyeBoard++* keyboard designs for Indic languages

We also develop eye typing interfaces suitable for composing Indic language texts. For this purpose, we extend the *EyeBoard* layout design principle into Indic language designs [160]. The keyboards maintain proper balance between three parameters namely size of the keys, space between keys and number of keys present in the layout. To accommodate almost 55 characters into the Indic language layouts, those are distributed into two parts namely *high* and *low* frequent character sets. The first one contains the set of characters which are more frequent whereas second part is having less frequent characters. At a time, only a single part can be displayed. In both the layout, position of Spacebar remains unchanged as it is most frequently occurred and thus placed at the middle portion where user's eye always move around [160]. Characters, for both the layout part, are placed in the zones in a row-major order (means highest frequent character is placed in the leftmost top corner, the next frequent character is just right hand side of that and so on). Further, depending on the bigram frequencies between each character pair, we rearrange the characters in each row of a layer following bigram frequency. Unlike English keyboard [160], dependent vowels (*Matra*) are present in Indic languages where some of them are very frequent in occurrence. So, we decide to place four such keys in the initial layout. The proposed layout in Hindi language is depicted as

Figure 5.7: The *EyeBoard++* keyboard in Hindi

Fig. 5.7. Further, to support visual search time reduction, we implement highlighting next probable character utility in the layout. We conduct experiments with proposed keyboard designs of three Indic languages namely Hindi, Bengali and Telugu, respectively.

Existing designs: Five existing designs namely optimized *scrollable* keyboard (which saves the screen space proposed by Špakov et al. [186]), keyboard designed by Majaranta et al. [132] (maintaining adjustable dwell time), a design called *Iwrite*, which is a square shaped interface keeping characters at outer side and text area in middle for gaze typing [208] (Design 3), Morimoto and Amir's proposed design implementing *Context Switching* (CS) (a new activation mechanism for gaze controlled interfaces [147]), and *EyeBoard* (Panwar et al.'s key size and space optimized layout [160] with adjustable dwell time). All exiting design methodologies were mimicked in our own environment to avoid the biasness due to their experimental setup.

5.4.3 Participants

Ten participants (seven male, three female) were recruited (five of them participated in previous pilot studies and five were new) from the local university campus. Participants ranged from 20 to 31 years (mean = 22.5) (very few common as previous experiments, most of them were newly recruited). Participants possessed a college degree or were currently enrolled in college. All are regular computer users, access on an average four hours per day, but no prior experience with eye tracking. All participants except one have normal vision and rated themselves expertise in composing text through digital

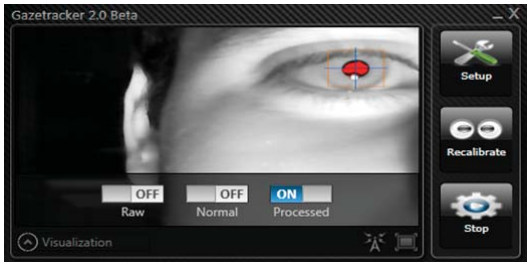
devices. They all cited QWERTY keyboards as their typical text input device. Eight participants are right-eye dominant and two are left-eye dominant, as determined using an eye dominance test [37].

We also conduct user experiments with proposed *EyeBoard++* keyboard designs in Hindi, Bengali and Telugu languages. Six participants (four male, two female) (two each for Hindi, Bengali and Telugu) were contacted from the local area. Participants' age are ranged from 25 to 35 years (mean = 28). All are daily computer users, access their Desktop or Laptops on an average five hours per day, but no prior experience with eye tracking. All participants have normal vision and expertise in composing text through digital devices. five participants are right-eye dominant and one is left-eye dominant, as determined using an eye dominance test [37].

5.4.4 Procedure

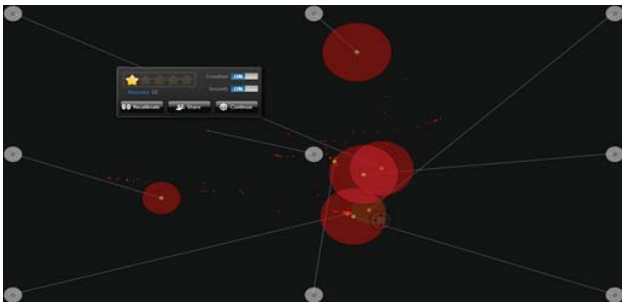
To perform user-based evaluation, users first need to synchronize their eye movement with the gaze tracker. The synchronization process, calibration, with detail description is provided next. The typing sessions started after calibration, when mouse pointer smoothly moved with eye gaze. Participants were asked to remember the testing phrases because experiments, once started, could not be stopped and if happened, recalibration and fresh start was required. As a solution to this situation, the target text could be presented on top of the interface. But, this type of arrangement increased the overall eye movement and as a result, text composing rate got slowed down [185]. The participants were further instructed to transcribe the phrases as fast as possible while making as few errors as possible. Correcting errors was possible by erasing text using backspace key and retyping it.

Calibration: Before an eye tracking system can work with eye gaze, it must be calibrated for the specific user. This is usually done by showing a few (usually nine equally spaced, one at a time) points on the screen and asking the user to gaze at the points. The images of eye are stored separately for each point, and analyzed with corresponding screen coordinates of the same point. These main points are used to calculate any other point on screen via interpolation of the data. The position of eye or the head plays an important role in calibration, thus it should be fixed throughout calibration as well as experiment. Subsequently, to achieve better controlling mouse pointer by eye, calibration is required every time. Accuracy of the tracker is directly proportional to successful calibration. It helps to determine eye movement properly, even in noisy environment, so that it can be used in practical domain. Different stages of calibration process are shown in Fig. 5.8. Figure 5.8(a) depicts the pupil detection process of *ITU GazeTracker*. From



(a) Pupil detection by ITU Gazetraker

(b) Successful calibration results and the points on the screen



(c) Unsuccessful calibration results and the points on the screen

Figure 5.8: Different stages of calibrating eye

Fig. 5.8(b) and (c), we can see that how calibration is done and how accuracy varies in case of successful and unsuccessful calibration. If the accuracy is less than 5 stars, user experiences uneven movement of mouse and it becomes very hard to control. The reason of unsuccessful calibration is mostly due to involuntary head or eye movement and improper lighting conditions. For obtaining impressive result, we set the successful calibration threshold as five stars and less than 1° angle for *Monocular* eye detection.

For participants, first few sessions were spent as training sessions, where they were briefed about the nature of the experiment and completed a short demographic questionnaire. They were introduced to the eye tracking hardware (camera and Infrared lamp positions) and the proposed interface along with other designs (Fig. 5.9). The total time for this interaction was about 10 minutes.

Experiment: As discussed above, inability of users for moving the eyes beyond the visibility range of screen during text entry session is a major issue. So, as an way out, participants needed to memorize the phrase to be typed before starting of experiments. In this context, participants first wrote the phrase using pen and paper or listen while instructor prompting it. The total time needed for this interaction is about

Figure 5.9: Participant performing experiments

10 minutes. After practicing on paper, participants were given two practice phrases with proposed design as well as one of the other designs, chosen randomly. The order was counterbalanced across the participants. First session lasted about half an hour, and data were not considered for analysis. After completion of training, each participant on an average, composed nine texts for testing each design. Participants were instructed to balance speed and accuracy during the session, but main objective was text entry as fast as possible. On the average, each testing session took about 45 minutes. For conducting experiments, nine texts were selected and among these, one is taken from the in-domain corpus and other eight are taken from out-of-domain texts such as novels, short stories etc. for judging the efficacy of the designs. Each text contains 10 phrases containing approximately 25 characters. The selected phases are chosen in such a way that they are easy to remember. The phrase set was tested for its correlation with common English using the frequency counts in developed training corpus. The result is $r = 0.973$ for the single-letter correlation and $r = 0.908$ for the digraph correlation. Each participant performed nine sessions, each for a text, for each of the designs. At the end of each session, participants completed a subjective questionnaire.

In case of experiments with Indic language keyboard designs, after practicing on paper, participants were given two practice phrases with eye typing interfaces appeared at random order. First session took about one hour, and data were not considered for analysis. After completion of training, each participant on an average, composed 8 texts for testing. Before starting of each sessions, users assured the instructors about their memorability of the practiced set. Random ordering was followed for selecting a keyboard in each session. The order was counterbalanced across the participants. Each testing session took about eight to 15 minutes which was dependent on user's proficiency

on eye typing using the setup. On average, a gap of 45 minutes was maintained between two consecutive sessions with respect to single user. On an average, a user spent time in performing three to four experiments per day.

5.4.5 Experimental results

The main objective of the experiments was to study the text entry rates of the study designs and compare eye typing rate as well as error rate of existing gaze-based text entry interface designs with the proposed interface. Also, we require to analyze learning curve obtained based on longitudinal user studies on several design along with proposed one. Apart from this, users were also involved for subjective evaluations with respect to user friendliness, usability etc. Data for each participant were averaged for each session to form single measures per participant per session on a variety of metrics, including entry rate in wpm and error rates [184]. For analyzing study designs, five representative participants completed a total of one trial $\times$ three designs $\times$ nine sessions $=$ 27 trials. With five participants, the entire study comprised of 135 trials. On the other hand, the comparison analysis of developed interface with existing designs involved 10 participants each performed two trials $\times$ six designs $\times$ nine sessions $=$ 108 trials. So, the study consists of the results of $108 \times 10 = 1080$ trials.

5.4.5.1 Text entry rate analysis among study designs

Summary of within-subject user experiment results (calculated by averaging all user results with three designs) is depicted in Fig. 5.10. It reveals that *Design* 2 gives, on an average, 9.13 wpm ($SD = 1.23$) which is 68.84% more than *Design* 1 (effectively, visual feedback with next character highlighting did not effect much on participant's small duration text typing experiments). The proposed design in average achieved 82.97% improvement over design 1 (9.34 wpm $SD = 1.43$). As the proposed design requires sufficient learning (curve angle is not higher), the user experiment result did not acquire much benefit over Design 2.

During the eye typing sessions of participants with different designs, we also marked the corrected and uncorrected error traces and calculated the rates accordingly. The average corrected error rate for Design 1, 2 and proposed interface are 9.14%, 8.73% and 10.29%, respectively (Fig. 5.11(a)). The average rate indicates that proposed *EyeBoard++* design corrected most errors occurred and Design 1 corrected minimum. We can not claim strongly that the trend will be maintained in later user experiments, but maintaining the speed-accuracy trade-off, the proposed design contested with other

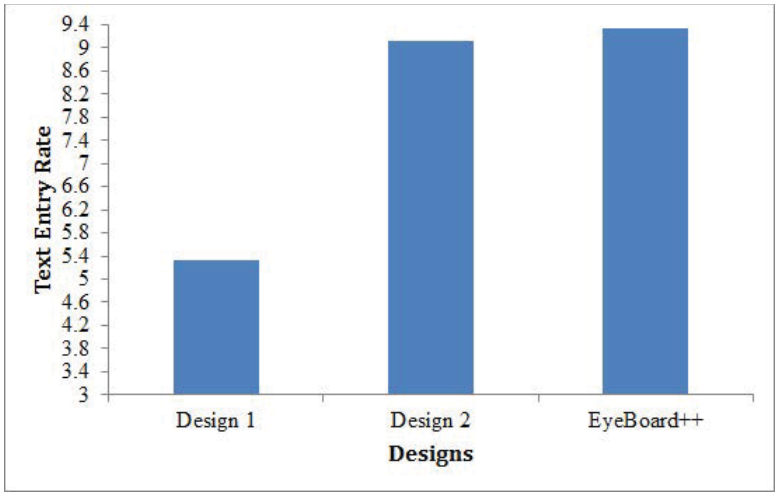
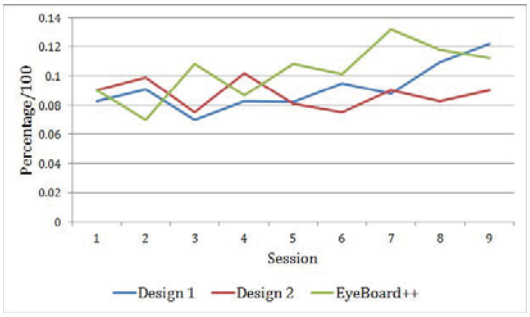


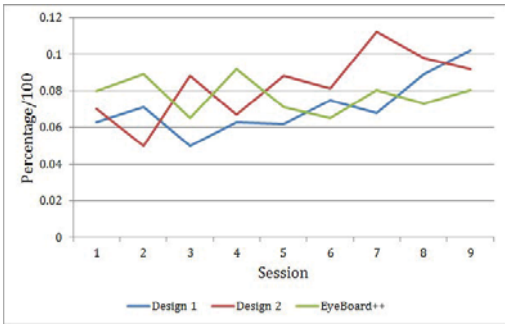
Figure 5.10: Comparison among different study designs

designs. If we concentrate on uncorrected error rate, the average results for Design 1, 2 and proposed *EyeBoard++* interface are 7.14%, 8.29% and 7.73%, respectively (Fig. 5.11(b)). The trend of this graph does not reveal any conclusive remark about uncorrected error results for different designs including proposed interface. Similarly, the total error rate results (Fig. 5.11(c)) do not come up with important remark (total error rate for Design 1, 2 and *EyeBoard++* are 16.29%, 17.02% and 18.02%, respectively; there is no significant difference in error rates between the designs ($F(2, 132) = 1.03$, n.s.) but one thing has been cleared with the outcomes that the designs set up a trend to the user for committing error as less as possible as long as they got habituated with any eye typing alternate designs.

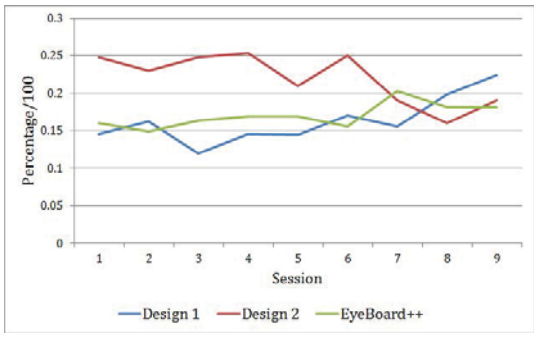
Participants completed a subjective questionnaire after each session (except the first). We collected the subjective ratings participants given against the questions with the *nonparametric Wilcoxon Matched Pairs Signed Ranks Test*. We discussed with participants about their eye strain and tiredness in $1 - 7$ Likert-scale before and after each session. The result concludes that users more or less like the proposed interface than other keyboard designs for ease of use ($z = 49.00, p < .001$). Overall, users agreed that careful and concentrated interaction with proposed design produced eye typing with maintaining optimized speed-accuracy trade-off. Users were also not confident about maintaining the same performance or seeking even better with design 1 and 2, but they assured steadily improved performance in speed and accuracy in eye typing with the *EyeBoard++* interface.



(a) Corrected error rates of three designs



(b) Uncorrected error rates of three designs



(c) Total error rates of 4 designs

Figure 5.11: Comparison between error rates of three designs

5.4.5.2 Text entry rate and error rate analysis among existing keyboards

Result of user experiments with 7 designs, based on the average speed of different sessions, trials and results, are depicted in Fig. 5.12. It reveals that *Adjustable dwell time*-based design yields 6.34% better text entry rate (4.58 wpm $SD = 1.07$) than *Scrollable keyboard* (though the design using *Adjustable dwell time* holds the same *Scrollable keyboard* interface, the additional dynamically dwell-time adjustment concept boosts up the average eye typing rate of users). Similarly, *Context switching* and *Iwrite* supported eye-typing interface, also give 5.35 wpm ($SD = 1.12$) and 6.05 wpm ($SD = 1.18$) text entry rates which are 20.39% and 32.97% more than *Scrollable keyboard* interface (*Context switching* and *Iwrite* both interfaces implement dwell-free eye typing, but the way second incorporated suits better to the participants). Basic *EyeBoard* layout gets 5.15 wpm ($SD = 1.05$) entry speed which is 16.67% better (it only concentrates on developing keyboard incurring less eye movement) than *Scrollable keyboard*. Proposed *EyeBoard++* layout achieves 48.91% more text entry rate than *Scrollable keyboard* interface (6.93 wpm and $SD = 1.16$). The analysis of variance (ANOVA) on text entry speeds

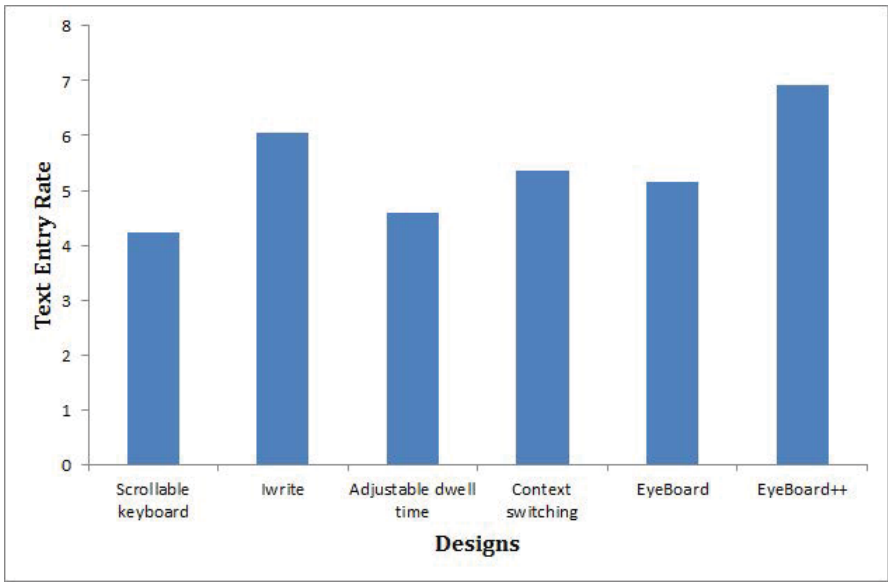


Figure 5.12: Comparison among different designs

shows that there is a significant difference between the means of user's performance on different keyboard designs ($p < 0.05$). Further, The Post-hoc using *Tukey HSD* test reveals significant difference between performance of *EyeBoard++* and other keyboard designs ($p < 0.05$). Also, for the sessions, significant difference is observed on wpm, as participants speed up with each design ($p < 0.05$).

Total error rate : Over nine sessions, the total error rate (Fig. 5.13) is 26.29% for the proposed *EyeBoard++* interface and 32.88%, 32.57%, 33.47%, 33.64%, 33.18% for *Scrollable keyboard*, *Adjustable dwell time*, *Iwrite*, *Context switching* and *EyeBoard* designs, respectively. From Fig. 5.13, it has been observed that proposed interface yields less total error rates than others. However, total error rates drop significantly over sessions ($p < 0.05$).

The results we got from the error analysis do not strictly reflect better performance of the proposed system than other designs. An analysis of variance reveals that there is no significant difference in error rates between the existing keyboard designs (n.s.).

5.4.6 Learning curve

Learning curves are typically created for measuring task performance speed over time [26]. It does not reflect the study of initial interactions with the system, rather indicates that whether more or less training is required to get habituated. Figure 5.14 indicates that proposed interface needs more initial effort to learn compared to other designs (many factors what we observed drive the inability of other systems to come up with

increased typing speed like for *Scrollable keyboard* and *EyeBoard*, a moderate amount of time were spent for dwelling and it was increased with number of characters to be typed as well as for *Iwrite*, *Context switching* keyboard, users need not to spare dwelling time, but they spent more time in correcting the character selection errors occurred during eye typing with those interface); however, after 20 to 25 sessions, proposed interface outperformed five designs except Mackenzie and Zhang's interface supporting word completion and prediction. The proposed interface reaches nearly 5.71 wpm by the 25^{th} session whereas the performance of the *Scrollable keyboard*, *Adjustable dwell time*, *Iwrite*, *Context switching* and *EyeBoard* interfaces achieve up to 5.43 wpm text entry speed in average (at max). We use standard *regression* models in the form of the curve fitting as it follows *Power law of learning*. The prediction equations and the *Squared correlation coefficients* for the curves are illustrated in Fig 5.14. The longitudinal study lasts for 60 sessions for each experienced and inexperienced users. We plot the results and construct the learning curves which inevitably reflect the increasing efficiency of users after performing several sessions (see Fig 5.14). So, the observation supports the previously stated hypothesis inevitably. The highest text entry rate achieved by user on typing similar length texts for the proposed *EyeBoard++* layout is 8.25 wpm and for the *Scrollable keyboard*, *Adjustable dwell time*, *Iwrite*, *Context switching* and *EyeBoard* designs, results are 7.31, 6.79, 6.55, 7.85 and 7.45 wpm, respectively.

5.4.7 Subjective evaluation

We collected the subjective ratings from the participants with the *nonparametric Wilcoxon Matched Pairs Signed Ranks Test*. We talked with the participants before and after each session asking them about their eye strain and tiredness in 1 to 7 Likert-scale.

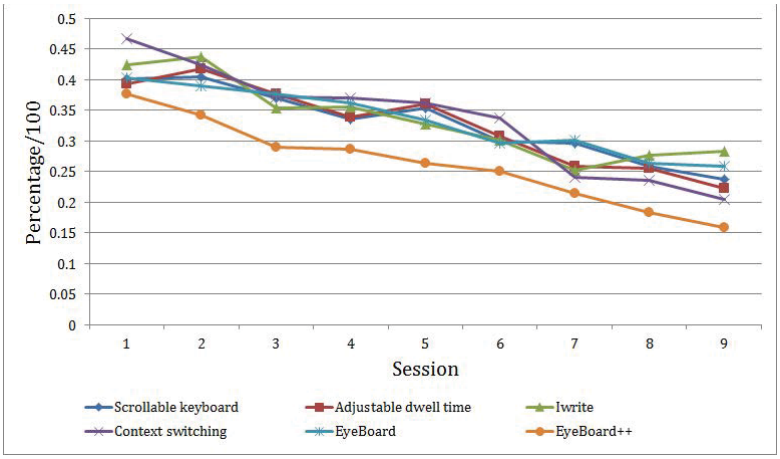


Figure 5.13: Comparison between total errors of six designs

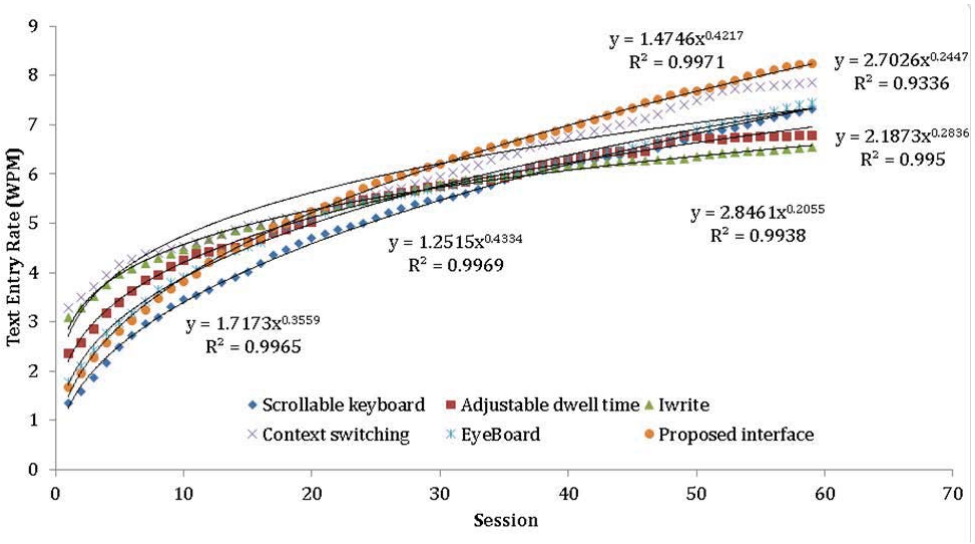


Figure 5.14: Learning curve

The result reveals that users like the proposed interface than other keyboard designs for ease of use ($z = 57.00, p < .001$) and less stressfulness ($z = -55.00, p < .001$). They found proposed interface faster ($z = 51.00, p < .01$) and thus fun to use. However, users agreed that concentration was needed for eye typing through proposed keyboard, but they could improve their eye typing skill with practice easily. They also felt that gaze typing was clearly slower than using a conventional hardware keyboard.

The level of tiredness was quantified by subtracting the first value from the later which were got from Likeart-scale. Analyzing the experimental results, we observed no significant difference between the average level of the tiredness, which is 0.35 in the first and 0.47 in the last session. Instead, the level of tiredness with respect to other interface accessing was slower. We also calculated the text entry speed, ease of use, and general fatigue after each session using a questionnaire with a scale from 1 to 5. An increment of text entry rate was observed (3.4 to 4.6). On the other hand, the observed ease of use, with average rating of 4.5, and general fatigue (≈ 3.5) remain approximately on the same level. Finally, participants were again interviewed after completion of the series of sessions. Participants felt that typing by gaze is fairly easy, easier than their expectations, but clearly slower than using a conventional, hand operated hardware/virtual keyboard. More or less, half of the participants experienced problems in using the interface operated with no dwell time. But when comfortable, participants performed the interaction with ease spending less time. Side-by-side, they admitted that after using the proposed interface for sometime, the visual feedback helped

them in finding next probable characters. After habituated with the interaction, users found 'inside' and 'outside' eye writing procedure (incorporating dwell freeness) much effective in selecting a key (effectively, if it is a word of the suggestion list). they explained the causes as a) no particular pattern need to follow, b) no specific path they need to follow, c) no fixed time need to be maintained for task completion and d) they got ample time for discontinuing the interaction from any point in between. Overall, participants admitted that they gained much higher speed and accuracy after learned the proposed system compare to other designs after learning. Many of them felt that the proposed interface had been worked as proper eye typing aid on which they can rely upon.

5.4.8 Experimental results with Indic language keyboard designs

Several user experiments were performed to judge the effectiveness of the proposed system with respect to performance evaluation metrics namely *Text entry rate* and *Total error rate*. Three *EyeBoard++* designs developed for Hindi, Bengali and Telugu were taken into consideration for this purpose. Users were also involved for subjective evaluations with respect to *user friendliness*, *usability* etc. Data for each participant are averaged for each session to form single measures per participant per session on the basis of two aforementioned metrics [184]. Participants complete a total of three trials $\times$ six sessions $=$ 18 trials for each language. With six participants, the entire study comprised of 108 trials.

Text Entry Rate: For Hindi, Bengali and Telugu language, *EyeBoard++* achieves text entry rates, on an average, of 9.82, 9.46 and 9.04 WPM, respectively.

Total Error Rate: Combination of *Corrected* and *Uncorrected* error rates, the *Total Error Rate* is calculated on an average as 2.80%, 3.67% and 3.92% for *EyeBoard++* designs for Hindi, Bengali and Telugu languages, respectively.

Subjective Evaluation: We collect the subjective ratings from the participants with the *nonparametric Wilcoxon Matched Pairs Signed Ranks Test*. We talked with the participants before and after each session asking them about their eye strain and tiredness in $1-7$ Likert-scale. The result, averaged over all the three languages, reveals that users like the proposed word prediction and completion interface than other keyboard designs for ease of use ($z = 57.00, p < .001$) and less stressfulness ($z = -55.00, p < .001$). They find *EyeBoard++* interface faster ($z = 51.00, p < .01$) and thus fun to use. However, users agreed that concentration was needed for eye typing through proposed keyboard, but they could improve their eye typing skill with practice easily. They also felt that gaze typing was clearly slower than using a conventional hardware keyboard.

5.5 Summary

In this chapter, we present gaze-based text entry systems proposed for English as well as Indic language based text entry. By analyzing the user experiment results, it is evident that after training, users achieve faster eye typing speed using the proposed interface in English than existing designs. In the proposed interface, the compact on-screen keyboard holds optimal size of the key, space between keys and magnification factor of each key which yield optimal eye movement during eye typing. The augmented word completion and prediction modules help users in typing faster with gaze. Nevertheless, speed-accuracy trade-off was prominently present in user experimental results. The error rates achieved by users reduces slightly with the proposed interface. So, it can be said that augmenting error correction supports participants in correcting syntactic and semantic level errors. The computer proficient people, at least, are not now wasting time by searching the key. The proposed interface provides word=level prediction facility which enhances users' text entry rate significantly. Error correction facility is initially seems to be not effective to users. But, as the learning rate is quick, users pick up the speed after few sessions of eye typing and feeling comfortable afterwords. Surprisingly, it has been observed that people, after learning the interface, are not using the character highlighter as much as the word prediction and completion. We explore two types of controlled eye movement for selecting a key without fixing the gaze on a key button (i.e. dwell time) for some time. This interaction after introducing dwell-freeness is initially seems to be harder to users. But, as the learning rate is quick, users pick up the speed after few sessions of eye typing and feeling comfortable afterwords. The limited screen space acquired in proposed interface also offers an advantage over off-screen targets in limiting saccade distance to the dimensions of the window of proposed interface.

We also extend the mechanism for Indic languages. We design the on-screen keyboard as a two layer layout as Indic language alphabet contain many more characters than English. To switch from one layout to the others, one needs to select navigational key present in each layout. As no such text entry system exists in Indic language, we conduct user experiments with our proposed layouts developed in Hindi, Bengali and Telugu languages.

Chapter 6

Conclusion and Future Research

The main objective of our research is to develop text entry systems suitable for mouse-, touch- and gaze-based interactions for text entry in Indic languages. The outcomes of our research are discussed in Chapter 3 to 5 in this thesis. In this chapter, we summarize some salient features so far the research contributions are concerned. This research considers some assumptions, which may raise about the validity of our claim. We point out the threats to validity of those considerations. Finally, we give some future research directions in our area of research.

6.1 Virtual keyboard design in Indic languages

Text entry rate with Indic character mapped QWERTY keyboard, compare to English, is less [174] even worse in the context of text entry through virtual keyboard interfaces ($5 - 6$ wpm) [96, 97]. Moreover, a large alphabet set and linguistic complexities not only increase the chance of user error occurrence during text entry, but also induce many interaction issues with small-sized mobile device interfaces. To the best of our knowledge, existing Indic language text entry mechanisms fail to give support in correcting user errors. Therefore, it is necessary to support text composition task and facilitate the users to perform the task effortlessly with better text entry methodologies. Moreover, while developing existing text entry interfaces for composing texts in Indic scripts, the device and interaction specific issues are grossly ignored. Some of them are, large mouse movement and key searching time while composing texts in desktop devices through virtual keyboard with large alphabet set, *fat finger* [24] problem while accessing small-sized buttons in touch-enabled mobile devices, accommodating large alphabet set within small screen area etc. Overcoming the above mentioned issues, we have designed virtual

keyboards suitable for large and small touch-enabled mobile display devices facilitating both mouse and touch-based interactions for Indic languages.

We propose to retain good design principles of English language virtual keyboard in our design suitable for large screen devices. This guides us to have the merits of existing virtual keyboards into the proposed one. Next, an optimum multi-zonal arrangement of keys in the layout has been made. This enables us to reduce eye and mouse movements and thus improve text entry rate. Then we have augmented the mouse movement time optimized keyboard with a strategy that deals with inflexions to reduce further mouse movements. Based on our approach, we have proposed a generic structure suitable for mouse-based interaction in large screen devices (named as *iLiPi*) which has been mapped to three Indic languages namely Hindi, Bengali and Telugu.

We have further designed a keyboard suitable for touch-enabled mobile devices where display area is an important constraint. The developed keyboard layout (named as *mLiPi*) displays minimum keys in the initial design with every key size fitted with finger or thumb size to avoid *fat finger* problem. Apart from the characters on the initial layout, rest of the characters are displayed dynamically, based on already selected characters. The basic *mLiPi* keyboard design ensures an effective solution of accommodating large alphabet set within area-constrained display devices. Next, we have proposed a mechanism which after selecting a character, predicts the next probable characters and shows them in a panel placed within the keyboard. This design principle not only enhances the text typing rate and minimizes the key searching time, but also helps user to avoid character level errors while typing.

6.2 Word-level prediction in Indic languages

Indian language texts are phonetic and most of them fall under alphasyllabary category [97]. Many characteristics present in those scripts, like presence of large character set, *glyph*, *halant*, conjunct, presence of complex input sequence, typographic variation and presence of phonetic and graphical similar characters etc. [64, 96, 97, 115, 168, 174], make the text composition task more challenging and error prone as well as affect the text entry rate.

We have developed an efficient word-level prediction mechanism which suggests to be completed and next words based on previous context and currently typed text. It can be augmented with any language, specifically Indic languages which are having typical linguistic features and complexities. This mechanism, in fact, predicts correct words even in presence of typing error within the composed word chunk. We have implemented

a multi-variate method of ranking the words to be displayed in the prediction list where features are chosen based on phonetic or graphical similarity between characters, typographical error, word probability and recency of the chosen words. The mechanism also handles errors occurred at the first character of typed word chunk by implementing confusion set. The developed mechanism, after augmenting with text entry interface in three Indic languages namely Hindi, Bengali and Telugu, achieves significantly higher text entry rate and keystroke savings than the other word prediction methodologies. Language model files used in this mechanism are developed from *Wikipedia* source in corresponding languages. Each unigram file contains 65001 unique word entries, which makes the word combination entries large in the bigram and trigram language files. As a result, files size become large. We have aimed to run the developed word prediction system meant for Indic languages in a memory and speed constrained mobile device in order to enhance the text composition rate. For this, we have implemented *Arithmetic coding*, an efficient data compression algorithm, to compress the unigram, bigram and trigram language files in Hindi, Bengali and Telugu languages.

6.3 Gaze-based text entry mechanism

Eye gaze has been evolved as an alternate interaction mode of text entry for English and other popular languages, however, it is yet to be explored in Indic languages. In this work, we have analyzed common issues of developing gaze-based text entry (*Eye typing*) interfaces and then investigated language or specific interaction related issues. With reference to the work related to the layout of eye typing interface, the approaches [79,145,186,208,239] are mainly focused on utilizing less screen area deciding the reduced size of the key buttons, smaller space between buttons and minimum number of buttons present in keyboards [79, 145, 186, 208]. However, there is no work reported which develops layout designing strategy focusing on eye movement minimization. Several approaches [79,145,186,208,239] have been proposed for minimizing [79,208,239] as well as diminishing [145, 186] dwell time during eye typing. Still, there exists a scope to explore a better approach which can minimize the chance of occurring error alongwith diminish dwell time. To the best of our literature review, no such existing work on developing eye typing interface in Indic language has been reported so far.

We have designed an efficient eye typing interface consisting of an on-screen keyboard, a predicted character list, and a text entry area where composed text chunks are displayed. In order to do that, we first analyzed the interaction similarity between mouse and gaze-based text entry task performance. Then, we have explored virtual keyboard

design principles to find out the best design suitable for eye typing tasks. After that, we have further modified the key arrangement in the selected design to achieve better text entry performance. In a dwell-based eye typing system, users need to fix their eyes for a prolonged time, that is, dwell time, on a key to select it, which effectively enhances the total time of text entry. Mechanisms to minimize the dwell time also enhance the chance of wrong selection of characters, which is known as *midus touch* problem. We have developed a mechanism which allows user to select the intended character through performing an eye gesture. While designing the mechanism, we have decided the accurate gestural pattern from some alternate patterns by conducting user experiments and analyzing the results. We have also developed a next character highlighting mechanism which helps user to find desired character quickly in the keyboard. Based on the proposed eye typing mechanism, we have designed text entry interfaces in three Indic languages namely Hindi, Bengali and Telugu.

6.4 Threats to Validity

Our experiments of proposed text input mechanisms are involved with different languages and interaction techniques. Eventually, the experimental results we have reported in this thesis are subjected to the validity of the available resources and assumptions on values of parameters. In the following, we discuss the validity of our experiments and experimental results.

6.4.1 Internal validity

In our proposed text entry mechanisms, some parameters may affects the experimental findings. Hence, we asses the internal validity of the proposed systems based on the different parameters in the following.

Text entry rate may be influenced by some virtual keyboard design parameters like layout size, sizes of keys, sizes of fonts on keys, background and foreground colors of the fonts etc. Hence, our experimental results are under assumptions of default settings of these parameters. Although these settings have been chosen after a number of trials and consultation with domain experts. It would be another interesting matter to validate the results with other platform such as Smartwatch, iPod etc. In the present work, we have performed experiments to desktop and touch-enabled mobile devices, only.

Proposed word-level prediction mechanism is included within frequency-based *iLiPi* and *mLiPi* keyboard layouts where experiments are conducted. Consequently, the results may vary with other keyboard layouts.

For gaze-based text entry experiment, we have developed a low-cost eye tracking set-up which can be easily replicated. However, the accuracy of this still is not up to the mark and thus, the applicability confines within performing experiments in controlled environments. We further have tried to improvise the situation by fixing the infrared (IR) filters within visible range and placing the camera as close to eye and on the path of eye and centering position of the screen for more accurately detecting eye gaze during calibration phase. We still struggle with the situation where during experiment, if users see outside of the screen area, calibration gets disturbed. Also, similar to the study outcomes of Räihä and Ovaska [164], our observation finds that the hand-made infrared light can cause dryness and irritation of the eyes during a long period of use. To prevent this, we currently relieve participants after performing longitudinal experiments as an ad-hoc solution.

6.4.2 External validity

For virtual keyboard based text entry experiment in desktop device, we have followed two modes of evaluation: user and model-based. For the user-based evaluation, we have considered 35 to 50 users with various backgrounds (novice, intermediate and expert). On the other hand, in case of small display devices, we have conducted two studies namely *first-time usability study* and *longitudinal evaluation*. We have considered 33 users for the evaluation. Further, we follow two strategies: *Form fill in* (where users enter their own text) and *Read and type* (with some carefully chosen texts). The results of these user evaluations are therefore subject to the limitations on number of subjects involved, forms considered and texts chosen in our experiments. In model-based evaluation followed for evaluating virtual keyboards in large display device, we have considered Fitts'-digraph law for measuring the mouse movements and *Keystroke-level model* to measure other activities (motor activity for button press and release, cognitive activity for deciding the next target etc.). Using other task-oriented evaluation tool namely ACT-R [5], model-based evaluation can also be performed.

We have conducted experiments to judge the efficacy of word-level prediction mechanism with 32 Hindi, 30 Bengali and 28 Telugu participants with varying level of computer knowledge, reading/writing capability, occupation, and educational background. Our achieved results are thus limited upto the validation with a small chunk of the actual population. The user experiments (also simulated evaluations for desktop devices) of prediction mechanisms augmented text entry interface are conducted in desktop and mobile devices. The result may vary with other devices with different form factors.

On the other hand, user experiments are not conducted with the potential user group for eye tracking software and people with motor disabilities. The major objective of the work rather lies in developing prototype interface designing for able-bodied participants who are proficient with mouse or touch-based text typing.

6.4.3 Construct validity

Toward virtual keyboard designing for desktop device in Indic languages, we consider data in user-based evaluation after a number of sessions (40 to 50 sessions in a wide time frame of three months), when we confirm insignificant fluctuations ($\pm 5\%$) in the observations. The optimal arrangement of character placing in the layout is governed by the factors: bigram probability and genetic algorithm (GA). We have considered the *Wikipedia* corpus to calculate character-level unigram, bigram and trigram probabilities for languages aimed in this work. It has been verified that sizes of the corpus are comparable to the sizes of other sources such as daily newspapers. Optimization problem solving could be carried out with optimization methodology such as Simulated Annealing (SA), Ant Colony Optimization (ACO), Particle Swarm Optimization (PSO) alternative to GA. We have chosen GA because of its better fitting with the objective function, higher convergence rate and hence faster computation.

In order to develop the language model for word-level prediction algorithm, we use *Wikipedia* corpus for Hindi, Bengali and Telugu languages. The observed results in these three languages are based on the above mentioned corpus. Instead of *Wikipedia*, we could use other corpus such as newspaper text, the result would have been different.

6.5 Future Scope of Work

While we have made significant improvement in the development of text entry mechanisms suitable for mouse, touch and gaze-based interactions in desktop and mobile devices in Indic languages. This thesis opens up several future avenues for research. Some of them are mentioned below.

- Virtual keyboards in Indic languages contain a large set of characters. Finding a target character in such a virtual keyboard demands a high reaction time and decreases the text entry rate. One of the possible solutions is highlighting next probable characters which can be implemented further.
- While composing text with virtual keyboard in desktop devices, enough eye movements are involved to search desired word within the predicted word list.

The scenario also demands hand movements to select a word from the prediction window. In order to achieve a better text entry rate, the eye and hand movements need to be minimized. One way to achieve minimization can be to place the prediction window at a precisely calculated location instead of at any ad hoc place, which will be explored in future.

- To enhance the performance of error correction mechanism of proposed word-level prediction system further, a context based error correction facility can be incorporated.
- There is a lack of calculating errors during text composition through word prediction system. A new metric can further be developed to judge the error behavior during text composition using word prediction.
- The user adaptation and personalization can be incorporated for further enhancement of user comfort while accessing text entry interfaces in desktop and mobile devices.
- In a gaze-based text entry scenario, *midas touch* problem refers to selection of neighbouring characters leading to incorrect text entry. Further research can be conducted in order to mitigate such kind of problem and produce accurate texts.

References

[1] J. S. Agustin, H. Skovsgaard, E. Mollenbach, M. Barret, M. Tall, D. W. Hansen, and J. P. Hansen, "Evaluation of a Low-Cost Open-Source Gaze Tracker," in *Proceedings of the 2010 Symposium on Eye-Tracking Research & Applications*. New York, NY, USA: ACM, 2010, pp. 77–80.

[2] O. H. Alliance, "Alliance Members," available: http://www.openhandsetalliance.com/oha_members.html, accessed on November 2014.

[3] N. Alm, L. J. Arnott, and A. F. Newell, "Prediction and Conversational Momentum in an Augmentative Communication System," *Communications of the ACM*, vol. 35, no. 5, pp. 46–57, 1992.

[4] C. Amma, M. Georgi, and T. Schultz, "Airwriting: Hands-free Mobile Text Input by Spotting and Continuous Recognition of 3D-space Handwriting with Inertial Sensors," in *Proceedings of International Symposium on Wearable Computers (ISWC)*. Newcastle, UK: IEEE, 2012, pp. 52–59.

[5] J. R. Anderson, D. Bothell, M. D. Byrne, S. Douglass, C. Lebiere, and Y. Quin, "An Integrated Theory of the Mind," *Psychological Review*, vol. 111, no. 4, pp. 1036–1060, 2004.

[6] L. Anthony and J. O. Wobbrock, "A Lightweight Multistroke Recognizer for User Interface Prototypes," in *Proceedings of Graphics Interface (GI)*. Ottawa, Canada: Canadian Information Processing Society, 2010, pp. 245–252.

[7] L. Anthony and J. O. Wobbrock, "$N-protractor: A Fast and Accurate Multistroke Recognizer," in *Proceedings of Graphics Interface (GI)*. Toronto, Canada: Canadian Information Processing Society, 2012, pp. 117–120.

[8] AppBrain, "Sparsh indian keyboard," 2010, available: http://www.appbrain.com/app/sparsh-indian-keyboard/com.sparsh.inputmethod#descriptionsection, accessed on January 2014.

[9] L. Balasubramaniam, "Translation Article Knowledgebase - Spell it Right! (in the Context of Hindi)," 2006, available: http://www.proz.com/translation-articles/articles/705/, accessed on October 2014.

[10] O. Bau and W. E. Mackay, "OctoPocus: a Dynamic Guide for Learning Gesture-based Command Sets," in *Proceedings of the Symposium on User Interface Software and Technology (UIST)*. Monterey, USA: ACM, 2008, pp. 37–46.

[11] N. Bee and E. André, "Writing with Your Eye: A Dwell Time Free Writing System Adapted to the Nature of Human Eye Gaze," in *Perception in Multimodal Dialogue Systems*, ser. Lecture Notes in Computer Science, E. André, L. Dybkjær, W. Minker, H. Neumann, R. Pieraccini, and M. Weber, Eds. Springer Berlin Heidelberg, 2008, vol. 5078, pp. 111–122.

[12] T. Bellman and I. S. MacKenzie, "A Probabilistic Character Layout Strategy for Mobile Text Entry," in *Proceedings of Graphics Interface (GI)*. Toronto, Canada: ACM, 1998, pp. 168–176.

[13] A. Bharati, P. Rao, R. Sangal, and S. M. Bendre, "Basic Statistical Analaysis of Corpus and Cross Comparision," in *Proceedings of ICON*, Hyderabad, India, 2002.

[14] S. Bhattacharya and S. Laha, "Bengali Text Input Interface Design for Mobile Devices," *Universal Access in the Information Society (UAIS)*, vol. 12, no. 4, pp. 441–451, 2013.

[15] S. Bhattacharya, A. Basu, and D. Samanta, "Performance Models for Virtual Scanning Keyboards: Reducing User Involvement in the Design," in *Proceedings of the International Conference on Information and Communication Technologies for Development (ICTD)*, Bangalore, India, 2007.

[16] X. Bi, B. A. Smith, and S. Zhai, "Quasi-qwerty Soft Keyboard Optimization," in *Proceedings of the Conference on Human Factors in Computing Systems*. Atlanta, USA: ACM, 2010, pp. 283–286.

[17] X. Bi, Y. Li, and S. Zhai, "FFitts Law: Modeling Finger Touch with Fitts' Law," in *Proceedings of the Conference on Human Factors in Computing Systems*. Paris, France: ACM, 2013, pp. 1363–1372.

[18] X. Bi, B. A. Smith, and S. Zhai, "Multilingual Touchscreen Keyboard Design and Optimization," *Human–Computer Interaction*, vol. 27, no. 4, pp. 352–382, 2012.

[19] X. Bi and S. Zhai, "Bayesian Touch: A Statistical Criterion of Target Selection with Finger Touch," in *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST)*. St. Andrews, UK: ACM, 2013, pp. 51–60.

[20] R. Block, "The iPhone is not a Smartphone. 9 january 2007. retrieved 11 july 2010," 2007, available: http://www.engadget.com/2007/01/09/sling-medias-clip-sling-sling-video-clips-to-anyone, accessed on November 2014.

[21] P. Boissiere, "An Overview of Existing Writing Assistance System," in *Proceedings of the IFRATH Workshop*, Paris, France, 2003.

[22] P. Boissiere and D. Dours, "VITIPI: Versatile Interpretation of Text Input by Persons with Impairments," in *Proceedings of the International conference on Computers helping people with special needs. Part I.* R. Oldenbourg Verlag GmbH, 1996, pp. 165–172.

[23] P. Boissière and D. Dours, "VITIPI: A Universal Writing Interface for All," in *Proceedings of the 6th ERCIM Workshop on User Interfaces for All*, 2000.

[24] S. Boring, D. Ledo, X. Chen, N. Marquardt, A. Tang, and S. Greenberg, "The Fat Thumb: Using the Thumb's Contact Size for Single-handed Mobile Interaction," in *Proceedings of MobileHCI*. San Francisco, USA: ACM, 2012, pp. 39–48.

[25] C-DAC: GIST, "Research Areas - Search Engine Technologies," http://pune.cdac.in/html/gist/research-areas/set.aspx, 2008, accessed on September, 2010.

[26] S. K. Card, T. P. Moran, and A. Newell, *The Psychology of Human Computer Interaction*. Hilldale, NJ: Elbaum Associates, 1983.

[27] A. Carlberger, J. Carlberger, T. Magnuson, M. S. Hunnicutt, S. E. Palazuelos-Cagigas, and S. A. Navarro, "Profet, a New Generation of Word Prediction: An Evaluation Study," in *Proceedings of the 2nd Workshop on NLP for Communication Aids*, 1997.

[28] A. Carlberger, T. Magnuson, J. Carlberger, H. Wachtmeister, and S. Hunnicutt, "Probability-Based Word Prediction for Writing Support in Dyslexia," in *Proceedings of Fonetik Conference*, 1997, pp. 17–20.

[29] J. Carlberger, "Design and Implementation of a Probabilistic Word Prediciton Program," Master's thesis, Computer Science, Nada, KTH, Stockholm (Sweden), 1997.

[30] CDAC, "Indian Language Search Engine Technologies - Problems and Solutions," 2010, available: http://iplugin.cdac.in/search-engine.htm, accessed on September 2014.

[31] CDAC, "Problems with Existing Unicode Based Engines," 2010, available: http://pune.cdac.in/html/gist/research-areas/set.aspx, accessed on June 2014.

[32] T. Chakraborty, S. Sarcar, and D. Samanta, "Design and Evaluation of a Dwell-free Eye Typing Technique," in *Extended Abstracts on Human Factors in Computing Systems*. Toronto, Canada: ACM, 2014, pp. 1573–1578.

[33] J. Clawson, K. Lyons, A. Rudnick, R. A. Iannucci, and T. Starner, "Automatic Whiteout++: Correcting Mini-QWERTY Typing Errors using Keypress Timing," in *Proceeding of the Annual Conference on Human Factors in Computing Systems*. Florance, Italy: ACM, 2008, pp. 573–582.

[34] J. Clawson, A. Rudnick, K. Lyons, and T. Starner, "Automatic Whiteout: Discovery and Correction of Typographical Errors in Mobile Text Input," in *Proceedings of the conference on Human-computer interaction with mobile devices and services (MobileHCI)*. ACM, 2007.

[35] CMU-SLM, "The CMU Statistical Language Modeling (SLM) Toolkit," Available: http://homepages.inf.ed.ac.uk/lzhang10/slm.html, accessed on January 2014.

[36] Cnet.com, "The Android Atlas," available: http://www.cnet.com/android-update, accessed on November 2014.

[37] J. F. Collins and L. K. Blackwell, "Effects of Eye Dominance and Retinal Distance on Binocular Rivalry," *Perceptual Motor Skills*, vol. 39, pp. 747–754, 1974.

[38] U. Consortium, "Unicode Normalization Forms," 2010, available: http://www.unicode.org/reports/tr15/, Accessed on November 2014.

[39] U. Consortium, "South Asian Scripts-I," 2011, available: http://www.unicode.org/versions/Unicode5.0.0/ch09.pdf, accessed on January 2011.

[40] P. Constable, "Proposal on Clarification and Consolidation of the Function of ZERO WIDTH JOINER in Indic Scripts," Review document, Unicode Consortium,

06 2004, available: http://www.unicode.org/review/pr-37.pdf, accessed on April 2011.

[41] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, Third Edition.* MIT press, 2001.

[42] F. Coulmas, *The Writing Systems of the World.* Wiley-Blackwell, 1991.

[43] K. Dandekar, B. I. Raju, and M. A. Srinivasan, "3-d finite-element models of human and monkey fingertips to investigate the mechanics of tactile sense," *Journal of biomechanical engineering*, vol. 125, no. 5, pp. 682–691, 2003.

[44] J. J. Darragh and I. H. Witten, "Adaptive Predictive Text Generation and the Reactive Keyboard," *Interacting with Computers*, vol. 3, no. 1, pp. 27–50, 1991.

[45] R. L. Deininger, "Human Factors Engineering Studies of the Desgn and Use of Pushbutton Telephone Sets," *Bell System Technical Journal*, vol. 39, pp. 995–1012, 1960.

[46] M. Dell'Amico, J. C. D. Diaz, M. Iori, and R. Montanari, "The Single-finger Keyboard Layout Problem," *Computers & Operations Research*, vol. 36, no. 11, pp. 3002–3012, 2009.

[47] Deloitte, "Short Messaging Services versus Instant Messaging: Value versus Volume," available: https://www2.deloitte.com/content/dam/Deloitte/au/Documents/technology-media-telecommunications/deloitte-au-tmt-short-messaging-services-versus-instant-messaging-011014.pdf, accessed on November 2014.

[48] A. Dix, J. Finley, G. Abowd, and R. Beale, *Human-computer Interaction*, 3rd ed. London, UK: Prentice-Hall Inc., 2004.

[49] M. Dunlop and J. Levine, "Multidimensional Pareto Optimization of Touchscreen Keyboards for Speed, Familiarity and Improved Spell Checking," in *Proceedings of the Conference on Human Factors in Computing Systems.* Texas, USA: ACM, 2012, pp. 2669–2678.

[50] M. D. Dunlop, N. Durga, S. Motaparti, P. Dona, and V. Medapuram, "QWARTH: An Optimized Semi-ambiguous Keyboard Design," in *Proceedings of MobileHCI.* San Francisco, USA: ACM, 2012, pp. 23–28.

[51] A. Dvorak, N. Merrick, W. Dealey, and G. Ford, *Typewriting Behavior*. New York: American Book Company., 1936.

[52] eMarketer, "Smartphone Users Worldwide Will Total 1.75 Billion in 2014," 2014, available: http://www.emarketer.com/Article/Smartphone-Users-Worldwide-Will-Total-175-Billion-2014/1010536, accessed on November 2014.

[53] L. Ergonomics, "Clevertexting," available: http://www.paninikeypad.com/images/media/Thubms_up_to_clever_texting.pdf, accessed on July 2014.

[54] A. Fazly and G. Hirst, "Testing the Efficacy of Part-of-Speech Information in Word Completion," in *Proceedings of the 10th Conference of the European Chapter of the Association for Computational Linguistics*, 2003, pp. 9–16.

[55] A. Fazly, "The Use of Syntax in Word Completion Utilities," Master's thesis, Department of Computer Science, University of Toronto, 2002.

[56] A. R. Fischer, K. J. Price, and A. Sears, "Speech-based Text Entry for Mobile Handheld Devices: An Analysis of Efficacy and Error Correction Techniques for Server-based Solutions," *International Journal of Human-Computer Interaction*, vol. 19, no. 3, pp. 279–304, 2005.

[57] Fitaly, "The One-Finger Keyboard," http://www.fitaly.com/fitaly/fitaly. htm, 2009, accessed on July, 2010.

[58] P. M. Fitts, "The Information Capacity of the Human Motor System in Controlling the Amplitude of Movement," *Journal of Experimental Psychology*, vol. 47, pp. 381–391, 1954.

[59] N. Garay-Vitoria and J. Abascal, "A Comparison of Prediction Techniques to Enhance the Communication Rate," in *User-Centered Interaction Paradigms for Universal Access in the Information Society*. Springer Berlin / Heidelberg, 2004, vol. 3196, pp. 400–417.

[60] N. Garay-Vitoria and J. Abascal, "Text Prediction Systems: a Survey," *Universal Access in the Information Society*, vol. 4, no. 3, pp. 188–203, 2006.

[61] gate2home, "Online Onscreen Virtual Hindi Keyboard Emulator on the Internet," http://www.gate2home.com/?language=hi, 2010, accessed on October, 2010.

[62] Geek.com, "PDA Review: Ericsson R380 Smartphone," available: http://www.geek.com/hwswrev/pda/ericr380, accessed on November 2014.

[63] C. O. Getschow, M. J. Rosen, and C. Goodenough-Trepagnier, "A Systematic Approach to Design a Minimum Distance Alphabetical Keyboard," in *Proceedings of RESNA (Rehabilitation Engineering Society of North America)*, Minnesota, USA, 1986, pp. 396–398.

[64] P. K. Ghosh and D. E. Knuth, "An Approach to Type Design and Text Composition in Indian Scripts," Ph.D. dissertation, Stanford University, 1983.

[65] C.-D. GIST., "A Document for Enhanced InScript Keyboard Layout 5.2," available: http://malayalam.kerala.gov.in/images/8/80/Qwerty_enhancedinscriptkeyboardlayout.pdf, accessed on April 2014.

[66] D. Goldberg and D. Richardson, "Touch-typing With a Stylus," in *Proceedings of Conference on Human Factors in Computing Systems (INTERCHI)*. Amsterdam, The Netherlands: ACM, 1993, pp. 80–87.

[67] J. Goodman, G. Venolia, K. Steury, and C. Parker, "Language Modeling for Soft Keyboards," in *Proceedings of the International Conference on Intelligent User Interfaces (IUI)*. San Francisco, USA: ACM, 2002, pp. 194–195.

[68] Google, "Indic Onscreen Keyboard iGoogle Gadgets," 2009, available: http://www.google.com/ig, accessed on October 2014.

[69] Google, "Google Input Tools on Windows," 2014, available: www.google.com/inputtools/windows, accessed on October 2014.

[70] D. Gopher and D. Raij, "Typing with a Two-hand Chord Keyboard: Will the QWERTY become Obsolete?" *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 18, no. 4, pp. 601–609, 1988.

[71] GrabStats, 2014, available: http://www.grabstats.com/statmain.aspx?StatID=402, accessed on October 2014.

[72] N. Green, J. Kruger, C. Faldu, and R. S. Amant, "A Reduced QWERTY Keyboard for Mobile Text Entry," in *Extended Abstracts on Human Factors in Computing Systems*. Vienna, Austria: ACM, 2004, pp. 1429–1432.

[73] T. R. Group, "Email Statistics Report, 2014-2018," April 2014, available: http://www.radicati.com/wp/wp-content/uploads/2014/01/Email-Statistics-Report-2014-2018-Executive-Summary.pdf, accessed on June 2014.

[74] D. L. Grover, M. T. King, and C. A. Kuschler, "Reduced Keyboard Disambiguating Computer," USA Patent US5 818 437 A, July 6 Oct, 1998.

[75] J. T. Grudin, *Cognitive Aspects of Skilled Typewriting*. Springer-Verlag, New York, 1983, ch. Error Patterns in Novice and Skilled Transcription Typing, pp. 121–143.

[76] Guruji, "Indian Internet Search Engine," 2009, available: http://www.guruji.com/hi/index.html, accessed on October 2014.

[77] J. P. Hansen, D. W. Hansen, and A. S. Johansen, "Bringing Gaze-based Interaction back to Basics," in *Proceedings of HCII*, New Orleans, USA, 2001, pp. 325–328.

[78] J. P. Hansen, A. S. Johansen, D. W. Hansen, K. Itoh, and S. Mashino, "Command Without a Click: Dwell Time Typing by Mouse and Gaze Selections," in *Proceedings of INTERACT*, Zurich, Switzerland, 2003, pp. 121–128.

[79] J. P. Hansen, A. S. Johansen, D. W. Hansen, K. K. Itoh, and S. Mashino, "Language Technology in a Predictive, Restricted On-screen Keyboard with Ambiguous Layout for Severely Disabled People," in *Proceedings of EACL*, Budapest, Hungary, 2003.

[80] I. L. Hetherington, "PocketSUMMIT: Ssmall-footprint Continuous Speech Recognition," in *Proceedings of INTERSPEECH*, Antwerp, Belgium, 2007, pp. 13–51.

[81] D. J. Higginbotham, "Evaluation of Keystroke Savings Across Five Assistive Communication Technologies," *Augmentative and Alternative Communication*, vol. 8, pp. 258–272, 1992.

[82] L. Hinkle, A. Brouillette, S. Jayakar, L. Gathings, M. Lezcano, and J. Kalita, "Design and Evaluation of Soft Keyboards for Brahmic Scripts," *ACM Transactions on Asian Language Information Processing*, vol. 12, no. 2, pp. 6:1–6:37, 2013.

[83] G. Hirst and A. Budanitsky, "Correcting Real-word Spelling Errors by Restoring Lexical Cohesion," *Natural Language Engineering*, vol. 11, no. 1, pp. 87–111, 2005.

[84] T. P. Hong, H. S. Wang, W. Y. Lin, and W. Y. Lee, "Evolution of Appropriate Crossover and Mutation Operators in a Genetic Process," *Applied Intelligence*, vol. 16, no. 1, pp. 7–17, 2001.

[85] S. Hoober, "How do users really hold mobile devices?" 2014, available: http://www.uxmatters.com/mt/archives/2013/02/how-do-users-really-hold-mobile-devices.php, accessed on July 2014.

[86] D. Huggins-Daines, M. Kumar, A. Chan, A. W. Black, M. Ravishankar, and A. I. Rudnicky, "Pocketsphinx: A Free, Real-time Continuous Speech Recognition System for Hand-held Devices," in *Proceedings of the Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2006.

[87] N. Hughes, "The War For Your Wrist," 2013, available: http://appleinsider.com/articles/13/03/21/new-samsung-smart-watch-will-be-companys-third-stab-at-wrist-accessory, accessed on November 2014.

[88] G. Iannizzotto and L. Vita, "A Multiscale Turning Angle Representation of Object Shapes for Image Retrieval," in *Visual Information and Information Systems*. Springer, 1999, pp. 609–616.

[89] IBM, "SPSS Statistics," http://www-01.ibm.com/software/analytics/spss/products/statistics/, accessed on June, 2014.

[90] R. Ishida, "An Introduction to Writing Systems & Unicode: A review of script Characteristics Affecting Computer-based Script Support and Unicode," Available: http://people.w3.org/rishida/docs/unicode-tutorial, 2010, accessed on January 2014.

[91] P. Isokoski, "Text Input Methods for Eye Trackers using Off-screen Targets," in *Proceedings of the Symposium on Eye tracking Research & Applications (ETRA)*. Florida, USA: ACM, 2000, pp. 15–21.

[92] P. Isokoski, "Performance of Menu-augmented Soft Keyboards," in *Proceedings of the Conference on Human factors in Computing Systems*. Vienna, Austria: ACM, 2004, pp. 423–430.

[93] P. Isokoski, "Existing Text Input Methods," 2004, available: http://www.sis.uta.fi/~pi52316/g/node6.html, accessed on Novemeber 2014.

[94] T. Joachims, "Optimizing Search Engines using Clickthrough Data," in *Proceedings of the SIGKDD international conference on Knowledge discovery and data mining*. Alberta, Canada: ACM, 2002, pp. 133–142.

[95] T. Joachims, "Support Vector Machine for Ranking," Website, 2009, http://www.cs.cornell.edu/People/tj/svm_light/svm_rank.html, accessed on November 2014.

[96] A. Joshi, G. Dalvi, M. Joshi, P. Rashinkar, and A. Sarangdhar, "Design and Evaluation of Devanagari Virtual Keyboards for Touch Screen Mobile Phones," in *Proceedings of MobileHCI*. Stockholm, Sweden: ACM, 2011, pp. 323–332.

[97] Y. Jung, D. Joshi, V. Narayanan-Saroja, and D. P. Desai, "Solving the Great Indian Text Input Puzzle: Touch Screen-based Mobile Text Input Design," in *Proceedings of MobileHCI*. Stockholm, Sweden: ACM, 2011, pp. 313–322.

[98] D. Jurafsky and J. Martin, *Speech and Language Processing*. Prentice Hall, 2000.

[99] S. M. Katz, "Estimation of Probabilities from Sparse Data for the Language Model Component of a Speech Recognizer," in *IEEE Transactions on Acoustics, Speech and Signal Processing*, 1987, pp. 400–401.

[100] D. E. Kieras, *The Human-Computer Interaction Handbook*, 2nd ed. Lawrence Erlbaum Associates, Mahwah, New Jersey, 2002, ch. Model-based Evaluation.

[101] D. Kieras, "Using the Keystroke-Level Model to Estimate Execution Times," ftp://www.eecs.umich.edu/people/kieras/GOMS/KLM.pdf, 1993, accessed on September, 2014.

[102] K. Kin, B. Hartmann, T. DeRose, and M. Agrawala, "Proton++: A Customizable Declarative Multitouch Framework," in *Proceedings of the Symposium on User Interface Software and Technology (UIST)*. Cambridge, USA: ACM, 2012, pp. 477–486.

[103] K. Kin, B. Hartmann, T. DeRose, and M. Agrawala, "Proton: Multitouch Gestures as Regular Expressions," in *Proceedings of the Conference on Human Factors in Computing Systems*. Texas, USA: ACM, 2012, pp. 2885–2894.

[104] A. L. Koerich, R. Sabgurin, and C. Y. Suen, "Large Vocabulary Off-line Handwriting Recognition: A Survey," *Pattern Analysis and Applications*, vol. 6, no. 2, pp. 97–121, 2003.

[105] H. H. Koester and S. P. Levine, "Model Simulations of User Performance with Word Prediction," *Augmentative and Alternative Communication*, vol. 14, no. 1, pp. 25–35, 1998.

[106] A. Komninos and M. Dunlop, "Using a smart-watch as an Input Device for Text," iEEE Pervasive, in press 2014.

[107] P. O. Kristensson, "Five Challenges for Intelligent Text Entry Methods," *AI Magazine*, vol. 30, no. 4, pp. 85–94, 2009.

[108] P. O. Kristensson, T. Nicholson, and A. Quigley, "Continuous Recognition of One-handed and Two-handed Gestures Using 3D Full-body Motion Tracking Sensors," in *Proceedings of the International Conference on Intelligent User Interfaces (IUI)*. Lisbon, Portugal: ACM, 2012, pp. 89–92.

[109] P. O. Kristensson and K. Vertanen, "The Potential of Dwell-free Eye-typing for Fast Assistive Gaze Communication," in *Proceedings of the Symposium on Eye Tracking Research and Applications (ETRA)*. California, USA: ACM, 2012, pp. 241–244.

[110] P. O. Kristensson. and S. Zhai, "SHARK^2: A Large Vocabulary Shorthand Writing System for Pen-based Computers," in *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST)*. ACM, 2004, pp. 43–52.

[111] P. O. Kristensson and S. Zhai, "Relaxing Stylus Typing Precision by Geometric Pattern Matching," in *Proceedings of the International Conference on Intelligent User Interfaces (IUI)*. San Diego, USA: ACM, 2005, pp. 151–158.

[112] P. O. K. Kristensson and K. Vertanen, "The Inviscid Text Entry Rate and Its Application as a Grand Goal for Mobile Text Entry, booktitle = Proceedings of the International Conference on Human-computer Interaction with Mobile Devices and Services (MobileHCI), year = 2014, pages = 335–338, address = Toronto, Canada, organization = ACM."

[113] P. O. Kristensson, "Discrete and Continuous Shape Writing for Text Entry and Control," Ph.D. dissertation, Department of Computer and Information Science, Linköping University, 2007.

[114] P. Kristensson and L. Denby, "Text Entry Performance of State of the Art Unconstrained Handwriting Recognition: a Longitudinal User Study," in *Proceedings of the Conference on Human Factors in Computing Systems*. Boston, USA: ACM, 2009, pp. 567–570.

[115] A. Kumar, H. Shah, and A. Joshi, "Saral: Devanagari Text input system in Mobile phones," in *Proceedings of UbiComp*. Florida, USA: ACM, 2009.

[116] G. P. Kurtenbach, A. J. Sellen, and W. A. Buxton, "An Empirical Evaluation of Some Articulatory and Cognitive Aspects of Marking Menus," *Human-Computer Interaction*, vol. 8, no. 1, pp. 1–23, 1993.

[117] S. E. Levinson, "Structural Methods in Automatic Speech Recognition," *Proceedings of the IEEE*, vol. 73, no. 11, pp. 1625–1650, 1985.

[118] D. Levy, "The fastap keypad and pervasive computing," in *Pervasive Computing*. Springer, 2002, pp. 58–68.

[119] J. R. Lewis, "Typing-key Layouts for Single-finger or Stylus Input: Initial User Preference and Performance," International Business Machines Corporation, Boca Raton, FL, USA, Tech. Rep. 54729.

[120] J. R. Lewis, P. J. Kennedy, and M. J. LaLomia, "Development of a Digram-Based Typing Key Layout for Single-Finger Stylus Input," in *Proceedings of the Human Factors and Ergonomics Society 43rd Annual Meeting*, Houston, Texas, USA, 1999.

[121] J. R. Lewis, K. Potosnak, and R. Magyar, *Handbook of Human-computer Interaction*, 2nd ed. Amsterdam: Elsevier Science, 1997, ch. Keys and Keyboards, pp. 1285–1315.

[122] Y. Li, "Protractor: A Fast and Accurate Gesture Recognizer," in *Proceedings of Conference on Human Factors in Computing Systems*. Georgia, USA: ACM, 2010, pp. 2169–2172.

[123] LingPipe, "Character lm tutorial," available: http://alias-i.com/lingpipe/demos/tutorial/lm/read-me.html, accessed on January 2014.

[124] Lipik, "A Predictive Text Input System," http://www.lipik.in, 2007, accessed on October 2014.

[125] LooKeys, "Indian Languages Virtual Keyboard," http://www.keyboard4all .com, 2009, accessed on October 2014.

[126] I. S. MacKenzie, "Mobile Text Entry using Three Keys," in *Proceedings of NordiCHI*. Aarhus, Denmark: ACM, 2002, pp. 27–34.

[127] I. S. MacKenzie and R. W. Soukoreff, "Text entry for mobile computing: Models and methods, theory and practice," *Human–Computer Interaction*, vol. 17, no. 2-3, pp. 147–198, 2002.

[128] I. S. MacKenzie and K. Tanaka-Ishii, *Text Entry Systems: Mobility, Accessibility, Universality.* MA, USA: Morgan Kaufmann Inc., 2007.

[129] I. S. MacKenzie and S. X. Zhang, "The Design and Evaluation of a High Performance Soft Keyboard," in *Proceedings of the Conference on Human Factors in Computing Systems*, Pittsburgh, Pennsylvania, USA, 1999.

[130] M. Magazine, "Ericsson introduces the new r380e," 2011, available: http://www.mobilemag.com/2001/09/25/ericsson-introduces-the-new-r380e, Accessed on November 2014.

[131] P. Majaranta, "Text Entry by Eye Gaze," Ph.D. dissertation, University of Tampere, 2009.

[132] P. Majaranta, U. K. Ahola, and O. Špakov, "Fast Gaze Typing with an Adjustable Dwell Time," in *Proceedings of the Conference on Human Factors in Computing Systems.* Boston, USA: ACM, 2009, pp. 357–360.

[133] P. Majaranta, A. Aula, and K. J. Räihä, "Effects of Feedback on Eye Typing with a Short Dwell Time," in *Proceedings of the Symposium on Eye Tracking Research & Applications (ETRA).* Texas, USA: ACM, 2004, pp. 139–146.

[134] P. Majaranta and K. J. Räihä, *Text Entry Systems: Mobility, accessibility, universality.* San Francisco, CA: Eds. Morgan Kaufmann, 2007, ch. Text Entry by Gaze: Utilizing Eye-tracking, pp. 175–187.

[135] J. Mankoff and G. D. Abowd, "Cirrin: A Word-level Unistroke Keyboard for Pen Input," in *Proceedings of the 11th annual ACM symposium on User Interface Software and Technology (UIST).* San Francisco, CA, USA: ACM, 1998, pp. 213–214.

[136] C. D. Manning and H. Schutze, *Foundations of Statistical Natural Language Processing.* MIT Press, June 1999.

[137] T. Masui, "POBox: An Efficient Text Input Method for Handheld and Ubiquitous Computers," in *Handheld and Ubiquitous Computing.* Springer, 1999, pp. 289–300.

[138] E. Matias, I. S. MacKenzie, and W. Buxton, "Half-QWERTY: A One-handed Keyboard Facilitating Skill Transfer from QWERTY," in *Proceedings of the Conference on Human Factors in Computing Systems*. Amsterdam, The Netherlands: ACM, 1993, pp. 88–94.

[139] E. Matias, I. S. MacKenzie, and W. Buxton, "One-handed Touch-typing on a Qwerty Keyboard," *Human-Computer Interaction*, vol. 11, no. 1, pp. 1–27, 1996.

[140] J. Matiasek, M. Baroni, and H. Trost, "FASTY - A Multi-Lingual Approach to Text Prediction," in *Proceedings of International Conference on Computers Helping People with Special Needs*, ser. Lecture Notes in Computer Science, K. Miesenberger, J. Klaus, and W. Zagler, Eds., vol. 2398. Austria: Springer, July 2002, pp. 243–250.

[141] MayabiSoft, "Mayabi soft keyboard," available: http://mayabi-bangla-keyboard.soft112.com, accessed on October 2012.

[142] E. Mays, F. J. Damerau, and R. L. Mercer, "Context Based Spelling Correction," *Information Processing and Management*, vol. 25, no. 2, pp. 517–522, 1991.

[143] M. S. Mayzner and M. E. Tresselt, "Tables of Single-letter and Digram Frequency Counts for Various Word-length and Letter-position Combinations," *Psychonomic Monograph Supplements*, vol. 1, no. 2, pp. 13–32, 1965.

[144] Microsoft, "Onscreen Keyboard," 2009, available: http://windows.microsoft.com/en/windows7/Type-without-using-the-keyboard-On-Screen-Keyboard, accessed on October 2014.

[145] D. Miniotas, O. Spakov, and G. Evreinov, "Symbol Creator: An Alternative Eye-based Text Entry Technique with Low Demand for Screen Space," in *Proceedings of INTERACT*, Zurich, Switzerland, 2003, pp. 137–143.

[146] C. Moor, "T-Mobile G1 Event Round-up," 2008, available: http://www.talkandroid.com/260-t-mobile-g1-details, accessed on November 2014.

[147] C. H. Morimoto and A. Amir, "Context Switching for Fast Key Selection in Text Entry Applications," in *Proceedings of the Symposium of Eye Tracking Research and Applications (ETRA)*. Texas, USA: ACM, 2010, pp. 271–274.

[148] A. Mukherjee, S. Bhattacharya, P. Haider, and A. Basu, "A Virtual Predictive Keyboard as a Learning Aid for People with Neuro-Motor Disorders," in

Proceedings of IEEE International Conference on Advanced Learning Technologies, (ICALT). Kaohsiung, Taiwan: IEEE, july 2005, pp. 1032–1036.

[149] P. Mukherji and P. P. Rege, "Shape Feature and Fuzzy Logic-based Offline Devnagari Handwritten Optical Character Recognition," *Journal of Pattern Recognition Research*, vol. 25, no. 4, pp. 52–68, 2009.

[150] MyLanguage, "Hindi Virtual Keyboard," http://www.mylanguages.org/hindi_keyboard.php, 2008, accessed on October 2014.

[151] A. Nafiz and F. T. Yarman-Vural, "An Overview of Character Recognition Focused on Off-line Handwriting," *IEEE Transactions on Systems, Man, and Cybernetics-Part C*, vol. 31, no. 2, pp. 216–233, 2001.

[152] NCIP, "Writing with Word Prediction Software," 1994, available: http://www.edc.org/NCIP/LIBRARY/wp/Profile.htm, accessed on July 2014.

[153] S. Nesbat, "A System for Fast, Full-text Entry for Small Electronic Devices," in *Proceedings of the International Conference on Multimodal Interaction (ICMI).* Vancouver, Canada.: ACM, 2003, pp. 4–11.

[154] C. Neti, G. Potamianos, J. Luettin, I. Matthews, H. Glotin, D. Vergyri, J. Sison, A. Mashari, and J. Zhou, "Audio-visual Speech Recognition," in *Final Workshop Report*, vol. 764, 2000.

[155] NextBigWhat, "Smartphone Insights: Indian Users Now Spend More Time On Content Than Voice and SMS," 2014, available: http://www.nextbigwhat.com/smartphone-market-statistics-for-india-297, accessed on July 2014.

[156] T. Ni, D. Bowman, and C. North, "AirStroke: Bringing Unistroke Text Entry to Freehand Gesture Interfaces," in *Proceedings of the Conference on Human Factors in Computing Systems.* Vancuver, Canada: ACM, 2011, pp. 2473–2476.

[157] Nuance, "Swype," available: http://www.swype.com/, accessed on October 2014.

[158] M. Obitko, "Selection-Introduction to Genetic Algorithms," http://www.obitko.com/ tutorials/genetic-algorithms/selection.php, 1998, accessed on June 2014.

[159] OmicronLab, "Avro Bengali Keyboard," http://www.omicronlab.com/avro-keyboard.html, 2010, accessed on October 2014.

[160] P. Panwar, S. Sarcar, and D. Samanta, "EyeBoard: A Fast and Accurate Eye Gaze-based Text Entry System," in *Proceedings of International Conference on Human-Computer Interaction (IHCI)*. Kharagpur, India: IEEE, 2012, pp. 1–8.

[161] J. Pedler, "Computer Correction of Real Word Spelling Errors in Dyslexic Text," Ph.D. dissertation, PhD thesis. Birkbeck, University of London, 2007.

[162] K. Perlin, "Quikwriting: Continuous Stylus-based Text Entry," in *Proceedings of Symposium on User Interface Software and Technology (UIST)*. San Francisco, USA: ACM, 1998, pp. 215–216.

[163] R. Plamondon and S. N. Srihari, "On-line and Off-line Handwriting Recognition: A Comprehensive Survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 1, pp. 63–84, 2000.

[164] K. J. Räihä and S. Ovaska, "An Eexploratory Study of Eye Typing Fundamentals: Dwell Time, Text Entry Rate, Errors, and Wworkload," in *Proceedings of the Conference on Human Factors in Computing Systems*. ACM, 2012, pp. 3001–3010.

[165] F. Rose, "Pocket Monster: How DoCoMo's Wireless Internet Service Went from Fad to Phenom - and Turned Japan into the First Post-PC Nation." 2004, available: http://archive.wired.com/wired/archive/9.09/docomo_pr.html, accessed on November 2014.

[166] D. Rubine., "Specifying Gestures by Example." in *Proceedings of SIGGRAPH*. Las Vegas, USA: ACM, 1991, pp. 329–337.

[167] D. Rudchenko, T. Paek, and E. Badger, "Text Text Revolution: a Game that Improves Text Entry on Mobile Touchscreen Keyboards," in *Pervasive Computing*. Springer, 2011, pp. 206–213.

[168] D. Samanta, S. Sarcar, and S. Ghosh, "An Approach to Design Virtual Keyboards for Text Composition in Indian Languages," *International Journal Human Computer Interaction*, vol. 29, no. 8, pp. 516–540, 2013.

[169] S. Sarcar, P. Panwar, and T. Chakraborty, "EyeK: An Efficient Dwell-free Eye Gaze-based Text Entry System," in *Proceedings of the Asia Pacific Conference on Computer Human Interaction*. New York, NY, USA: ACM, 2013, pp. 215–220.

[170] A. Schick, D. Morlock, C. Amma, T. Schultz, and R. Stiefelhagen, "Vision-based Handwriting Recognition for Unrestricted Text Input in Mid-air," in *Proceedings of the International Conference on Multimodal Interaction (ICMI)*. Santa Monica, USA: ACM, 2012, pp. 217–220.

[171] K. Seymore and R. Rosenfeld, "Scalable Backoff Language Models," in *Proceedings of Fourth International Conference on Spoken Language*, vol. 1. IEEE, 1996, pp. 232–235.

[172] S. Shanbhag, D. Rao, and R. K. Joshi, "An Intelligent Multi-layered Input Scheme for Phonetic Scripts," in *Proceedings of the* 2^{nd} *International Symposium on Smart Graphics*. ACM, 2002, pp. 35–38.

[173] M. K. Sharma, "Word Prediction System with Virtual Keyboard for Text Entry in Hindi," M.S. thesis, School of Information Technology, Indian Institute of Technology Kharagpur, 2012.

[174] M. K. Sharma and D. Samanta, "Word Prediction System for Text Entry in Hindi," *ACM Transactions on Asian Language Information Processing*, vol. 13, no. 2, pp. 8:1–8:29, 2014.

[175] K. K. Shieh and C. C. Lin, "A Quantitative Model for Designing Keyboard Layout," *Percept Motor Skills*, vol. 88, no. 1, pp. 113–125, 1999.

[176] Shilpa, "Swathanthra Indian Language Computing Project," available: http://smc.org.in/silpa/Soundex, Accessed on March 2012.

[177] B. Shneiderman and C. Plaisant, *Designing the User Interface: Strategies for Effective Human Computer Interaction*, 4th ed. Addison Wesley, 2004.

[178] G. L. Sholes, C. Glidden, and S. W. Soule, "Improvement in Type-writing Machines," U.S. Patent Patent 79,868, 1868.

[179] R. M. K. Sinha, "A journey from indian scripts processing to indian language processing," *IEEE Annals of the History of Computing*, vol. 31, no. 1, pp. 8–31, 2009.

[180] H. Skovsgaard, J. C. Mateo, and J. P. Hansen, "Evaluating Gaze-based Interface Tools to Facilitate Point-and-selcet Tasks with Small Targets," *Behaviour & Information Technology*, vol. 30, no. 6, pp. 821–831, 2011.

[181] SNLTR, "Society for Natural Language Technology Research," http://www.nltr.org/, 2009, accessed on December, 2010.

[182] R. W. Soukoreff and I. S. MacKenzie, "Theoretical Upper and Lower Bounds on Typing Speed Using a Stylus and Soft Keyboard," *Behaviour and Information Technology*, vol. 14, pp. 370–379, 1995.

[183] R. W. Soukoreff and I. S. MacKenzie, "A Character-level Error Analysis Technique for Evaluating Text Entry Methods," in *Proceedings of the Second Nordic Conference on Human-Computer Interaction (NordiCHI)*, New York, 2002, pp. 241–244.

[184] R. W. Soukoreff and I. S. MacKenzie, "Metrics for Text Entry Research: An Evaluation of MSD and KSPC, and a New Unified Error Metric," in *Proceedings of the ACM Conference on Human Factors in Computing Systems*, New York, 2003, pp. 113–120.

[185] R. W. Soukoreff and I. S. MacKenzie, "Recent Developments in Text Entry Error Rate Measurements," in *Extended Abstracts of the Conference on Human Factors in Computing Systems*. Vienna, Austria: ACM, 2004, pp. 1425–1428.

[186] O. Špakov and P. Majaranta, "Scrollable Keyboards for Eye Typing," in *Proceedings of COGAIN*, Prague, Czech Republic, 2008, pp. 63–66.

[187] O. Špakov and D. Miniotas, "On-line Adjustment of Dwell Time for Target Selection by Gaze," in *Proceedings of the Nordic Conference on Human-computer Interaction (NordiCHI)*. ACM, 2004, pp. 203–206.

[188] I. L. Stats, "Internet users," available: http://www.internetlivestats.com/internet-users/, accessed on July 2014.

[189] S. Strassel, M. Maxwell, and C. Cieri, "Linguistic Resource Creation for Research and Technology Development: A Recent Experiment," *ACM Transsctions on Asian Language Information Processing*, vol. 2, no. 2, pp. 101–117, 2003.

[190] M. Sugimoto and K. Takahashi, "SHK Single Hand Key Card for Mobile Devices," in *Conference on Human Factors in Computing Systems*. Vancouvar, Canada: ACM, 1996, pp. 7–8.

[191] SwiftKey, "Swiftkey," 2008, available: http://swiftkey.com/en/, accessed on October 2014.

[192] Z.-H. Tan and B. Lindberg, "Speech recognition on mobile devices," in *Mobile Multimedia Processing*. Springer, 2010, pp. 221–237.

[193] C. C. Tappert, C. Y. Suen, and T. Wakahara, "The State of the Art in On-line Handwriting Recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 12, no. 8, pp. 787–808, 1990.

[194] TDIL, "Inscript Keyboard," available: http://tdil.mit.gov.in/keyoverlay.htm, accessed on October 2014.

[195] Technology Development for Indian Languages, "Keyboard Standard," http://www.tdil.mit.gov.in/pdf/KeyboardStandard2.pdf, 1986, accessed on June 2014.

[196] Technology Development for Indian Languages, "iLeap - Multilingual Word Processor," http://www.cdac.in/html/ gist/products/ileap.asp, 2009, accessed on October, 2010.

[197] S. Thottingal, "Soundex codes for Indic languages," available: http://thottingal.in/soundex/soundex.html, accessed on March 2014.

[198] TRAI, "Indian Telecom Services Performance Indicator Report January - March 2014," available: http://www.trai.gov.in/WriteReadData/PIRReport/Documents/IndicatorReports-Mar-14.pdf, accessed on July 2014.

[199] K. Trnka, J. McCaw, D. Yarrington, K. F. McCoy, and C. Pennington, "User Interaction with Word Prediction: The Effects of Prediction Quality," *ACM Transaction on Accessible Computing*, vol. 1, no. 3, pp. 1–34, 2009.

[200] K. Trnka and K. McCoy, "Evaluating Word Prediction: Framing Keystroke Savings," in *Proceedings of ACL-08: HLT, Short Papers*. Columbus, Ohio: Association for Computational Linguistics, June 2008, pp. 261–264.

[201] K. Trnka, D. Yarrington, J. McCaw, K. F. McCoy, and C. Pennington, "The Effects of Word Prediction on Communication Rate for AAC," in *Proceedings of Human Language Technologies 2007*. Association for Computational Linguistics, 2007, pp. 173–176.

[202] Twiddler, "The Chording Keyboard," available: http://twiddler.tekgear.com/, accessed on November 2014.

[203] T.-M. UK, "T-Mobile G1 Hits the UK," 2008, available: www.opt-development.co.uk/press-office/release.php?id=242, accessed on November 2014.

[204] Unicode Consortium, "Unicode Bengali Code Chart," http://www.unicode.org/charts/PDF/U0980.pdf, 2008, accessed on December 2014.

[205] Unicode Consortium, "Unicode Hindi Code Chart," http://www.unicode.org/charts/PDF/U0900.pdf, 2008, accessed on December 2014.

[206] Unicode Consortium, "Unicode Telugu Code Chart," http://www.unicode.org/charts/PDF/U0C00.pdf, 2008, accessed on December 2014.

[207] M. H. Urbina, "Gaze Controlled Applications and Optical-See-Through Displays - General Conditions for Gaze Driven Companion Technologies," Ph.D. dissertation, Department of Computer Science, 2012.

[208] M. H. Urbina and A. Huckauf, "Dwell Time Free Eye Typing Approaches," in *Proceedings of the 3rd Conference on Communication by Gaze Interaction*, 2007, pp. 3–4.

[209] M. H. Urbina, M. Lorenz, and A. Huckauf, "Pies with Eyes: the Limits of Hierarchical Pie Menus in Gaze Control," in *Proceedings of the Symposium on Eye-Tracking Research and Applications (ETRA)*. Texas, USA: ACM, 2010, pp. 93–96.

[210] G. C. Vanderheiden and D. P. Kelso, "Comparative Analysis of Fixed-vocabulary Communication Acceleration Techniques," *Augmentative and Alternative Communication*, vol. 3, no. 4, pp. 196–206, 1987.

[211] R. D. Vatavu, L. Anthony, and J. Wobbrock, "Gestures as Point Clouds: A $P Recognizer for User Interface Prototypes," in *Proceedings of the International Conference on Multimodal Interaction (ICMI)*. Santa Monica, USA: ACM, 2012, pp. 273–280.

[212] K. Vertanen and P. Kristensson, "Parakeet: A Continuous Speech Recognition System for Mobile Touch-screen Devices," in *Proceedings of the 14th International Conference on Intelligent User Interfaces (IUI)*. Florida, USA: ACM, 2009, pp. 237–246.

[213] Vistawide, "Top 30 Language Spoken in theWorld by Number of Speakers," 2009, available: http://www.vistawide.com/languages/top_30_languages.htm, acessed on November 2014.

[214] T. Wandmacher, "Adaptive Word Prediction and its Application in an Assistive Communication System," Ph.D. dissertation, University of Tubingan, 2009.

[215] T. Wandmacher, J. Antoine, and F. Poirier, "SIBYLLE: A System for Alternative Communication Adapting to the Context and its User," in *Proceedings of the ACM SIGACCESS conference on Computers and Accessibility (ASSETS)*. Arizona, USA: ACM, 2007, pp. 203–210.

[216] F. Wang and X. Ren, "Empirical evaluation for finger input properties in multi-touch interaction," in *Proceedings of the Conference on Human Factors in Computing Systems*. ACM, 2009, pp. 1063–1072.

[217] D. J. Ward, A. F. Blackwell, and D. J. MacKay, "Dasher - A Data Entry Interface using Continuous Gestures and Language Models," in *Proceedings of the Symposium on User Interface Software and Technology (UIST)*. ACM, 2000, pp. 129–137.

[218] Wikipedia, "Comparison of High-definition Smartphone Displays," available: http://en.wikipedia.org/wiki/Comparison_of_high-definition_smartphone_displays, accessed on October 2014.

[219] Wikipedia, "Demographics of india," available: en.wikipedia.org/wiki/Demographics_of_India, accessed on July 2014.

[220] Wikipedia, "Hindispeaking and writing," available: http://en.wikibooks.org/wiki/Hindi/Speaking_and_Writing, accessed on July 2014.

[221] Wikipedia, "Smartphone," available: http://en.wikipedia.org/wiki/Smartphone, accessed on November 2014.

[222] Wikipedia, "Devnagari," http://en.wikipedia.org/wiki/Devanagari, 2008, accessed on July 2014.

[223] Wikipedia, "Languages of India," http://en.wikipedia.org/wiki/ Languages_of_India, 2008, accessed on July 2014.

[224] Wikipedia, "Arithmetic coding," 2014, available: http://en.wikipedia.org/wiki/Arithmetic_coding, accessed on July 2014.

[225] J. O. Wobbrock and B. Myers, "Trackball Text Entry for People with Motor Impairments," in *Proceedings of the Conference on Human Factors in Computing Systems*. Montreal, Canada: ACM, 2006, pp. 479–488.

[226] J. O. Wobbrock and B. A. Myers, "Analyzing the input stream for character-level errors in unconstrained text entry evaluations," *ACM Transactions on Computer-Human Interaction (TOCHI)*, vol. 13, no. 4, pp. 458–489, 2006.

[227] J. O. Wobbrock, B. A. Myers, and J. A. Kembel, "EdgeWrite: A Stylus-based Text Entry Method Designed for High-accuracy and Stability of Motion." in *Proceedings of the Symposium on User Interface Software and Technology (UIST)*. Vancuver, Canada: ACM, 2003, pp. 61–70.

[228] J. O. Wobbrock, J. Rubinstein, M. W. Sawyer, and A. T. Duchowski, "Longitudinal Evaluation of Discrete Consecutive Gaze Gestures for Text Entry," in *Proceedings of the Symposium on Eye Tracking Research and applications (ETRA)*. GA, USA: ACM, 2008, pp. 11–18.

[229] J. O. Wobbrock, A. D. Wilson, and Y. Li, "Gestures without Libraries, Toolkits or Training: A $1 Recognizer for User Interface Prototypes," in *Proceedings of the Symposium on User Interface Software and Technology (UIST)*. Newport, USA: ACM, 2007, pp. 159–168.

[230] E. Wolf, S. Vembu, and T. Miller, "On the Use of Topic Models for Word Completion," *Advances in Natural Language Processing*, vol. 4139, pp. 500–511, 2006.

[231] M. Wood, "Syntactic Pre Processing in Single Word Prediction for Disabled People," Ph.D. dissertation, University of Bristol, 1996.

[232] A. H. Wright, "Genetic Algorithms for Real Parameter Optimization," in *Foundations of Genetic Algorithms*, G. J. E. Rawlins, Ed. Morgan Kaufmann Inc., 1991, pp. 205–218.

[233] H. YAMADA, "A Historical Study of Typewriters and Typing Methods: from the Position of Planning Japanese Parallels," *Journal of Information Processing*, vol. 4, no. 2, pp. 175–202., 1980.

[234] H. Yu and S. Kim, "SVM Tutorial: Classification, Regression, and Ranking," http://hwanjoyu.org/publication/svmtutorial.pdf, 2009, accessed on 17th February, 2012.

[235] B. L. Z.-H. Tan, "Automatic Speech Recognition on Mobile Devices and Over Communication Networks," *Advances in Computer Vision and Pattern Recognition*, 2008.

[236] S. Zhai, M. Hunter, and B. A. Smith, "Performance Optimization of Virtual Keyboards," *Human Computer Interaction*, vol. 17, no. 2, pp. 229–269, 2002.

[237] S. Zhai and P. O. Kristensson, "Interlaced QWERTY: Accommodating Ease of Visual Search and Input Flexibility in Shape Writing," in *Proceedings of the Conference on Human Factors in Computing Systems*. Florance, Italy: ACM, 2008, pp. 593–596.

[238] S. Zhai, M. Hunter, and B. A. Smith, "The Metropolis Keyboard - an Exploration of Quantitative Techniques for Virtual Keyboard Design," in *Proceedings of the annual ACM symposium on User Interface Software and Technology (UIST)*. San Diego, CA, USA: ACM, 2000, pp. 119–128.

[239] X. Zheng, S. Goose, J. Kiekebosch, and J. W. Lin, "AVIN (Assisted Visual Interactive Notepad): A Novel Interface Design to Expedite the Eye Writing Experience," in *Proceedings of HCII*, Florida, USA, 2011, pp. 635–644.

[240] S. Zhou, Z. Dong, W. J. Li, and C. P. Kwong, "Hand-Written Character Recognition Using MEMS Motion Sensing Technology," in *Proceedings of IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM)*. Xi'an, China: IEEE, 2008, pp. 1418–1423.

Publications out of this work

Published

- Debasis Samanta, **Sayan Sarcar** and Soumalya Ghosh, "An Approach to Design Virtual Keyboards for Text Composition in Indian Languages", *International Journal of Human-Computer Interaction (IJHCI)*, Vol-29, No-8, pp. 516-540, 2013.
- Tuhin Chakraborty, **Sayan Sarcar** and Debasis Samanta, "Design and Evaluation of a Dwell-free Eye Typing Technique", *Proceedings of CHI'14 Extended Abstracts on Human Factors in Computing Systems*, ACM, pp.1573-1578, 2014, Toronto, Canada.
- **Sayan Sarcar**, Prateek Panwar and Tuhin Chakraborty, "EyeK: An Efficient Dwell-Free Eye Gaze-Based Text Entry System", *Proceedings of Asia Pacific Conference on Computer Human Interaction (APCHI)*, ACM, pp.215-220, 2013, Bangalore, India.
- **Sayan Sarcar** and Prateek Panwar, "Eyeboard++: an enhanced eye gaze-based text entry system in Hindi", *Proceedings of International Conference on Human Computer Interaction (India HCI)*, ACM, pp.354-363, 2013, Bangalore, India.
- Prateek Panwar, **Sayan Sarcar** and Debasis Samanta, "EyeBoard: A Fast and Accurate Eye Gaze-based Text Entry System", *Proceedings of International Conference on Human Computer Interaction (IHCI)*, IEEE Xplorer, pp.1-8, 2012, Kharagpur, India.
- **Sayan Sarcar**, Soumalya Ghosh, Pradipta Kumar Saha and Debasis Samanta, "Virtual Keyboard Design: State of the Arts and Research Issues", *Proceedings of Students' Technology Symposium (TechSym)*, IEEE Xplorer, pp.289-299, 2010, Kharagpur, India.

Communicated

- Manoj Kumar Sharma, Debasis Samanta, and **Sayan Sarcar**, "A Machine Learning Approach to Word Prediction for Text Entry in Indic Languages", *ACM Transactions on Computer Human Interaction (TOCHI)*. (Paper id: TOCHI-2013-0084, Submitted on 30 September, 2014)