The Effects of Google Smart Compose on The Process of Open-Ended Writing

Sijan Shrestha
University of Mississippi

THE EFFECTS OF GOOGLE SMART COMPOSE ON THE PROCESS OF OPEN-ENDED WRITING

THESIS

A Thesis
presented in partial fulfillment of requirements
for the degree of Master of Science
in the Department of Computer and Information Science
The University of Mississippi

by

SIJAN SHRESTHA

Aug 2023

ABSTRACT

Recently, a lot of new natural language models have been developed which help individuals with writing. One of the widely used tools is Google's Smart Compose which is a text-predictive system that helps individuals to write by reducing the need for repetitive typing. Though this tool has been well established and used by a lot of people, there is a lack of studies on its impact on open-ended writing.

In this thesis, we investigate the effect of Google's Smart Compose in open-ended writing. To do this, we built a custom software that collects data while the users are writing on Google Docs web application. We recruited 119 individuals who wrote on Google Docs where 55 of the individuals had smart compose enabled while the other 59 had the smart compose disabled. We then compared the writings of those two groups. Additionally, we also compared the participants' writing process under different suggestions.

Our results show that Google's Smart Compose does not have a significant quantitative or qualitative effect on open-ended writing. However, one positive impact of Google's Smart Compose is that it reduced the time required to write a character by 35.3 milliseconds. This was measured by comparing the time it took to write 10 characters before and 10 characters after the suggestion appeared.

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my advisor, Dr. Thai Le, for his continuous support and guidance throughout my Master's thesis. His expertise, time, and suggestions were invaluable, and I am grateful for his mentorship.

I would also like to thank my committee members, Dr. Feng Wang and Dr. Timothy Holston, for their expertise, time, and suggestions. Their feedback helped me to improve my thesis, and I am grateful for their insights.

I would like to thank Dr. Robert Cummings for providing me with the opportunity to work on this project and for his help during the experiments. His guidance was essential to the success of my research.

I would like to thank the Department of Computer and Information Science and the Institute of Child Nutrition for making my Master's journey so smooth. Their academic and financial support was essential to my success.

Finally, I would like to thank my family and friends for their love and support. Without their encouragement, I would not have been able to achieve my goals.

TABLE OF CONTENTS

ABSTRACT ii

ACKNOWLEDGEMENTS iii

LIST OF TABLES v

LIST OF FIGURES vi

1 INTRODUCTION 1

2 RELATED WORK 8

3 RESEARCH QUESTIONS 10

4 METHODOLOGY 11

5 RESULTS 23

6 LIMITATIONS 29

7 CONCLUSIONS 30

BIBLIOGRAPHY 31

VITA 34

LIST OF TABLES

4.1 Participant Genders 15
4.2 Participant Ages 15
4.3 Participant Race and Ethnicities 16
4.4 Participant Education 16
4.5 Participant Auto-complete Technologies Used 16
4.6 Participant Use of Auto complete Technology 17
4.7 Participant Likeness of Auto complete Technology 17

5.1 Comparison of word, sentence, paragraph, and words per sentence counts . . 23
5.2 Comparison of two groups' keystroke-to-character ratios 23
5.3 Comparison of Writings in Terms of Qualitative Values from Coh-Metrix . . 24
5.4 Average time (milliseconds) to type a character before and after suggestions 27
5.5 Difference in time (in ms) to type a character before and after suggestion appears for different suggestions 28

LIST OF FIGURES

4.1 Software Workflow Diagram 13
5.1 Distribution of the Ratio of the number of keystrokes and the number of total characters 24
5.2 Box Plot comparing Latent Semantic (a) and Lexical Diversity (b) 25
5.3 Box Plot comparing Readability (a) and Referential Cohesion (b) 25
5.4 Box Plot comparing Syntactic Complexity (a) and Syntactic Simplicity (b) . 26
5.5 Comparison of typing speed standard deviation using box plots 26
5.6 Box plot of time (in ms) taken to write a single character under different suggestions 28

1 INTRODUCTION

The rise in Natural Language Processing (NLP) technology has changed the way we write and communicate. NLP-powered tools have made writing more efficient and convenient, and one of the most popular NLP tools is Google's Smart Compose. Smart Compose is an assistive tool that helps writers type faster by reducing the need for repetitive typing. It is an extension of traditional predictive text and autocorrect systems, but it goes further by predicting entire phrases and sentences based on the writer's context.

1.1 Auto-correction

Auto-correction (also called AutoCorrect) is a technology that automatically corrects misspelled words. The work to develop this technology started as early as 1960 [12]. There are three main parts involved in autocorrecting words: Detection of misspelled words, selection of candidates for replacing the misspelled words, and correction of those misspelled words [10].

There are two main methods of detecting misspellings[12]:

- Dictionary check: The word is checked against a dictionary to see if it exists.
- N-gram check: The last n words are checked to see if they form a valid sequence. If the sequence does not exist in the database, the word is marked as possibly misspelled.

The hard part of autocorrect is selecting candidates to replace the incorrect words. Many methods have been proposed to solve this problem and mostly used ones are:

1.1.1 Minimum edit distance technique

In this technique, the word from the database which can be reached from the misspelled word with a minimum number of edits is replaced by the misspelled word [12]. There are two main edit distances: Levenshtein and Damerau–Levenshtein. Levenshtein distance measures the number of edit operations (insertions, deletions, and substitutions) required to transform one string into another. Damerau–Levenshtein distance also includes transpositions (swapping two adjacent characters) [10]. Many optimized variants of the algorithm are available which makes the correction response time quicker. In [14], the authors used dynamic programming, and in [13], the authors store the same word multiple times where the duplicate word has one letter missing. Since the most common type of misspelled word is missing a letter, they developed a hash function to check the correct word for the misspelled word [12].

1.1.2 Similarity Key Technique

In this technique, the strings are mapped to the keys where similarly spelled words have identical or similar keys. The words which have the same keys as the misspelled word are chosen as the candidates.

1.1.3 Rule-based Technique

This is one of the techniques for correcting the misspelled word by applying a set of rules. The rules are based on the patterns of common misspellings and the frequency of occurrence of words in the language. In [16], the authors analyzed 1377 spelling errors made by adults and developed rules to correct the misspellings based on the collected errors. They used empirical rules based on observations such as: The most common misspellings have one character missing, people often hit the adjacent key instead of the correct key, etc. The rules are applied to the misspelled word to identify the most likely correct spelling.

1.1.4 N-gram based Technique

In this technique, the similarity between the two strings is calculated by counting the number of n-grams the two strings have in common [2]. Here, an n-gram is a sequence of n-letters of the string. To correct a misspelled word, the n-gram based technique first divides the misspelled word into n-grams. Then, it compares the n-grams of the misspelled word to the n-grams of all the words in the dictionary. The words in the dictionary with the most matching n-grams with the misspelled word are chosen as the candidates for replacing the misspelled word.

1.1.5 Neural Nets Technique

Neural nets techniques are more recent and advanced than traditional spelling correction techniques. In this technique, the input of the neural net is the misspelled word and the output is the correct version of the misspelled word. The neural nets are first trained on the dataset of misspelled words and their corresponding correct words. The earlier neural networks were just simple neural networks consisting of an input layer, an output layer, and a few hidden layers. But the recent ones are more advanced neural network models like Convolutional Neural Networks (CNN) and versions of Recurrent Neural Networks (RNN). The neural networks basically have two structures: the encoder and the decoder [18]. The encoder encodes the input text into the feature vector and the decode part decodes the feature vector to the output text. Some of the new neural network models [15] change the problem of spelling correction to the language translation problem where the sentence with the misspelled word is translated to the sentence with correctly spelled words.

1.2 Predictive Text Systems

Predictive text suggestion systems are the extension of traditional auto-completion and text replacement systems that help writers type faster by suggesting words or phrases that are likely to be typed next [6]. The predictive text system suggests phrases while the user is writing and the suggestions are based on the user's current text. The purpose of the predictive systems is to reduce the effort in typing and the time it takes to write the message. There are multiple predictive text suggestion systems that are based on suggesting the next character or syllables or words or sentences. The predictive text systems are used in a variety of applications like message typing on smartphones, writing emails, etc. Two different kinds of models are being used in these systems[9]:

1.2.1 Statistical Models

The statistical models use probability to predict the next word in the sequence. These models need a large corpus of text data to get the probability of a word occurring in a given context. For example, from the text corpus, the model may learn that 'am' is more likely to come after 'I' than the word 'is'. Statistical models are very powerful for text prediction and can be divided into three categories:

1.2.1.1 Word Frequency

This approach depends on a text corpus. In this approach, the frequencies of words in the given corpus are saved as a list and the list is sorted from highest to lowest. The words are suggested based on the highest frequency. These methods are easy to implement but won't suggest the correct words most of the time as they will suggest only a few words that are on the top of the sorted list. The suggestions can be made more personalized by updating the list as the user types. So next time the words that the users write more are suggested more often.

1.2.1.2 N-gram models

[17] In this approach, the next word is predicted based on the previous n words in the sequence. To train these models, a large corpus of text is needed. The text is broken

down into sequences of n words, called n-grams. For example, if the text corpus is divided into two-word sequences, then it is called a bi-gram. The frequency of each n-gram sequence is saved. Next time the writer writes, the last n words written by the writer are checked. If the sequence exists, the most common word after the sequence is suggested.

1.2.1.3 Markov Models

Markov models are a statistical method that can be used to predict the next word in a sequence. They work by considering the probability of the next word occurring, given the previous words in the sequence[9]. In the context of text prediction, each unique word in the text corpus is considered a state. A transition matrix is then created, which includes the probabilities of words that come after each state [1]. For example, if the current state is 'the', the transition matrix might indicate the next words that may come after the word 'the' and their probabilities. To predict the next word, the Markov model simply looks up the current state in the transition matrix and returns the word with the highest probability.

1.2.2 Neural Language Models

The Neural Language Models are the most recent ones and work better than the previous ones. They are the advanced form of statistical models. But these models are more complex and computationally expensive. In text prediction systems, there are mainly variants of two types of neural language models used and they are Convolutional Neural Networks (CNN) and variants of Recurrent Neural Networks (RNN) models. The performance of these models depends on the quality and quantity of the text data. A large amount of text is used to train the model and are trained to predict the next word in the sequence. The models are trained to learn the statistical relationships between the words and given a few words written by the writer, it can suggest the next word or phrases. In [5], the authors built and compared two types of language models: Transformers and Long Short-Term Memory (LSTM) (a type of RNN). They found that for similar-sized models, the Transformer model performed slightly better than the LSTM model in terms of accuracy, but was worse in terms of latency.

1.3 Google's Smart Compose

Google's Smart Compose(GSC) [5] is a system developed by Google to help users write faster by reducing the need for repetitive typing. It offers real-time interactive suggestions, which can help users write quickly and eliminate the need to type the same thing over and over again. Smart Compose was initially developed for Gmail, but it is now also available in Google Docs. It is a predictive text system that is designed to provide personalized suggestions. To do this, Smart Compose blends personal and global models.

The global model uses an LSTM language model, while the personal model uses an n-gram model with Katz-backoff. The subject of the current email and the previous email go through the context encoder in the global model. The output of the context encoder and the current email prefix are then fed into the language model, which predicts the next word. The personal model is trained for each user, so it needs to be small, adaptive, and easy to train periodically. Because of this, Google chose to use an n-gram model with Katz-backoff[11] as the personal model. If the number of n-grams in the text dataset is lower than a certain threshold, the Katz-backoff version of n-gram will back off to lower-order (n-1) grams until the desired threshold is found. N-gram models are easier to train and require less data than other types of language models. In a single Smart Compose request, the context is first encoded. This includes the previous message body, the current email subject, and the current prefix. The encoded contexts are then fed to the language model, which generates a representation of the complete encoded prefix. This representation is then used in the beam search to find possible suggestions. The final prediction is a weighted average of the predictions from the global model and the personal model computed during the beam search. It is calculated as

$$Pfinal = aPpersonal + (1 - a)Pglobal \tag{1.3.1}$$

Here, $Pfinal$ is the final output, $Ppersonal$ is the output from the personal model, $Pglobal$ is the output from the global model, and a is constant and controls how personalized the suggestions are.

2 RELATED WORK

The majority of studies on the effects of predictive text systems have been conducted using short texts. There hasn't been much work done to evaluate the effectiveness of these tools in open-ended writing. In [7], they let the participants type the 100 words sample email with Smart Compose On and Off. Their results didn't show a significant difference between the email written with smart compose on and off. In their study, they only studied the effect of smart composing while writing short emails and they evaluated the performance of the writers based on how many words were correctly typed. In this thesis, we are evaluating the effect of Google's Smart Compose on open-ended writing. We are also going to do the quantitative and qualitative analysis between the writings of the users with the smart compose on and off. Additionally, we are going to study the effect of different kinds of suggestions on the typing of the users.

In [3], the authors study the effects of predictive features of Mobile Keyboards on Text Entry Speed and errors and found out that their method of text prediction saves the need to type 3.43 characters per phrase but it added an extra 2 seconds time to type the word. In this study, the authors only studied the effect of predictive text systems while messaging using Smart Phones. In comparison, this thesis studies the effects of Google's Smart Compose in open-ended writing and on a desktop computer. Though the system we are analyzing is very similar to the system in the paper, the method of writing and the writing interface are very different.

In a study [4], the authors used a single-layer LSTM model with a hidden state of dimension 2048 was used as the text prediction system. The model was trained on image captions. Participants were recruited to write captions for images, and the study found that captions written with the predictive system ON were shorter and had fewer unpredictable words. Additionally, the predictive system was found to have a positive effect on typing speed. However, the study was limited in that participants only wrote 1 to few sentences and wrote on mobile phones. In contrast, our study allowed users to write for 25 minutes on laptops.

3 RESEARCH QUESTIONS

To get better results, it is always important to ask the better question and be sure of what needs to be answered. This section describes the questions that we want to answer through this thesis. Through this thesis, we are trying to find answers to the following research questions:

i. Are there differences in the writings of users with Google's Smart Compose On and Off? Since the main objective of this research is to find out if Google's Smart Compose has any impact on users' writing, we would like to find out if there are quantitative and qualitative differences between the writings of the users with the GSC on and off. We would like to check if the writers with Google Smart Compose can write longer texts and have more text quality.

ii. Do the suggestions on the screen affect the user's writing process? Google's Smart Compose suggestions have the potential to reduce the need to come up with own words or phrases and help users write faster and help users be more attentive to their writing. However, it can also distract the users by suggesting irrelevant phrases. So, in this thesis, it is important to answer the question of whether the suggestions have any effect on the writing process.

iii. Do the different types of suggestions have different effects on the writing process? The Smart Compose suggestions are sometimes accepted by the users and sometimes they are rejected or partially accepted. We would like to check if the favorability of the suggestions provided affects the writing process of the users.

4 METHODOLOGY

This paper focuses on the effect of Google's Smart Compose on open-ended writing. But there is a lack of a public dataset that can be used for analyzing the effect of Google Smart Compose on open-ended writing. There is no public API available that can be used to get the related data from Google Docs, so a custom software needs to be developed which allows users to write on Google Docs and simultaneously collect the necessary data. The collected data then needs to be analyzed to see if there are any effects of Google Smart Compose on the users' writings. This section is divided into Software Development, Data Collection, Data Analysis, and the Result.

4.1 Software Development

We needed data to study the effect of Google Search Console (GSC) on users' writing. However, there is no public API available for this task. Therefore, we built custom software that allowed users to write in Google Docs while collecting data about their writing. The custom user interface was built using Google Chrome Driver. The interface looks exactly like the Google Chrome browser, but it allows us to interact with web pages programmatically and automate tasks such as form filling and button clicking.

The software is multi-threaded where we used three threads to achieve our data collection goal. The first thread is responsible for collecting keystrokes data and final completed text and storing them in online and local databases. The second one collects, processes, and stores the Google Docs data in the online MongoDB database and in the local drive, and the third one keeps track of time and alerts users when the time approaches the 25 minutes time limit and when the time limit has passed. The software was built using the Python and JavaScript programming languages. The Selenium library with Chrome Web driver was used to automatically navigate to Google Docs and sign in users. JavaScript was used to collect the completed user writing at the end of the session and to alert the user in the browser about the time limit using a popup box.

We used SeleniumWire, an extension of the Selenium library, to build a proxy server between the Google Docs Web application and the GSC server. This server can access the requests made by the browser as well as the response received by the browser from the real server. The main purpose of this proxy server is to collect the data related to Google's Smart Compose. To prevent the slow performance because of the proxy server, we allowed the proxy server to capture the data only related to Google's Smart Compose.

The request and the response body were processed to get the time, current text, suggest prompt, and suggest result. The processed data is then sent to the secure online MongoDB database and is saved to the local device as well. We used the Pynput library to listen to the keyboard strokes to collect the keyboard strokes and time of it. At the end of the writing session, the software runs the JavaScript code to get texts written by the participants which are saved in the online MongoDB database as well in the local device.

Our experiment was conducted in person, and we had a limited number of laptops available for participants. To make the experiment run faster, smoother and to reduce the wait time for participants, we automated some of the tasks using our software. The software would automatically:

- Login to the assigned Google account
- Create the file and Set the file name
- Send the collected data to an online MongoDB server and save it to the local drive as well
- Alert the users when the time limit was approaching or reached

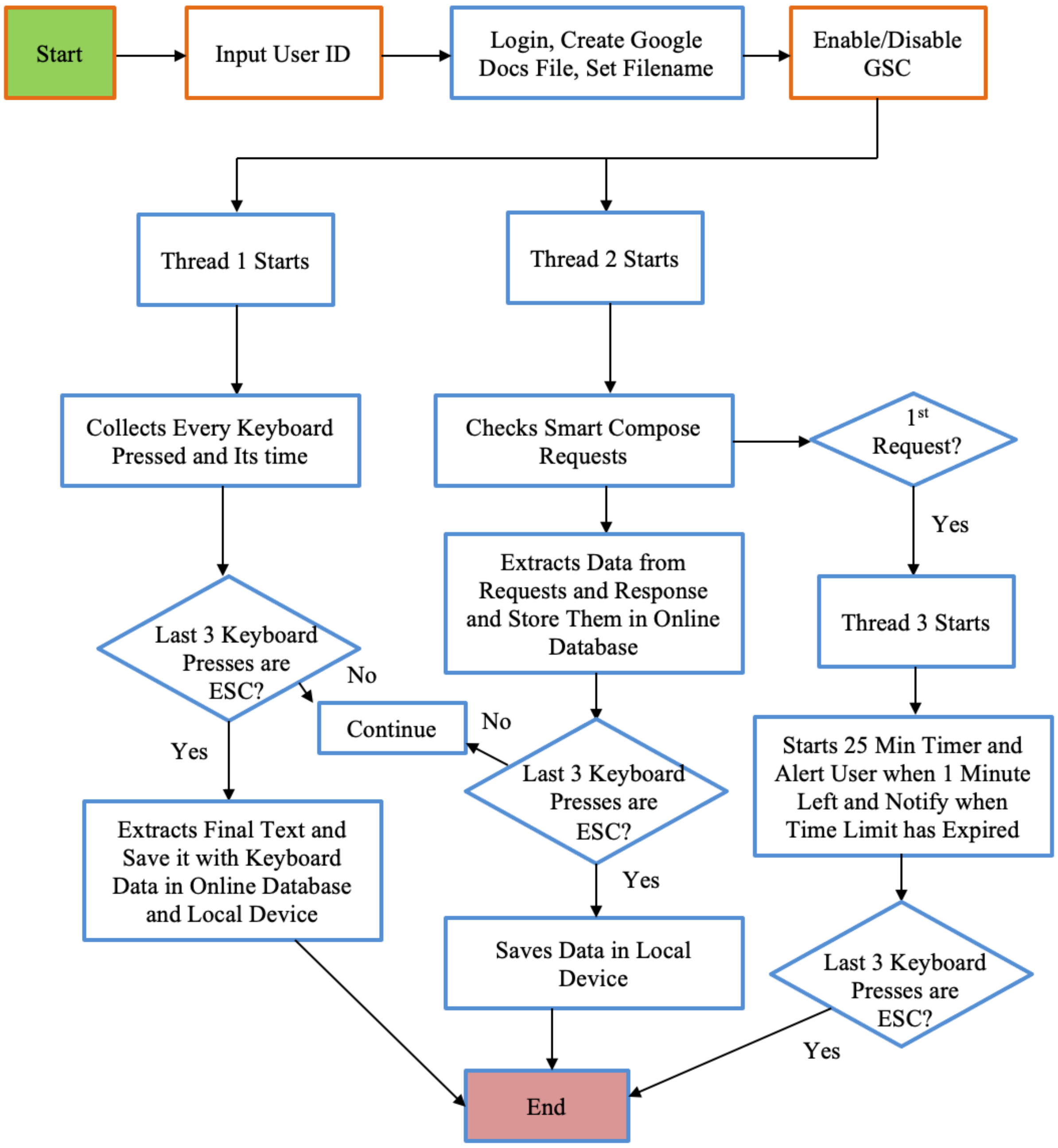

Start
Input User ID
Login, Create Google Docs File, Set Filename
Enable/Disable GSC
Thread 1 Starts
Thread 2 Starts
Collects Every Keyboard Pressed and Its time
Checks Smart Compose Requests
1st Request?
Yes
Extracts Data from Requests and Response and Store Them in Online Database
Thread 3 Starts
Last 3 Keyboard Presses are ESC?
No
Continue
No
Yes
Last 3 Keyboard Presses are ESC?
Starts 25 Min Timer and Alert User when 1 Minute Left and Notify when Time Limit has Expired
Yes
Extracts Final Text and Save it with Keyboard Data in Online Database and Local Device
Saves Data in Local Device
Last 3 Keyboard Presses are ESC?
Yes
End

Figure 4.1: Software Workflow Diagram

Our main code file is a Python file that runs in the background while the user writes on Google Docs. Once we run the Python file, it asks for a Google account number assigned to the participants. It then automatically opens the Google Chrome browser, logs in to the

Google account, creates a Google Doc file, and names the file as the assigned Google number. The timer starts after the participant starts writing.

In Google Docs, Smart Compose is a feature that uses machine learning to suggest words and phrases as you type. When Smart Compose is turned on, the web application sends a request to the server for every character written, to check for any possible suggestions. If any suggestions are shown on the screen, then they are extracted from the response body sent by the server. The software processes the data and saves the data in the database. The software workflow diagram is shown in Figure 4.1. Here, the tasks inside the box with an Orange color border are done manually and the ones with the blue color are done by software.

4.2 Data Collection

In this section, we are going to discuss how, what, and why we collected different data required for studying the effects of Google Smart Compose on open-ended writing. This section is divided into three sub-sections: Experimental Settings, Participants, and Collected Data.

4.2.1 Experimental Settings

We conducted our experiment in the Student Union of the University of Mississippi. We booked a room where four MacBook laptops were available, and participants were assigned to the laptop randomly based on availability and first come first serve order.

4.2.2 Participants

We recruited 119 participants from various backgrounds to write for 25 minutes on a given question. The participants were randomly divided into two groups: an experimental group with access to Google's Smart Compose, and a control group with no access to Google's Smart Compose.

During the study, we asked the participants the following demographic questions:

- Gender (Table 4.1)
- Age (Table 4.2)
- Racial and ethnic identities (Table 4.3)

We also asked the participants about their:

- Highest level of education completed (Table 4.4)
- Applications where they have used autocomplete technology (Table 4.5)
- Frequency of autocomplete technology usage (Table 4.6)
- Opinion of autocomplete technology (Table 4.7)

Below are the data about the participants:

Table 4.1: Participant Genders

Gender	**Number**	**Percentage**
Male	70	58.9%
Female	47	39.4%
Non-binary	2	1.7%
No Response	0	0%
Total	**119**	**100%**

Table 4.2: Participant Ages

Age Group	**Number**	**Percentage**
18-22	69	58%
23-29	34	29%
30-39	12	10%
40-49	4	3%
50+	0	0%
Total	**119**	**100%**

Table 4.3: Participant Race and Ethnicities

Race and Ethnicity	**Number**	**Percentage**
Caucasian	58	49%
African American	14	12%
Latino or Hispanic	3	3%
Asian	34	29%
Native American	0	0%
Native Hawaiian	0	0%
Two or more	6	5%
Other or Unknown	3	3%
No Response	1	1%
Total	**119**	**100%**

Table 4.4: Participant Education

Highest Level of Education Completed	**Number**	**Percentage**
Some High School	0	0%
High School	12	10%
Freshman	18	15%
Sophomore	18	15%
Junior	19	16%
Bachelor	29	24%
Master	20	17%
PhD	3	3%
No Response	0	0%
Total	**119**	**100%**

Table 4.5: Participant Auto-complete Technologies Used

Auto-complete Technology	**Number**	**Percentage**
Word Processor	89	75%
Texting Application	80	67%
Web-based Application	72	61%
Other	4	3%
No Response	2	2%

Table 4.6: Participant Use of Auto complete Technology

Frequency	**Number**	**Percentage**
Very Frequently	39	33%
Frequently	30	25%
Sometimes	30	25%
Rarely	16	13%
Never	1	1%
No Response	3	3%
Total	**119**	**100%**

Table 4.7: Participant Likeness of Auto complete Technology

Likeness	**Number**	**Percentage**
Very Favorable	44	37%
Favorable	43	36%
Neutral	23	19%
Negative	3	3%
Very Negative	0	0%
No Response	6	5%
Total	**119**	**100%**

4.2.3 Collected Data

During the experiment, we collected the following data:

i. Keyboard Data

Every keyboard press made by the participants was recorded using the Pynput library in Python. This data included each button pressed and the time it was pressed in chronological order.

ii. Google Docs Data

During a writing session in Google Docs, the web application sends a request to the server every time the user types something. The request includes the text that the user has typed so far. If Smart Compose is enabled, the server uses a language model to check if any suggestions can be made. If there is a suggestion, the server responds with an array that includes the suggested text and the part of the text that the user has typed that was used to make the suggestion. The web application then displays the suggestion to the user. The user can accept or reject the suggestion. The custom software we are using is programmed to extract the following rows of data for each request and response:

Time of request: The time at which the request was made.

Text written by user: The text that the user had written by the time the request was made.

Suggestions prompt: The text that was used by the server to make relevant suggestions.

Suggestion result: The text suggested by the server.

Type: It indicates whether a suggestion was made by the server, or user is typing the suggested letters, or the user is typing regularly

iii. Final Text

After the writing is done, the final text written by the user was collected separately. For this, a JavaScript code was run automatically on the console of the web application and the final text was collected and saved as the '.txt' files.

iv. Screen Recording

We used the Zoom application to record the participants' screens while they were writing on Google Docs. The screens were recorded as a backup and the recorded videos were automatically uploaded to the Zoom account which were later downloaded.

Of the 119 participants in our study, we were able to collect keyboard data and Google Docs data from 116 participants, and final text from 114 participants.

4.3 Data Analysis

This section describes the analysis we did with the collected data and the motivation behind those analyses. In this thesis, we would like to analyze if the Smart Compose has any effect on open-ended writings. Additionally, we would like to test how different kinds of suggestions affect the writing style.

Our main focus of this thesis is to see if the writings with Smart Compose On are different compared to the writings with Google's Smart Compose Off. For this, the final texts of the two groups were used for comparison. To answer our first research question, we would like to do quantitative and qualitative analyses of the two groups of writings. To perform these analyses, I am using Coh-metrix[8] 3.0 software. This software is a well-established tool used for analyzing the text and for any given text, it gives 108 values that provide us with various information ranging from basic information like the number of words to qualitative values like textual cohesion, reading difficulty, etc. Additionally, I will also analyze the number of keystrokes required to get the number of characters in the final text.

In quantitative analysis, we would like to see if there is a significant difference in the length of text between the two groups. Here, we are comparing the number of words, number of paragraphs, number of sentences, and number of words per sentence to compare the length of text between two groups. Following are the values used from Coh-metrix to get these values:

- **DESWC:** number of words
- **DESSC:** number of sentences
- **DESPC:** number of paragraphs
- **DESSL:** number of words per sentence

Additionally, Google Smart Compose is a feature that is supposed to make writing easier by reducing the need for repetitive typing. To see if Smart Compose has the same effect on open-ended writing, we compared the number of keystrokes participants pressed to the total number of characters in their final writings.

The number of keystrokes each participant pressed was extracted from the Keyboard data, and the total number of characters in their final writings was retrieved from the final text. We then divided the number of keystrokes by the number of characters to get the average number of keystrokes per character. We then calculated the mean, median, and standard deviation of the ratios for two groups of users. This allowed us to see how the two groups differed in terms of the number of key presses required.

In qualitative analysis, we would like to compare how easier it is to read and process the writings with Google's Smart Compose ON and OFF. For this, we used the following variables from the Coh-Metrix:

- **PCSYNz:** Syntactic simplicity measures the degree to which a text has sentences with fewer words and uses a simpler, familiar syntactic structure. This makes the text easier to process, as there are fewer words to process. A higher score indicates more syntactic simplicity.

- **PCREFz:** Referential cohesion measures the degree to which a text contains words and ideas that overlap across sentences. This makes the text easier to process, as the reader can more easily understand the relationships between the different sentences. A higher score indicates more referential cohesion.

- **LSASS1:** Latent semantic analysis (LSA) measures how conceptually similar one sentence is to the next sentence. A better text has higher semantic overlap between adjacent sentences. This means that the sentences are more closely related to each other, which makes the text easier to understand.

- **LDTTRc:** Lexical diversity measures the number of unique content words divided by the total number of content words in a text. A lower lexical diversity means that there are more repeated words in the text. The text with lower lexical diversity might be easier to read as the readers don't have to process a lot of different words while reading.

- **SYNLE:** Syntactic complexity measures the number of words before the main verb of the main clause in sentences. A higher score indicates more syntactic complexity. This means that the sentences are more complex and may be more difficult to read.

- **RDFRE:** Readability measures how easy it is to read a text. A higher score indicates easier reading. The value is calculated as

$$RDFRE = 206.835 - (1.015 \times \text{ASL}) - (84.6 \times \text{ASW}) \tag{4.3.1}$$

  Here, ASL is average sentence length and ASW is average number of syllables per word

Google Smart Compose is a feature that suggests words and phrases as you type. We hypothesized that the suggestions would help writers focus and write more smoothly, leading to shorter time gaps between consecutive keyboard strokes. To test this hypothesis, we collected data on the time gaps between consecutive keyboard strokes for writers with Smart Compose on and off. We removed outliers from the data and calculated the standard deviation for each user and compared the standard deviations between two groups.

Secondly, we want to know if different kinds of suggestions have any effect on the users' writing within the experimental group. For this, we divided the suggestions into three categories:

1. Full Acceptance: If all the suggested phrases are accepted
2. Partial Acceptance: If at least one of the words of the suggested phrases is accepted
3. Rejection: If none of the suggested words are accepted

The suggestions are accepted if the users find the suggested phrases to be a perfect fit in the given sentences. If the suggested phrases are acceptable to the writers, then it would free the writers from needing to come up with their words and allow them to focus more on the message of the writing. However, it can also distract the writers by suggesting irrelevant phrases. We want to analyze if the favorability of the suggested phrases have any effect on the writing process of the participants. We then analyzed the GSC-enabled participants' writing behavior during these three conditions. To see if the suggestions have any effect on the writing process of the writers, we performed two analyses:

1. The typing speed of users before and after the GSC suggestion

We wanted to investigate the immediate effects of the appearance of suggestions on the screen on the typing speed of participants. We also wanted to determine whether suggestions that participants preferred had a different effect on typing speed than suggestions that participants rejected. We compared typing speed under different numbers of keystrokes before and after the suggestion appeared to assess the immediate impact. The following numbers of keystrokes before and after the suggestion appeared were used to compare the immediate impact of GSC suggestions on users' writing: 5, 10, and 20. This analysis will provide us with information about the typing speed of participants before and after the suggestion appears as well as under different responses (Full Acceptance, partial Acceptance, and Rejection). These would help answer our research questions 2 and 3.

2. The time it took to accept or reject the suggested phrases

We wanted to compare the time it took to accept or reject suggested phrases. This would give us information about how quickly users could complete writing under various suggestions. These would help answer our research question number 3. We hypothesized that accepting suggested phrases would take less time because users could simply click the Tab or Right keys to accept them.

To do this, we calculated the total time it took to write the number of suggested words. In the case of full acceptance, the time taken to write all suggested words was considered. For partial acceptance and rejection, the time taken to write the same number of words as the number of suggested words was taken into account.

Since the length of suggestions can vary, we wanted to compare the average time it took to write a character under different suggestions. The average time to write a character was calculated as the total time taken divided by the total number of written characters. When counting the total number of characters, the characters from the text prompts were not counted. Only the characters that were written after the suggestion appeared were considered.

5 RESULTS

5.1 Analysis Between Writings with Smart Compose On and Off

Table 5.1: Comparison of word, sentence, paragraph, and words per sentence counts

	Smart Compose On					**Smart Compose Off**				
	Min	**Max**	**Mean**	**Med.**	**Std**	**Min**	**Max**	**Mean**	**Med.**	**Std**
Words	241	1245	585.5	562	205.1	303	1096	595.56	579	190.4
Sentences	14	77	29	27	10.9	11	60	29.66	28	11.14
Paragraphs	1	10	3.9	3	2.47	1	14	4.39	4	2.91
Words per Sentence	12.62	42	21.1	19.78	6.01	13.2	32.27	21.00	20	4.51

Table 5.1 compares the number of words, sentences, paragraphs, and words per sentence in writings with Smart Compose enabled and disabled. On average, writings with Smart Compose enabled have 10 fewer words. However, a t-test found that the difference is not statistically significant, with a p-value of 0.685. This means that there is no evidence to suggest that Smart Compose has a significant impact on the number of words written.

Table 5.2: Comparison of two groups' keystroke-to-character ratios

	Mean	**Median**	**Standard Deviation**
GSC On	1.17	1.16	0.09
GSC Off	1.20	1.19	0.10

Figure 5.1 shows that there is no clear difference in the distribution of the ratio of keystrokes to characters between the groups with Smart Compose on and off. The average ratio difference is 0.03, which is not statistically significant (p-value = 0.144). This means that there is no evidence to suggest that Smart Compose reduces the number of keystrokes needed to produce a piece of text in open-ended writing.

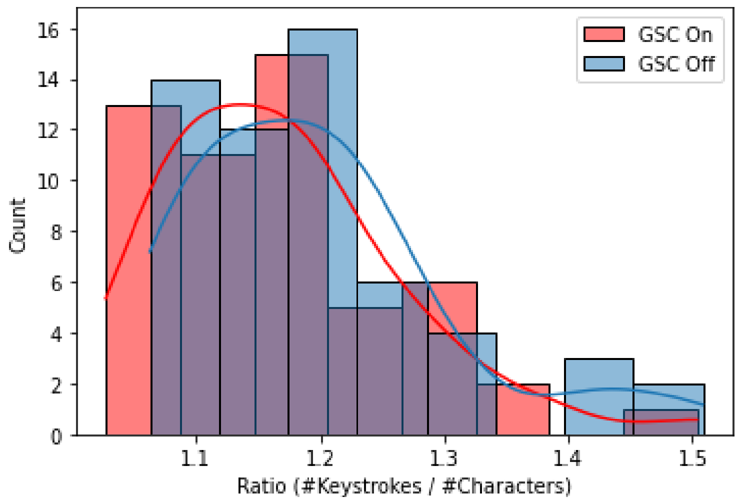
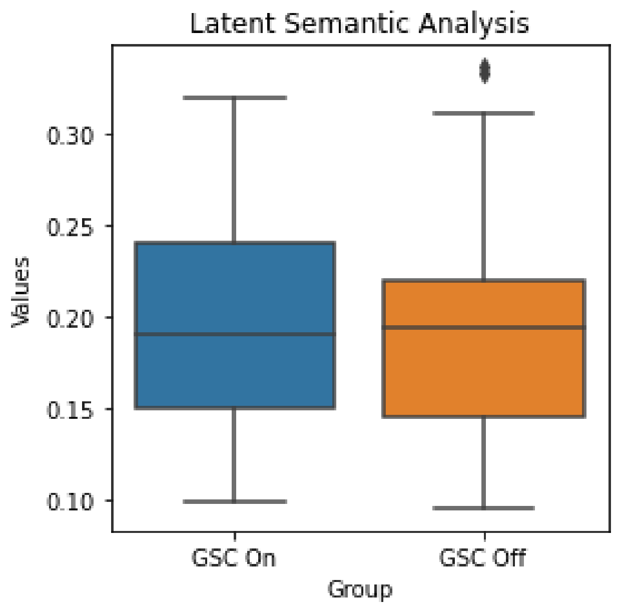


Figure 5.1: Distribution of the Ratio of the number of keystrokes and the number of total characters

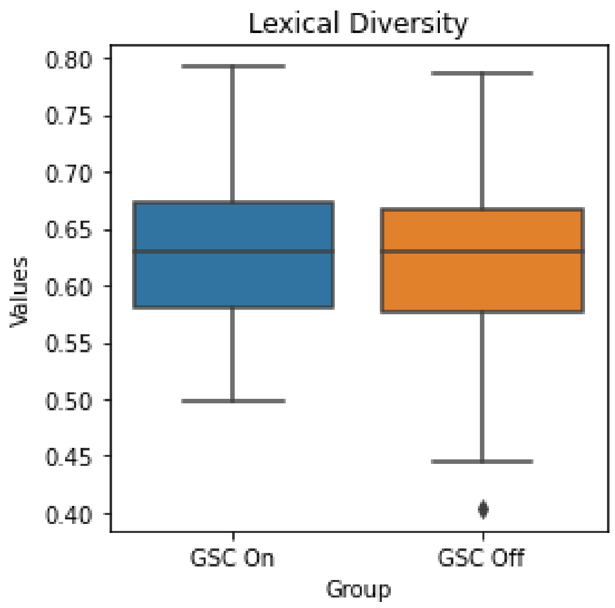
Table 5.3: Comparison of Writings in Terms of Qualitative Values from Coh-Metrix

	Smart Compose On					**Smart Compose Off**				
	Min	**Max**	**Mean**	**Med.**	**Std**	**Min**	**Max**	**Mean**	**Med.**	**Std**
PCSYNz	-2.02	1.12	-0.32	-0.24	0.62	-1.57	1.06	-0.30	-0.22	0.60
PCREFz	-1.10	2.14	0.36	0.45	0.75	-1.06	2.19	0.41	0.34	0.71
LSASS1	0.10	0.32	0.19	0.19	0.05	0.10	0.34	0.19	0.19	0.06
LDTTRc	0.50	0.80	0.63	0.63	0.07	0.40	0.79	0.62	0.63	0.10
SYNLE	2.25	7.60	4.16	3.83	1.38	2.10	9.00	4.39	4.28	1.31
RDFRE	37.72	80.70	65.62	66.92	9.10	39.94	82.14	67.02	68.19	8.09

The cells highlighted in green in Table 5.3 indicate better values in the respective categories on average. The table shows that writings with Google's Smart Compose On performed better in terms of syntactic complexity (SYNLE), the same in terms of latent semantic analysis, but worse in terms of referential cohesion, lexical diversity, syntactic simplicity, and readability. However, the t-test of these groups found that these differences are not statistically significant, as the p-values from t-test for syntactic simplicity (PCSYNz), referential cohesion (PCREFz), latent semantic analysis (LSASS1), lexical diversity (LDTTRc), syntactic complexity (SYNLE), and readability (RDFRE) are 0.85, 0.73, 0.88, 0.91, 0.25, and 0.39, respectively. Figures 5.2 to 5.4 show the differences in box plots.

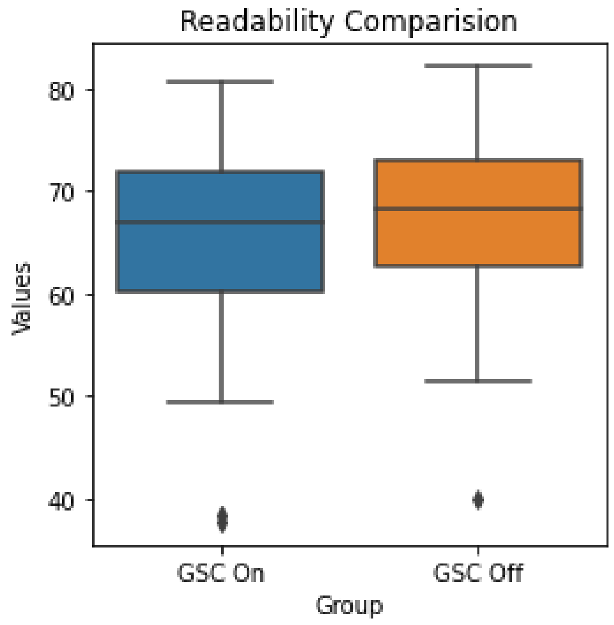


(a) Latent Semantic Analysis

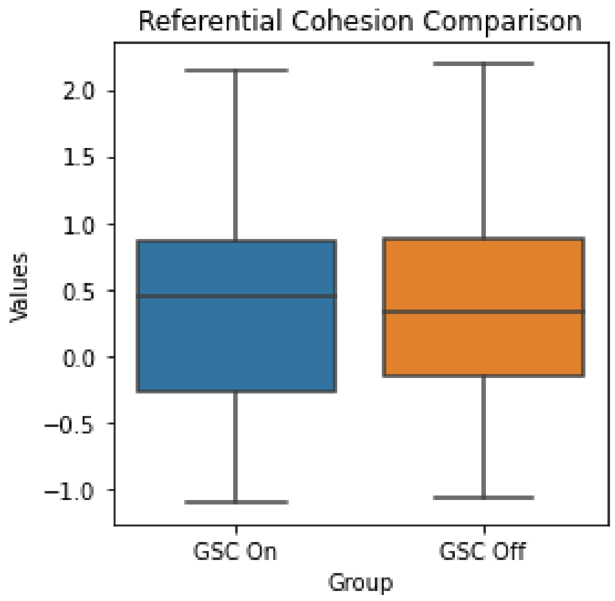


(b) Lexical Diversity

Figure 5.2: Box Plot comparing Latent Semantic (a) and Lexical Diversity (b)



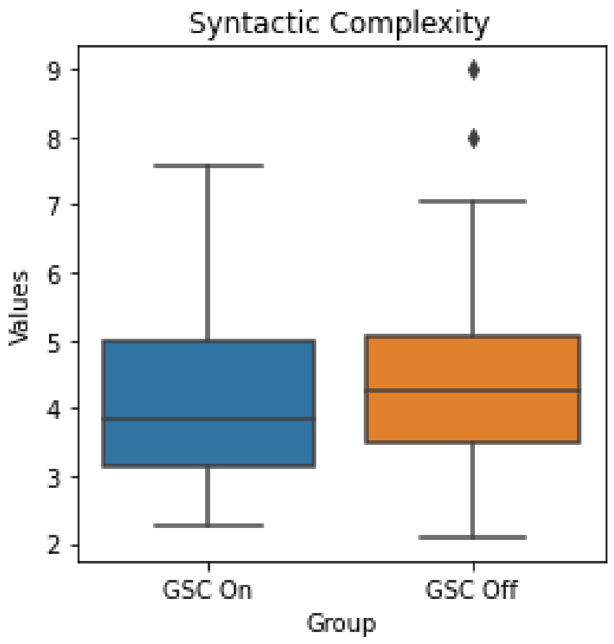
(a) Readability



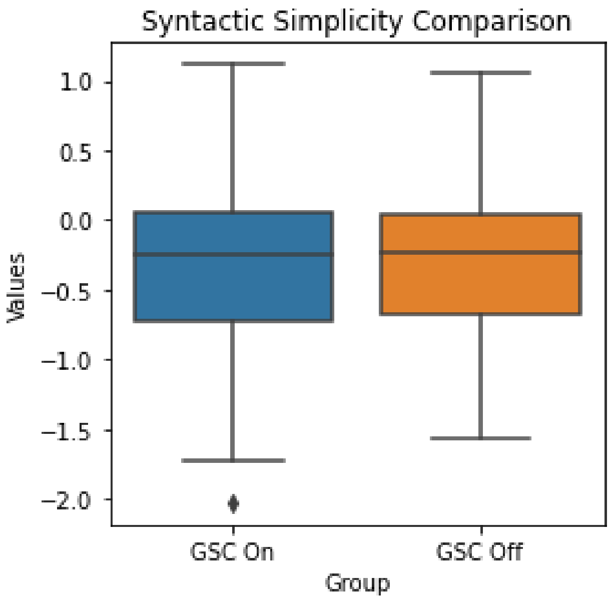
(b) Referential Cohesion

Figure 5.3: Box Plot comparing Readability (a) and Referential Cohesion (b)

Figure 5.5 shows a box plot comparing the standard deviation of typing speed between users with and without Smart Compose enabled. The figure shows that there is no significant



(a) Syntactic Complexity



(b) Syntactic Simplicity

Figure 5.4: Box Plot comparing Syntactic Complexity (a) and Syntactic Simplicity (b)

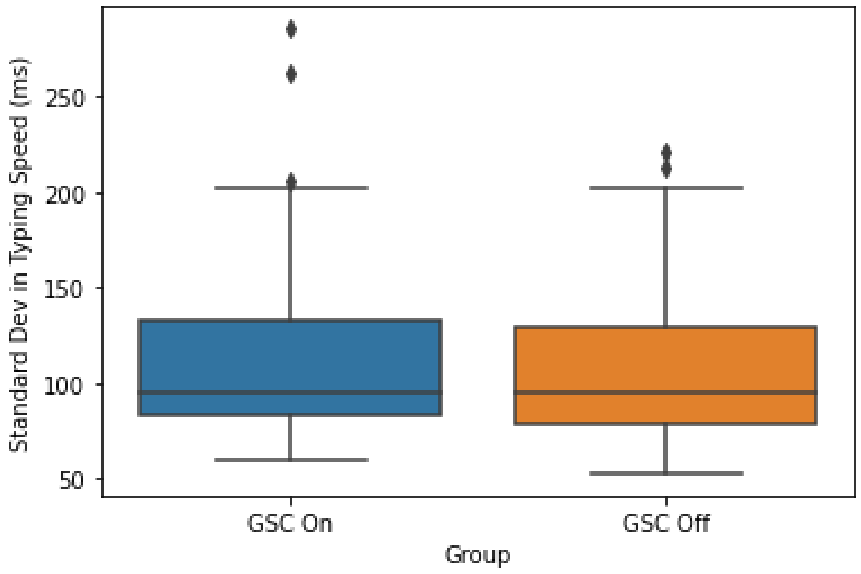


Figure 5.5: Comparison of typing speed standard deviation using box plots

difference between the two groups, meaning that the writings with Smart Compose enabled are not less erratic than the writings with Smart Compose disabled.

5.2 Effect of Suggestion Results

Table 5.4: Average time (milliseconds) to type a character before and after suggestions

	Mean		**Median**		
Number of Keystrokes	**Prev**	**After**	**Prev**	**After**	P-value<0.05
5	238.07	196.80	176.85	147.60	True
10	274.23	238.93	213.80	195.90	True
20	312.20	298.40	251.80	241.90	True

The results from Table 5.4 show that the average time it takes to write a character is less after the suggestion appears on the screen. The biggest difference is immediately after the suggestion appears, with the difference being greatest when there are 5 keystrokes between the suggestion and the character being typed. The smallest difference is when there are 20 keystrokes between the suggestion and the character being typed.

A Mann-Whitney U test was performed to compare the average time before and after the suggestion. The p-value was less than 0.05 in all three cases, which suggests that the difference is statistically significant. The average time it took users to type a character before and after the suggestion appeared was also compared. It was found that 71% of users took less time typing per character after the suggestion appeared when there were 5 or 10 keystrokes between the suggestion and the character being typed. When there were 20 keystrokes between the suggestion and the character being typed, 64% of users took less time typing per character after the suggestion appeared.

Table 5.5 shows the average and median difference in the time it takes to type a character before and after a suggestion appears, for different numbers of keystrokes and different types of responses. The results show that the average and median time to type a character decreases after a suggestion appears. This suggests that the typing speed increases after the suggestion is displayed with the biggest difference when the suggestion is rejected.

Table 5.5: Difference in time (in ms) to type a character before and after suggestion appears for different suggestions

	5 Keystrokes		**10 Keystrokes**		**20 Keystrokes**	
Response	**Mean**	**Median**	**Mean**	**Median**	**Mean**	**Median**
Full Acceptance	-44.82	-11.0	-40.53	-12.33	-13.51	-2.74
Partial Acceptance	-72.36	-24.3	-68.23	-16.12	-54.7	-20.96
Rejection	-95.85	-12.83	-62.29	-11.49	-54.11	-10.95

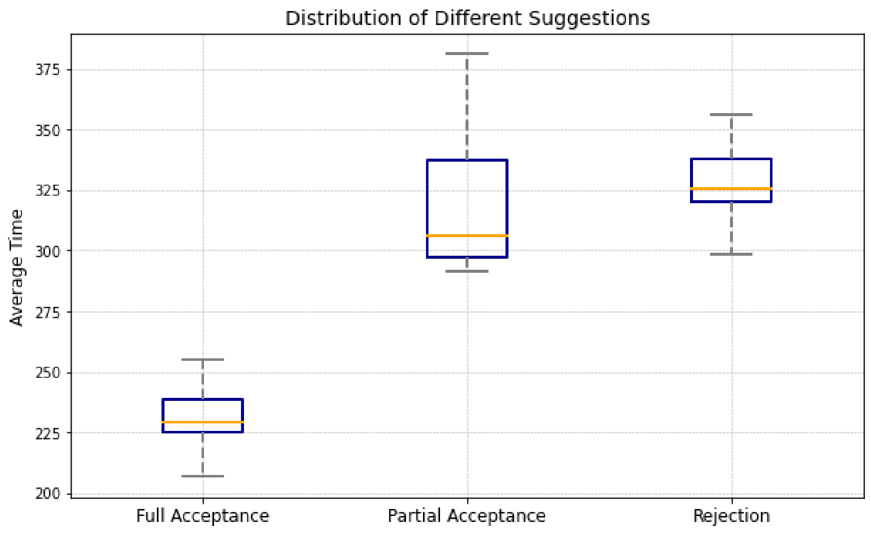


Figure 5.6: Box plot of time (in ms) taken to write a single character under different suggestions

The box plot in Figure 5.6 shows the time (in ms) taken to type a character under three different types of suggestions. Next, we found that 100% of participants took less time on average to write a character when they fully accepted a suggestion than when they partially accepted or rejected a suggestion. Additionally, 70.4% of participants took less time to write a character when they partially accepted a suggestion than when they rejected a suggestion. One reason the average time it takes to write a single character is lowest during full acceptance is that participants can accept suggested phrases without having to type anything themselves by pressing the Tab or Right Arrow keys on the keyboard.

6 LIMITATIONS

Google Smart Compose is a feature that suggests words and phrases as you type. One of its features is to make personalized suggestions based on the writer's previous writing plus the current text. However, in our experiment, we used a completely new Google account that didn't have any record of the user's previous writing. This meant that the suggestions were not as personalized as they could have been.

Secondly, we used the Macbook laptops as our device for writing which not all the participants might be comfortable typing.

7 CONCLUSIONS

In conclusion, this thesis investigated the effect of Google's Smart Compose on open-ended writing. This research expands the study of predictive text systems to include open-ended writing.

The thesis analyzed the quantitative and qualitative differences between writings with Smart Compose enabled and disabled using Coh-Metrix values. Additionally, the thesis studied the number of keystrokes required to write a character, as well as the effect of Smart Compose on the writing process in terms of typing speed and the time it takes to accept or reject suggested phrases.

The results showed that Google's Smart Compose does not have a significant effect on open-ended writing. However, the suggestions displayed on the screen do have an immediate effect of increasing typing speed.

In the future, a study can be done on the effect of predictive text systems by building a custom predictive text system and analyzing the effect under various conditions, such as the length of suggestions, frequency of suggestions, and so on. We can also study the effects of suggestions appearing on the screen as a human-computer interaction research topic. We can investigate whether the design of the suggestions affects focus while writing and the rate of acceptance of suggestions.

BIBLIOGRAPHY

[1] Arga Adyatama, *Text generation with markov chains*, (2020).

[2] Farag Ahmed, Ernesto De Luca, and Andreas Nürnberger, *Revised n-gram based automatic spelling correction tool to improve retrieval effectiveness*, Polibits **40** (2009), 39–48.

[3] Ohoud Alharbi, Wolfgang Stuerzlinger, and Felix Putze, *The effects of predictive features of mobile keyboards on text entry speed and errors*, Proc. ACM Hum.-Comput. Interact. **4** (2020), no. ISS.

[4] Kenneth C. Arnold, Krysta Chauncey, and Krzysztof Z. Gajos, *Predictive text encourages predictable writing*, Proceedings of the 25th International Conference on Intelligent User Interfaces (New York, NY, USA), IUI '20, Association for Computing Machinery, 2020, p. 128–138.

[5] Mia Xu Chen, Benjamin N. Lee, Gagan Bansal, Yuan Cao, Shuyuan Zhang, Justin Lu, Jackie Tsay, Yinan Wang, Andrew M. Dai, Zhifeng Chen, Timothy Sohn, and Yonghui Wu, *Gmail smart compose: Real-time assisted writing*, Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (New York, NY, USA), KDD '19, Association for Computing Machinery, 2019, p. 2287–2295.

[6] Nestor Garay-Vitoria and Julio Abascal, *Text prediction systems: A survey*, Universal Access in the Information Society **4** (2006), 188–203.

[7] Matt Gnacek, Eric Doran, Sharon Bommer, and Philip Appiah-Kubi, *The effectiveness of smart compose: An artificial intelligent system*, (2020).

[8] Arthur C. Graesser, Danielle S. McNamara, and Jonna M. Kulikowich, *Coh-metrix: Providing multilevel analyses of text characteristics*, Educational Researcher **40** (2011), no. 5, 223–234.

[9] Hozan K. Hamarashid, Soran A. Saeed, and Tarik A. Rashid, *A comprehensive review and evaluation on text predictive and entertainment systems*, Soft Computing **26** (2022), no. 4, 1541–1562.

[10] Daniel Hladek, Ján Staš, and Matus Pleva, *Survey of automatic spelling correction*, Electronics **9** (2020), 1670.

[11] Slava Katz, *Estimation of probabilities from sparse data for the language model component of a speech recognizer*, IEEE transactions on acoustics, speech, and signal processing **35** (1987), no. 3, 400–401.

[12] Karen Kukich, *Techniques for automatically correcting words in text*, ACM Comput. Surv. **24** (1992), no. 4, 377–439.

[13] M. Mor and A. S. Fraenkel, *A hash code method for detecting and correcting spelling errors*, Commun. ACM **25** (1982), no. 12, 935–938.

[14] Jean Veronis, *Computerized correction of phonographic errors*, Computers and the Humanities **22** (1988), no. 1, 43–56.

[15] Ziang Xie, Anand Avati, Naveen Arivazhagan, Dan Jurafsky, and Andrew Y. Ng, *Neural language correction with character-based attention*, 2016.

[16] Emmanuel J Yannakoudakis and David Fawthrop, *An intelligent spelling error corrector*, Information Processing & Management **19** (1983), no. 2, 101–108.

[17] Azita Yazdani, Reza Safdari, Ali Golkar, and Sharareh Rostam Niakan Kalhori, *Words prediction based on n-gram model for free-text entry in electronic health records*, Health Information Science and Systems **7** (2019).

[18] Mingrui Ray Zhang, He Wen, and Jacob O. Wobbrock, *Type, then correct: Intelligent text correction techniques for mobile text entry using neural networks*, Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology (New York, NY, USA), UIST '19, Association for Computing Machinery, 2019, p. 843–855.

VITA

Sijan Shrestha was born in Nepal in 1998. He completed his high school education there before moving to United States to attend the University of Mississippi. He graduated with a Bachelor of Science degree in Computer Science in May 2021. He then continued his studies at the University of Mississippi, where he earned a Master of Engineering Science degree with an emphasis in Computer Science in August 2023.