THE PENNSYLVANIA STATE UNIVERSITY
SCHREYER HONORS COLLEGE

SCHOOL OF SCIENCE, ENGINEERING, AND TECHNOLOGY

UTILIZATION OF RECEIVER-SPECIFIC DICTIONARIES FOR IMPROVED AUTO-COMPLETION FUNCTIONALITY

SAI PRAVALLIK REDDY GUJJULA
Spring 2018

A thesis
submitted in partial fulfillment
of the requirements
for baccalaureate degree
in Computer Science
with honors in Computer Science

Reviewed and approved* by the following:

Jeremy Blum
Associate Professor
Thesis Supervisor

Linda Null
Associate Professor
Faculty Reader

David Witwer
Director of Capital College Honors
Honors Adviser

* Signatures are on file in the [Schreyer Honors College Program].
NOTE: Signatory Page Template must be completed on special archival paper & turned in as a hardcopy

ABSTRACT

The auto-complete on mobile phones is one of the most commonly used keyboard features. It assists a user by predicting the intended choice of words based on partial input. These predictions increase productivity by reducing the number of keystrokes required to obtain an intended word. The current day auto-complete implementations focus on making the prediction algorithms efficient and precise by studying the user's choices of input. This proves to be highly beneficial as the prediction algorithms become user specific, and predict words based on the manner in which the user communicates. However, there is a need to consider another perspective in order to make these algorithms more precise and efficient, i.e. the receiver whom the user communicates with, and the need to make the word predictions receiver specific.

The importance of receiver specificity lies in a user's tendency to communicate using different set of words with a variety of receivers. Considering the receiver specificity in the choice of words when predicting, the auto-complete feature requires an additional resource to store words specific to a set of receivers. Together with a common dictionary that contains words that are common to all users, there is a need for receiver specific dictionaries in which each consists of words that are specific just to the user and a receiver. This should ensure that the predicting algorithm will select a more receiver-associated word from the related receiver dictionary, thus, increasing precision.

This thesis analyzes the implementation of an auto-complete algorithm with the receiver-specific dictionaries. It compares this implementation with the existing form of implementation that uses only one common dictionary. The results are based off a comparison on the number of keystrokes saved to obtain an intended choice of word using either of the implementations. The receiver specific implementation saw about a 25% reduction in the number of keystrokes used as compared to the baseline implementation with a single dictionary of all the previously used terms.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

Without the support of Dr. Jeremy Blum, this paper would not have been possible. Being my thesis advisor, he had provided me with useful insights and improved my understanding on the concepts related to this paper. He had been extremely supportive during my difficult times, and made sure that he did everything in his boundaries to help me get out of it. I would also like to take this opportunity to thank Dr. Linda Null for being my thesis reader and supporting me when I needed her the most. Along with this, I would like to thank my family and friends for being supportive with my decision on writing this paper. This would not have been possible without them, and all of your presence made me a stronger person.

Chapter 1: Introduction

By 2018, the number of mobile phone users is forecast to reach 4.93 billion [1]. One of the most important factors affecting the productivity of these systems is the mobile keyboard. A mobile keyboard consists of various value-added features that define its capabilities, and one of the most important is the auto-complete. This feature provides the user with various word predictions based on a partial input. This partial input can be of two types – non-space characters or a space. A partial input of a series of non-space characters leads to predictions that complete the input, whereas, a partial input of a space leads to predictions that can follow the previous word. The user can then choose a prediction if it matches the intended word. This reduces the number of characters that are to be typed, thus, saving time in the process.

1.1. Significance

With an increased usage of mobile keyboard for search related queries, the lack of user intended word suggestions leads to reduced productivity [2]. The auto-complete feature in keyboards addresses this issue through its beneficial capabilities. The significance of auto-complete lies in correcting spelling on known-item searches, building confidence amongst academic personnel on unfamiliar topics, speeding up search process, and augmenting search-term vocabulary [3]. However, the level of significance depends on the level of personalization in relation to the user. Therefore, in order to address the requirements of a wide range of mobile keyboard users, there is a need for the auto-complete feature to be specific in nature by predicting words based on the receiver.

1.2. Possible Improvement – Current Focus

The current day implementation of the auto-complete feature is indeed efficient and predicts precise words based on the user's partial input. It tunes itself to meet the user's communication style and suggests words accordingly. This approach however has limitations when a user employs different communication styles depending on the receiver – the person whom the user is communicating with. For example, a user can possibly use a certain set of informal personalized words with his friends, and a set of formal personalized words with all his colleagues. Accordingly, suggesting a colleague from the set of informal words or suggesting a friend from the set formal words might not exactly prove beneficial for the user. As a result, there is a need to look into the aspect of receiver specificity on top of the user's personalization in order to obtain precise predictions.

The thesis looks into the aspect of introducing a separate dictionary for each of the receivers in order to increase the user's level of personalization. This dictionary shall constitute of terms that are specific to the respective user-receiver communication. This ensures that the user-receiver's specific choice of terms is preserved, and not generalized with all the other non-related terms.

Chapter 2: Literature Review

2.1. Studying the Auto-Complete Feature

The auto-complete feature's role in a mobile phone ranges from providing the user with an intended choice of word to introducing the user with new ideas. Furthermore, the feature depends on various factors such as the manner of implementation, its acceptance amongst the users, and the completeness in the suggestions. As a matter of fact, in order to make the feature effective, these factors have to be put into their best use. Hence, improving these factors plays a key role in providing the user with an enhanced value-added feature.

2.1.1. Auto-Complete Models

An auto-complete implementation is designed based on various models which are dependent on its contextual purpose. Each of these purposes range from a simple textbox search on an online website to an extensive search on a large-sized database. These models are designed based on various algorithmic data structures, which organize data for efficient search capabilities.

2.1.1.1. Anima Model

A trie, also termed as a prefix tree, is a data structure that is used to dynamically store data with the help of key strings. The trie consists of nodes, and the starting node of the structure is termed as the root which has an empty string as a key. Each traversal down the trie to a lower level node, the key of the upper level node is a prefix of the key of the lower level node. The Anima keyboard is designed based on the trie data structure with weights at each node [4]. Each time a user enters a character, the prediction engine of the keyboard descends from the root to a lower level. With the help of the weights at each node, the engine is able to consider the number of times a user has traversed a particular path [4]. In addition to this, the engine consists of learning accelerators which assist in understanding the user's previous form of written communication.

2.1.1.2. Tree Model

This model is based on the tree data structure, wherein, each data point is called a node, and all the nodes are related to one another based on a parent-child relation. Mitica Manu, together with his team, came up with an algorithm that creates a hierarchical tree with a particular dataset [5]. Each word in the data set is divided into two parts – an 'initial partial word', and the 'rest of the word'. The 'initial partial word' acts as the parent in the word relation, and the 'rest of the word' as the child [5]. The resulting tree consists of all the relations between various 'initial partial word' and 'rest of the word', and is then used for search operations. In addition to that, it is also to be noted that the 'initial partial word' selected for the relation needs to be a common part of various other words. As a result, when a user enters an input that matches any parent, the children affiliated with it are produced as suggestions.

2.1.1.3. Graph Mining Model

Graph, in the field of computer science, refers to a data structure that consists of set of ordered pairs, called edges and arcs, of certain entities called nodes. In the case of an autocomplete implementation, each node in a graph represents a particular word in a legal sentence, and each edge the possible link between one node and another. Accordingly, graph mining refers to the procedure of searching for possible word suggestions based on the graph created with a set of legal sentences. Based on a study conducted by Neeraj A.

and Mrutyunjaya S., the utilization of graph mining as part of generating a suggestion list significantly outperforms existing document specific autocomplete search techniques [6]. In this study, various sentences are preprocessed to form graphs with nodes of two categories – keywords and stop words. After which, each of the keyboards is processed through a hashing algorithm, which maps data of an arbitrary size to data of fixed size [6]. The resulting graph is then used as the dataset for the necessary search operations.

2.1.2. Organization of Dictionaries

From the auto-complete feature's point of view, the dictionaries refer to datasets that constitute the list of all possible words for the prediction purpose. Each auto-complete implementation utilizes the dictionary in a unique manner to address its own needs. The dictionaries vary from a standard dataset of pre-defined words to a developing dataset that learns from the user's usage of words.

2.1.2.1. Anima Keyboard

The Anima keyboard utilized two forms of dictionaries as part of the testing purpose [4]. The first being a formal predefined dictionary with a word count of over a hundred thousand words. The second being a dictionary based on words from the user's past communication. The advantage of this form of a dictionary lies in minimizing memory usage by avoiding unnecessary terms from a predefined dictionary [4]. The purpose of utilizing two forms of dictionaries was to compare the performance of the experimental implementation with a standardized form. In addition to this, with a possible increase in size of the user dictionary, the prediction engine of the keyboard incorporated pruning [4]. Pruning assisted in reducing the size of the dictionary by considering the least commonly used terms in the dictionary as irrelevant.

2.1.2.2. Graph Mining Model

The study discussed earlier on the graph mining model, utilized a dictionary based on the content that is to be searched for, rather than a predefined dictionary [6]. The model preprocessed a fixed content by creating a graph with various nodes as the words, and edges as the relationships. After which, the graph was used as a form of dictionary to suggest possible terms as part of the search process [6]. The advantage of this form of a dictionary lied in the ability to work with a generic type of content.

2.1.3. Retrieval Procedure

Upon considering the models and datasets, it is important to understand the retrieval procedure of the possible suggestions. The efficiency of each retrieval procedure is dependent on the model and datasets of an implementation. Furthermore, the precision of a retrieval procedure lies in being able to produce an intended choice of word. As a result, it is important to consider how the retrieval procedure varies for different models.

2.1.3.1. Anima Keyboard

The prediction engine of the keyboard, traverses down the trie (discussed in Auto-Complete Models – 2.1.1) from the root for every character the user types [4]. Furthermore, for each of the nodes in the trie, there exists a weight representing the number of times the user has traversed the path. In the case of a newly typed word, the trie tracks traversed paths and adds words into its structure. Based on the weights of the nodes visited, the keyboard computes conditional probabilities, which assist in identifying the most probable choice of word [4]. With these probabilities, the user is then suggested words as part of the auto-complete feature.

2.1.3.2. Graph Mining Model

In this model, the hashing algorithm is used to map keywords of an arbitrary size to data of a fixed size [6]. Each of the 26 alphabets is assigned a digit value based on its position in the alphabet series. After which, the digits associated with the input characters of the keyword are multiplied, and the modulus is calculated by dividing the product with the size of the hash table (the total number of alphabets in this case) [6]. The resulting value is then used to search for the corresponding numbered location in the hash table. This method of hashing evenly distributes the keywords in the hash table, thus, speeding up the search process.

In addition to this, the method requires at least two characters to be input for processing through the hashing algorithm [6]. This helps limit the suggestions into a comprehensible number, and avoids giving suggestion that are irrelevant. This reduces the search space, and speeds up the search process. Upon inputting at least two characters, suggestions are displayed based on the traversal of the graph with respect to its type of word – keyword or stop word.

2.1.3.3. Phrase Retrieval

A study tried to implement the feature to suggest phrases that could be possible extensions to an initial word in order to complete an intended sentence [7]. The intent of typing at a fast pace can be considered as one of the basic reasons behind the auto-complete feature. Furthermore, predicting an entire phrase to complete a sentence can prove to be faster than predicting an extension of a word [7]. Thus, it can be understood that predicting a suitable phrase extension would prove more beneficial. The results based on the study of this implementation showed that, though people appreciated the phrase suggestions for content, style, and speed, a possible limitation could be seen as to the tediousness in suggesting a high-quality phrase unless there is something known on what the user intends to write about [7]. Hence, this particular aspect of the results can be given careful attention when designing an auto-complete implementation.

2.2. Additional Features

As part of an auto-complete implementation, it is often necessary to consider additional forms of improvements that are not directly related to its basic functioning. These improvements assist in increasing the precision of suggested terms by incorporating various measures to improve their credibility. As a result, these improvements can be considered as value-added features of an auto-complete implementation.

2.2.1. Tolerating Errors

An initial step to improving a simple auto-complete implementation lies in enabling it to tolerate errors. Often, a user mistypes an intended word, thus, leading to an incorrect initial partial string. As a result, a simple implementation will not be able to suggest a reasonable extension to the user-intended string. In order to resolve this, a study was conducted to ensure that the feature be able to tolerate the errors [8].

2.2.1.1. K-extension

The study suggests an algorithmic addition, named as the ‘k-extension’, to the basic implementation such that it would tolerate errors [8]. The ‘k-extension’ algorithm takes an initial partial input string with an edit distance of length k. This edit distance allows an implementation to conduct a certain number of moves to alter the partial input string, and produce a modified list of possible extensions [8].

2.2.1.2. Q-Gram & Trie-Based

The principle of k-extension is used as part of two auto-complete strategies – Q-Gram & Trie-Based – that improve the ability to tolerate errors. The Q-Gram strategy deals with looking into all the error-correction possibilities of a user input with a certain k-distance [8]. For example, if user inputs a five letter word and the k-distance is set to one, the strategy looks into all the possible strings that can be attained with the change in one letter to the user input. However, it is important to note that only some of these strings are legal. On the other hand, the Trie-Based strategy deals with looking into the error correction possibilities based on user's partial inputs at each point in time. These partial inputs are taken as prefixes, which are then used to look up for legal error-correction possibilities [8]. The results from the study show that the Trie-Based strategy is indeed a better choice as it reduces redundancy and repetition of error-correction possibilities.

2.2.2. Temporal Dynamics

Often times, the intended user input largely relies on temporal dynamics. Temporal dynamics, in field of word prediction, refers to the user's usage of a particular set of words during a certain period of time. These time relative set of words tend to minimize the search space by looking into only the relevant words of a certain time frame. As a result, this speeds up the suggestive capabilities of an implementation.

2.2.2.1. Based on Information Retrieval

Stewart Whiting's dissertation on 'information retrieval' discusses how information retrieval is dependent on temporal dynamics [9]. His dissertation states that, in the field of information retrieval, there is a presence of several temporal dynamics driven by past information behavior. This includes word and phrase use in a time-based collection [9]. Through his dissertation, he proposes and evaluates query auto-completion based on time-sensitivity [9]. This deals with how the temporal dynamic similarity of word and phrase can be used to identify relationships between them.

2.2.2.2. Anima Keyboard

The Anima keyboard learns the user's daily patterns and partitions a day into several intervals of time. The keyboard then suggests based on the words that exist in a particular time partition [4]. This tends to produce better predictions as the study indicates that user's choice of words in relation to the time of his/her input [4].

2.3. Evaluation Procedures

The level of success of an auto-complete implementation largely depends on its effectiveness in suggesting an intended word-choice. In order to evaluate the effectiveness, we would have to utilize certain established procedures that assist in doing so. Furthermore, these procedures can assist in directing the initial phase of the implementation by providing an insight of an expected result. As a result, performing timely evaluations with the help of these procedures can help gauge the effectiveness of an intended implementation.

2.3.1. Comparing Implementations Based on a Given Set

One of the most basic forms of evaluation relates to processing an implementation based on a given set of data. This provides an opportunity to compare the results with evaluations of other implementations, and analyze the extent to which the new implementation is successful.

2.3.1.1. Anima Keyboard

The auto-complete feature of the Anima Keyboard was evaluated with and without an initial dictionary by computing the hit ratio over how many predicted characters were returned in each partition [4]. The results showed that the implementation worked better without an initial dictionary. In addition to that, the results also showed that with an increased number of time partitions and the assistance of pruning (discussed earlier), the precision of the auto-complete predictions improved [4].

2.3.1.2. Graph Mining Model

The graph mining model underwent a performance evaluation to understand the impact of utilizing graphs as part of data organization. As part of the evaluation, two different text files with census data of various countries were used as the data sets [6]. Accordingly, these text files were searched upon by two groups of participants with and without the graph mining model. The results indicated that the use of graph mining significantly decreased search time [6].

2.3.1.3. Octopus

Xiao Jun, together with his team, developed an evaluation tool named ‘Octopus’ that efficiently measures the impact of touch-screen keyboard correction and recognition algorithms [10]. The tool consists of an application that compares and analyzes a predefined data-set with various new transcribed phrases [10]. Furthermore, the application produces a word score and a character score that help in understanding the correctness of the intended input [10]. As a result, this particular tool stands out efficient in identifying the level of closeness of the intended input to the suggested input.

2.3.2. Scoring the Precision of Predictions

Another form of evaluation relates to providing scores based on the precision of suggested input. These scores can be anywhere between a certain range, and they provide an idea on their standing based on the maximum attainable score. Hanmin Jung, together with his team, proposed an evaluation procedure with a similar conceptual idea [11]. The procedure consists of two different scales, one named as the ‘precision of task’ score and the other as the ‘precision in satisfaction’ score [11]. The ‘precision of task’ score tends to be based only on the available terms in the dataset. On the other hand, the ‘precision of satisfaction’ score relates to the level of user satisfaction on the basis of the correctness in the user’s intended choice of word. The scores produced based on these scales tend to provide an understanding on the progress of any auto-complete implementation. Hence, it can be seen that such a procedure tends to allow one to gauge the implementation’s direction of progress.

In summary, evaluating procedures tend to provide a well-established insight on the stand of a particular implementation. They provide an understanding on how established an implementation stands. Furthermore, they tend to provide a chance to analyze the likelihood of an implementation being successfully improving the auto-complete feature. As a result, paying close attention to the results obtained through the evaluations strengthens the value of an implementation.

2.4. Emphasis of Study

This paper puts forth the idea of introducing a separate dictionary for each of the receivers as part of improving the auto-complete prediction capabilities. This idea emerges from the fact that a user does not communicate with everyone in a similar manner. A user communicates with a variety of receivers using formal English terms and informal non-English terms. In addition, a general trend shows that a number of current day users communicate in more than one language. Often times the terms that belong to a non-English language are spelt out using the English alphabet, making those terms specific only to a certain set of receivers. Accordingly, terms used as part of a non-English language communication are not required to be part of the suggestions of an English language communication, and vice versa. A separate dictionary for each of the receivers would likely improve the performance of an auto-complete system by suggesting terms based off the receiver a user communicates with.

Chapter 3: Methodology

In this section of the paper, we discuss the overall structure of the receiver specific implementation, and the baseline implementation – two types of implementations used as part of the study in order to identify the effectiveness of a separate dictionary for each of the receivers. Following that, we look into the data sources, the auto-complete algorithm with the receiver-specific dictionaries, and the data that has been collected as part of the evaluation process.

3.1. Basic Elements

The study adopts an auto-complete implementation that makes use of various technical aspects that belong to the field of Computer Science. Each of these technical aspects have been simplified to only satisfy the study requirements. As a result, it is important to consider the functionalities of each of these simplified technical aspects in order to understand the study better.

3.1.1. Trie Node

A trie node is a basic unit of the study's auto-complete implementation. It holds the information of a character and the other trie nodes that consist of characters that follow the current character. For example, considering the words "the" and "to," the first trie node will have "t" as the character, and trie nodes with characters of "h" and "o" attached to it. Accordingly, the trie node with the character "h" will have a trie node with the character "e" attached to it. Whereas, the trie node with the character "o" will not have any trie nodes attached to it as no character follows the character "o" in the word "to."

Figure 1 shows an illustration of how the trie nodes are connected to one another.

3.1.2. Trie

A trie can be termed as the entire collection of trie nodes that make up all the words in a dictionary. Each dictionary is expected to have a separate trie. Using the partial input provided by the user, the trie produces a set of terms that start with the partial input. Taking these terms into consideration, a function in the dictionary identifies the terms with the highest frequencies and suggests them to the user. The advantage of using a trie lies in its ability to connect terms based on the sequence of characters that each term is made up of. This helps obtain all the terms that begin with a certain partial sequence of characters.

3.1.3. Main Dictionary

An auto-complete algorithm requires data support from a non-user related source in order to provide possible suggestions for new words that the user might use to communicate. This ensures that the user is suggested new words when none of the receiver's dictionary words match the user's partial input string. Together with this, it exposes the user to certain new terms that the user might want to use. For the case of the study, the main dictionary consists of words from an online English repository.

3.1.4. Message Formatter

The role of the message formatter lies in its functionality to remove the redundant terms out of the user communication data. Redundant terms include emoticons, use of excessive punctuation, and time at which the user sent each of the messages to the receiver. In addition to that, the formatter removes any names associated with the user and the receiver as part of maintaining anonymity. The significance of the formatter lies in its ability to automate the process for all the messages in order to suit the data required by the study's auto-complete implementation.

3.2. Receiver Specific Implementation

Receiver specific implementation refers to the auto-complete implementation that includes receiver specific dictionaries for each of the receivers that the user communicates with. It includes all the basic elements discussed in the above section in addition to these dictionaries. In the following sub-sections, we discuss various features that the implementation comprises of, and the theory on how the implementation works.

3.2.1. Receiver Dictionary

The receiver dictionary can be termed as the dataset that holds the terms that have been used by the user with a given receiver. It includes the frequencies for each of these terms, and a receiver trie (discussed in the next sub-section) derived from all these terms. A functionality of the receiver dictionary is to obtain a list of words associated with a partial input with the help of the receiver trie. Using this list of words, it suggests the top three words with the highest frequencies.

The receiver dictionary also consists of terms used by the receiver as part of the communication. The idea of including these terms lies in making it a robust resource. This ensures that the dictionary is able to suggest certain contextual terms that the user might not have used as part of the user-receiver conversation.

3.2.2. Receiver Trie

The receiver trie is formed using all the terms that a receiver dictionary consists of. The advantage of a receiver trie lies in its ability to ease the process of identifying terms that can be associated with a given partial input for a given receiver. In addition to that, the availability of a separate receiver trie for each receiver dictionary removes the need for searching the entire dictionary to identify if a term exists.

3.3. Baseline Implementation

The need for comparing the receiver specific implementation with a non-receiver specific implementation makes way for the baseline implementation. This implementation consists of a single dictionary, called the sender dictionary, in which all the terms previously used as part of the user's communication with all his/her receivers are stored. Accordingly, a sender trie is associated with the sender dictionary to help identify the terms that match a given partial input. One important difference between a receiver specific implementation and the baseline implementation is the fact that the latter does not associate the terms with its respective receiver. In addition to that, it is expected to be larger than an individual receiver specific dictionary. Upon comparing the performance of the baseline implementation with the receiver specific implementation, we can identify the level of impact receiver specific dictionaries have, in predicting the intended terms.

3.4. Implementation Steps

As an initial insight, it is important to note that the data used for either of the implementations is divided into three parts – 60-20-20. The first 60 percent of the data is used to train the implementation and create an initial model. The next 20 percent is used to validate the implementation and improve the predictive algorithm by making changes to how the predicted words were scored and suggested. The final 20 percent of the data is used to evaluate the implementation, and the results obtained from it are reported below in the following section. Below are the steps that are carried out as part of both the receiver specific and the baseline implementations:

- **Step 1:** A main dictionary is created with the words obtained from the predefined English dictionary source.
- **Step 2:** Using the initial 60 percent of the data, for each of the receivers, a separate receiver dictionary is created and populated with the user’s terms that were used as part of the communication with the receiver. Simultaneously, all these terms are also added into a single sender dictionary for the baseline implementation. On the other hand, the receiver’s terms used as part of the communication with the user are added into the main dictionary. This helps improve the quality of suggestions when a partial input cannot be matched to any of the terms in the receiver specific dictionary as discussed in the earlier section.
- **Step 3 (Only during development phase):** Both the implementations are validated using the next 20 percent of the data. This helps identify the effectiveness of the algorithm, and opens up an opportunity for changes to be made in order to improve the algorithm.

- **Step 4:** As a final step, upon being certain that the algorithm functions effectively for the validation data in step 4, both the implementations are evaluated using the last 20 percent of the data. The results indicate the number of keystrokes saved using the receiver specific implementation as compared to that of the baseline implementation. In addition to that, they also indicate the results on the number of keystrokes saved using either of the implementations as compared to using only the main dictionary where all the terms are added directly to it.

The impact that the receiver specific implementation holds is proportional to the number of keystrokes saved using the receiver specific dictionaries in place of a single sender dictionary.

3.5. Data Sources

The level of impact the inclusion of receiver specific dictionaries brings into an auto-complete implementation relies heavily on the practicality of data. In other words, the data used as part of implementation is required to be real-time data that users actually use to communicate with various individuals. As a result, the study utilizes actual user data that has been as part of user-receiver communication.

The degree of variety in user communication data is proportional to the significance receiver-specific dictionaries hold as part of an auto-complete feature. An increased variety in data leads to the dictionaries holding terms specific to a particular receiver. Upon which, the implementation can suggest based off a personalized list of terms for a given receiver. As a result, the data is obtained from users who communicate in more than one language using the English script.

The user data that is used as part of the study includes receivers with whom the user communicated using a wide range of non-English terms. These terms belong to the regional language, Telugu, which were spelt out using the English alphabet based on the syllables. Each user's data consists of a variety of receiver datasets that consist of English terms, non-English terms or a combination of both.

The data used as part of the implementation can be categorized into two types – predefined data and collected data. The predefined data consists of an English dictionary that includes the top 10,000 frequently used words with their respective frequencies [14]. This was obtained from the repositories available online. The collected data includes data obtained from a user's communication with various individuals (receivers) [14]. The messages are exports from the "Whatsapp" messaging application. The data is processed to exclude the details such as the time of conversation as an initial step using the message formatter (discussed in the earlier section). This processed data is then utilized as receiver specific data and sender data for the receiver specific implementation and the baseline implementation respectively. The processed data of the receivers is included as part of the main dictionary in the implementations.

The data obtained through the messaging application can be closely related to the general messaging. The sole difference lies in the use of internet to send messages rather than the mobile network that is used in general messaging. The data is made sure that names related to either users or receivers are removed. Each run of either of the implementation tests for number of keystrokes saved as part of a user's communication with an average of 9 different individuals. The overall data includes conversations of 3 different users and their respective receivers.

3.6. Algorithm

The main motive of the algorithm lies in suggesting the intended word in the minimum possible number of keystrokes. The ideal case in suggesting an intended term happens when the user has typed in a correct partial term. However, the fact that a user might stand a chance in typing an incorrect partial string requires the algorithm to look into letters adjacent to the user's choice of keyboard keys. This ensures that the algorithm is robust in accepting the user's variety of inputs, and suggesting terms that constitute the user's actual intended term. The following steps look into the manner in which the algorithm functions to determine the user's intended word.

1) The algorithm receives as an input the receiver whom the user is communicating with and the user's intended term (since the data is a collection from a past communication).
2) It then reads the intended term character by character, and considers the inclusion of each character as a separate partial input. For example, taking the case of the term "how," the resulting partial inputs are "h," "ho," and "how." Each of these partial inputs is the result of including the next character of the term onto the previous partial input, where the first previous partial input is empty.
3) In addition to each of the partial inputs obtained by reading the next character of the intended term, the algorithm also creates partial inputs based on the adjacent keys of the next intended. This is provided if the trie consists of terms that can be related to these newly formed partial inputs. For example, taking the above case of the term "how," if "o" is the next character to be read, the algorithm

creates the partial input “ho.” In addition to this, the algorithm also creates partial inputs with the adjacent keys for the letter “o” on the keyboard – “p,” “l,” “k,” and “i” – resulting in “hp,” “hl,” “hk,” and “hi.” Upon which, it only proceeds with each of these partial inputs further down into the algorithm only if there are terms associated with these partial inputs in the trie.

4) The algorithm then considers all the terms that result from these partial strings, and then chooses the top three terms based on the weighted frequencies.
5) If any of these top three terms is the actual intended term, the algorithm scores the term suggestion with the number of keystrokes saved. This is the difference in the number of keystrokes that the user had to input and the length of the term. The worst case is when the user has to enter the entire intended term.
6) The algorithm repeats the same process for all the words that constitute the last 20 percent of test data, and sums the keystrokes saved for each term.

The algorithm is same for both the implementations. However, the difference lies in the dictionary that the algorithm actually looks into when predicting the term. For the receiver specific implementation, the dictionary considered is the one that is personalized to the receiver. On the other hand, for the baseline implementation, the dictionary considered is the sender dictionary that is common to all receivers. The idea behind a similar form of algorithm for both the implementations lies in creating a fair comparison that looks into only how useful the receiver specific dictionaries are, and not any other factors that could help improve one implementation over the other.

3.7. Data Collection

The importance of the auto-complete feature lies in its ability to save a user’s time by minimizing the number of keystrokes required to obtain an intended word. Due to this reason, the data collected through the implementation is the number of keystrokes that are required to type in the 20 percent of test data – using both the receiver specific implementation and the baseline implementation. Using the number of keystrokes obtained from each of the two implementations, the one with a lesser number of keystrokes signifies a better choice, and the difference helps indicate the level of impact one has over the other. Considering the expected results, it can be predicted that the receiver specific implementation will save a higher number of keystrokes as compared to the baseline implementation. The reason lies in the receiver specific dictionaries’ ability to preserve a user-receiver specific communication style, and not generalize it with various others.

Chapter 4: Results and Discussion

This section of the paper deals with the results obtained using both the implementations – receiver specific and baseline. The results include the keystrokes required to obtain the test data using either of the implementations, and the precision of the suggested words.

4.1. Keystroke Evaluation

A user's productivity can be highly related to the amount of work done in a certain time. The lesser the time spent to perform an exact amount of work, the more the productivity. Accordingly, from the context of an auto-complete feature, a user is expected to save time if less keystrokes are required than usual in obtaining an intended term. The amount of time saved is proportional to the number of keystrokes saved in obtaining an intended term. The number of keystrokes saved in obtaining an intended word is equal to difference in the number of characters in the term and the number of characters that the user had to type in to obtain the term. This includes the cases where the user has entered an incorrect partial input, and expects a correct intended term.

Taking all this into consideration, when processing the 20 percent test data, the overall keystrokes saved is equal to the sum of keystrokes saved for obtaining each term in the data. Using these results obtained from both the implementations, the difference in the number of keystrokes saved helps us identify the better implementation.

4.1.1. Implementation Findings

The baseline implementation utilizes a single sender dictionary for all its receivers, whereas the receiver specific implementation utilizes a receiver specific dictionary for each of the receivers that the user communicates with. The 20 percent of test data for each of the three users and their respective receivers was processed using both the implementations. The results for each of the three users can be seen in the tables shown below. It can be observed that the values for the baseline implementation are generally lower than the values for the receiver specific implementation, indicating that more keystrokes were saved using the latter.

4.1.2. Calculated Findings

The percentage difference column accounts for the difference in keystrokes saved using the receiver specific implementation as compared to the baseline implementation. However, it is to be noted that a high percentage in this column simply indicates the fact that the receiver specific implementation works well with a certain receiver. In order to obtain the average percentage, each of the values in this column was weighted based on the total number of characters that constitute the 20 percent test data. This helps produce a fair comparison since all of the receiver datasets are not of the same size. Accordingly, weighing the percentages based on total number of characters ensures that the average percentage is fairly calculated.

4.2. Discussion

Upon comparing the results for both the implementations, we observed that the receiver specific implementation generally tends to save more keystrokes compared to the baseline implementation. The average percentage for the difference in keystrokes saved is in the range of 25-30%. This is based on considering only a limited number of receivers for each user. A larger number of receivers could result in the sender dictionary becoming a larger entity and making the baseline implementation's dictionary more general. It is also important to note that in the case of the receiver dictionaries being of a smaller size, the receiver specific implementation might not necessarily suggest precise terms to the user. This is a consequence of a limited number of words obtained from past communication, leading to a lack in data for precise suggestions. As a result, it is important to identify a reasonable amount of past communication data before implementing the receiver specific dictionaries.

Considering the case of non-English terms being used as part of a user's communication, it can be noted that the receiver specific implementation tends to play a fair role in either case. The reason for this lies in the sender dictionary being a general source for all kinds of terms, and thus leading the baseline implementation to suggest non-receiver related terms to the user. On the other hand, with a limited number of terms, and the preservation of the user-receiver communication, the receiver specific implementation is able to offer more precise suggestions for either of the cases – English and non-English.

The user input correction is another aspect that the receiver specific implementation accounts for. Often, a user's partial input might consist of incorrect keys being typed in. This would result in the baseline implementation suggesting terms that are unrelated to the user. For example, an incorrect partial input for a non-English term could lead to the baseline implementation suggesting an English term. However, the receiver specific implementation due to a lack in terms that match the incorrect partial input, could possibly try correcting the partial input by introducing adjacent keys of the characters.

This ensures that the suggestions do not deviate from the user-receiver’s actual form of communication.

In summary, there is a definite necessity for introducing receiver specific dictionaries. However, their inclusion has to take into account the amount of past communication data that exists between each user-receiver pair. This ensures that the receiver dictionaries play an efficient role in suggesting terms, rather than being an overhead by not suggesting precise terms.

Chapter 6: Conclusion and Future Work

This section summarizes the paper, and discusses the findings associated with each of the implementations – receiver specific and baseline. It also compares the actual findings to that of the expected findings. As a final aspect, it discusses the achievements of this receiver specific implementation, and states what future work could help improve the implementation.

6.1. Summary of Study

The implementation's sole focus lies in introducing receiver specific dictionaries. As a result, both the baseline and the receiver specific implementations do not include various value-added measures that current day auto-complete implementations possess. For example, considering the Anima Keyboard that was discussed earlier in the Literature Review section, it used temporal dynamics as part of its auto-complete implementation in order to suggest more precise words. However, the implementations discussed in this paper do not include any such features, and focus only on to what extent receiver specific dictionaries help improve the auto-complete feature's capabilities.

The study of introducing receiver specific dictionaries as part of the auto-complete implementation suggests that personalizing a user's choice of words based on the receiver indeed helps improve the performance of the feature. It reduces the number of keystrokes required by a user to obtain an intended word, and saves time as a result. Based on the results obtained upon testing the implementation, we concluded that the introduction of these dictionaries for a user with a reasonable amount of past communication helps save a reasonable number of keystrokes.

6.2. Expected vs. Actual Findings

The study shows that the inclusion of receiver specific dictionaries tends to reduce the number of keystrokes required to obtain an intended term by 25-30%. These results are based off simple auto-complete implementations that use either a single sender dictionary or a group of receiver specific dictionaries. It is also important to note the fact that inclusion of receiver specific dictionaries ensures that incorrect partial inputs are better corrected using adjacent keys on the keyboard as compared to a single sender dictionary. Together with these findings, it is also important to note the fact that the functionality of receiver specific dictionaries largely depends on the preservation of past communication data. The larger the user-receiver past communication data, the more useful the receiver dictionaries tend to be as they preserve the user's choice of terms when communicating with the receiver.

6.3. Future Work - Optimizing the Data Requirements

The introduction of receiver specific dictionaries, though a seemingly good inclusion to an auto-complete feature, can prove to be expensive. For each specific receiver that the user communicates with, there is a need to associate a separate dictionary. Each of these dictionaries is expected to include the terms that have been previously used as part of the user-receiver communication with their respective frequencies. However, this leads to a lot of redundant data store as dictionaries are expected to constitute of various common terms. A solution for this issue lies in combining several of the dictionaries together to form a single receiver specific dictionary for a set of receivers with whom the user communicates using a similar set of terms.

References

[1] Statista Inc. (n.d.). Number of mobile phone users worldwide from 2013 to 2019 [Online]. Available: https://www.statista.com/statistics/274774/forecast-of-mobile-phone-users-worldwide/

[2] Roger Kwame. (2014, July 9). Mobile Technology for Increased Productivity and Profitability [Online]. Available: https://www.linkedin.com/pulse/20140709223705-198356635-mobile-technology-for-increased-productivity-profitability

[3] D. Ward, et al, "Autocomplete as a research tool: a study on providing search suggestions." *Information Technology and Libraries,* 31(4), 6, 2012.

[4] K. Kumar, et al, "A survey of computation offloading for mobile systems," *Mobile Networks and Applications* 18.1: 129-140, 2013.

[5] K. Furtner, et al, “Effects of Auto-Suggest on the Usability of Search in eCommerce,” *14th International Symposium on Information Science,* pp.178-190, 2015.

[6] S. Chaudhuri, R. Kaushik, "Extending auto-completion to tolerate errors," *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data,* ACM, 2009.

[7] P. Sakkos, et al, “Anima: Adaptive Personalized Software Keyboard,” *arXiv preprint arXiv:1501.05696,* 2015

[8] K. Arnold, et al, "On Suggesting Phrases vs. Predicting Words for Mobile Text Composition," *Proceedings of the 29th Annual Symposium on User Interface Software and Technology,* ACM, 2016.

[9] E. Badger, et al, "User-centric soft keyboard predictive technologies," U.S. Patent No. 8,782,556, July 15, 2014.

[10] M. Manu, et al, "Text prediction with partial selection in a variety of domains," U.S. Patent No. 8,504,349, August 6, 2013.

[11] S. Whiting, "Temporal dynamics in information retrieval," PhD dissertation, School of Computing Science, University of Glasgow, 2015.

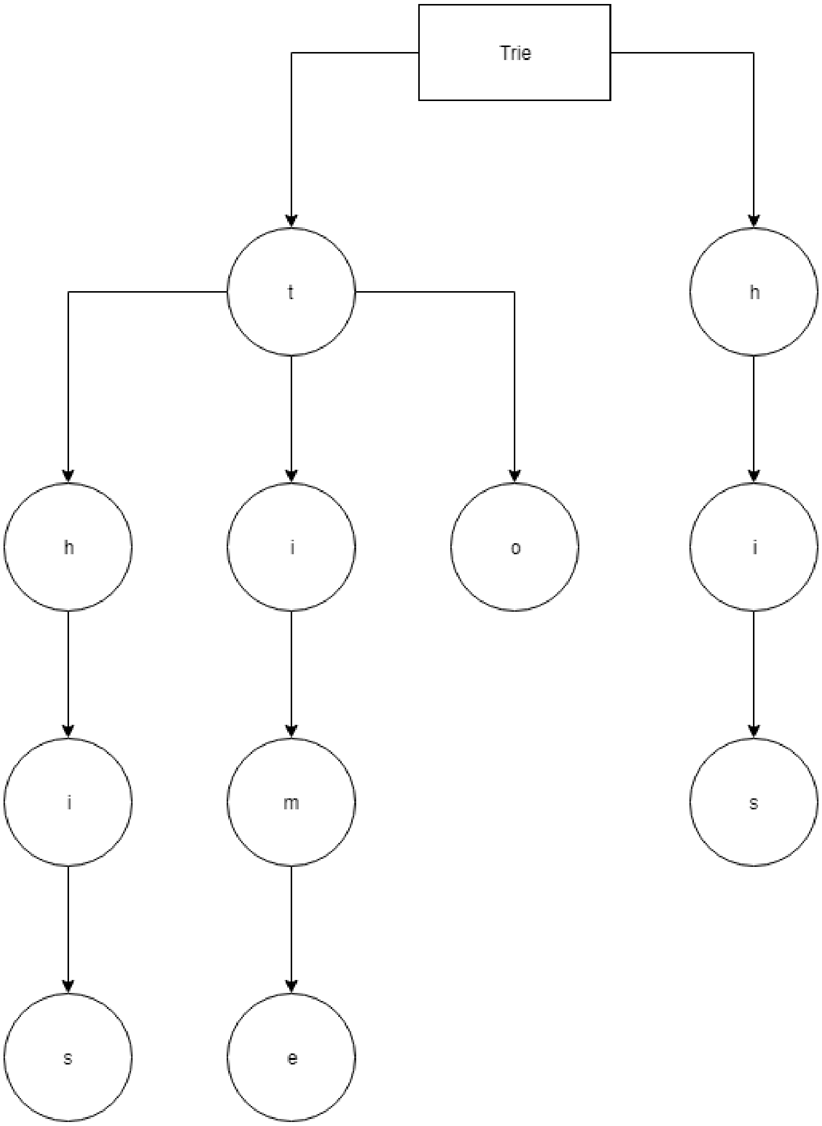
[12] X. Bi, et al, "Octopus: evaluating touchscreen keyboard correction and recognition algorithms via," *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ACM, 2013.

[13] H. Jung, et al, "Comparative Evaluation of Reliabilities on Semantic Search Functions: Auto-complete and Entity-Centric Unified Search," *International Conference on Active Media Technology*, Springer Berlin Heidelberg, 2009.

[14] Lancaster University. (n.d.). Word Frequencies in Written and Spoken English [Online]. Available: http://ucrel.lancs.ac.uk/bncfreq/lists/1_2_all_freq.txt

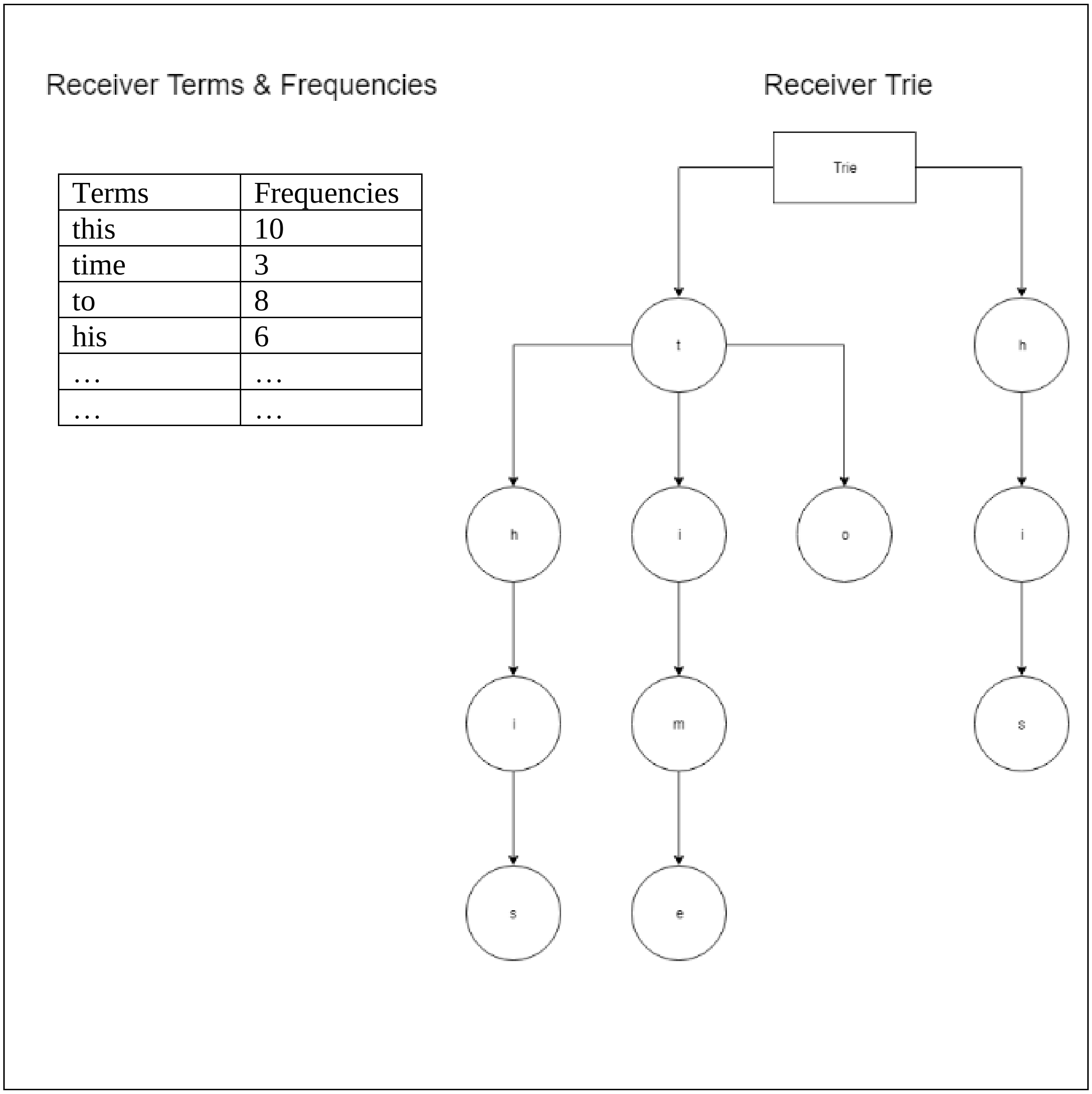
Figure 1: Structure of Trie

A trie constituting of words – "this", "time", "to", "his" – is expected to be in following form:



Taking the case of the user typing in a partial input of "t", the trie first accesses the trie node with the character "t". Taking this trie node to be the root, all of the associated terms are identified by reaching the leaves in the trie that can be accessed from this root. In this case, "this", "time", and "to" can all be accessed from the trie node "t". If the partial input is "th", the trie first positions itself at the trie node "h" by first passing through the trie node "t". Taking this trie node to be the root, all of the terms that can be obtained by reaching the leaves are identified. In this case, "this" is the only term that can be accessed.

Figure 2: Structure of Receiver Dictionaries



Terms	Frequencies
this	10
time	3
to	8
his	6
…	…
…	…

Each of the receiver dictionary is expected to have an associated trie that is derived based off the user's terms that were used as part of the user-receiver communication, and a list of terms and their frequencies. The trie is used to obtain the terms that match a given partial string, and then the list is used to get the frequencies of the terms that are obtained from the trie.

Table 1: Keystrokes Saved on User 1 Data

Receiver No	Baseline Implementation (a)	Receiver Specific Implementation (b)	Percentage Difference (c) = ((b-a)/a)*100%	Total No. of Characters Read (d)	Receiver Score (e) = (c*d) /100
1 English	236	267	13.14	4346	571.06
2 English	430	522	21.40	6139	1313.75
3 English	140	211	50.71	1926	976.67
4 English	182	255	40.11	3083	1236.59
5 English	535	671	25.42	5978	1519.61
6 Non-English	307	325	5.86	6245	365.96
7 Non-English	23	74	221.74	553	122.62
8 Non-English	63	73	15.87	1039	164.89
9 Non-English	118	220	86.44	2368	2046.90
10 Non-English	171	220	28.65	2700	773.55
				34377	9091.6
Average Percentage Difference for Keystrokes Saved				26.45	

Table 2: Keystrokes Saved on User 2 Data

Receiver No	Baseline Implementation (a)	Receiver Specific Implementation (b)	Percentage Difference (c) = ((b-a)/a)*100%	Total No. of Characters Read (d)	Receiver Score (e) = (c*d) /100
1 English	193	264	36.79	2972	1093.40
2 English	1251	1528	22.14	16335	3616.57
3 English	241	333	38.17	3286	1254.27
4 English	132	141	6.82	2424	165.32
5 English	219	335	52.97	2779	1472.04
6 Non-English	135	249	84.44	5784	4884.01
7 Non-English	211	247	17.06	4047	690.4182
8 Non-English	152	225	48.03	3558	1708.907
9 Non-English	575	570	-0.87	11314	-98.43
10 Non-English	73	129	76.71	2253	1728.28
				54754	16514.77
Average Percentage Difference for Keystrokes Saved				30.16	

Table 3: Keystrokes Saved on User 3 Data

Receiver No	Baseline Implementation (a)	Receiver Specific Implementation (b)	Percentage Difference (c) = ((b-a)/a)*100%	Total No. of Characters Read (d)	Receiver Score (e) = (c*d) /100
1 Both	401	499	24.44	8587	1093.40
2 Both	17	35	105.88	445	3616.57
3 Both	48	62	29.17	660	1254.27
4 Both	185	244	31.89	2558	165.32
5 Both	772	968	25.39	12473	1472.04
6 Both	500	639	27.80	6998	4884.01
				31721	8690.44
Average Percentage Difference for Keystrokes Saved				27.40	

Sai Pravallik Reddy Gujjula

1001 N Spring St, Apt E2
Middletown, PA 17057

gpravallikreddy@gmail.com
(717)-343-0725

Objective

To utilize my knowledge and skills in the field of computer science in order to achieve the prospective employer's goals, and to enhance my skills further in a dynamic workplace

Education

Pennsylvania State University – Capital College **Middletown, PA**
Bachelor of Science in Computer Science *(Expected 2018)*
Kennedy High the Global School, **Hyderabad, AP, India**
Central Board of Secondary Education *(2008-2012)*

Technical Skills

- **Languages:** C++, Java, Python, Scheme, Prolog, ML

Design and implementation of algorithms	Structured Programming	OOP with fundamental data structures

- **Related Coursework:** Data Structures, Algorithms, Operating Systems, Compilers, Databases, Computer Organization and Architecture, Artificial Intelligence, Software Engineering, Data Mining, , Principles of Programming Languages, Formal Languages, Secure Programming, Discrete Mathematics, Probability, Net-centric Computing
- **Operating Systems:** Windows, Unix

Employment Experience

- **Intern Undergraduate Associate Consultant, Dell Boomi** **May 2017 – Current**
  - Working on the Boomi Atomsphere Platform to help assist clients' in performing various integration tasks between any two endpoints
  - **External Project:** Service-Now and Open-Text end-to-end integration for Honeywell
  - **Internal Project:** LMS Absorb Training and Boomi Atomsphere Platform integration
- **Peer Tutor, Pennsylvania State University – Capital College** **August 2016 – Current**
  - Assisting fellow students in understanding their subject-related topics
  - Utilizing my subject knowledge in enhancing their problem solving skills
- **Lab Assistant, Pennsylvania State University – Capital College** **August 2015 – Current**
  - Assisted the professor in conducting laboratory experiments
- **Fire Rescue Officer, Singapore Civil Defense Force** **July 2013 – January 2015**
  - Showcased exemplary leadership qualities
  - Trained personnel to acquire proficient rescue operation skills

Certifications and Achievements

- **Schreyer's Honor Program** **Schreyer's Honors College, 2016 - Current**
- **Capital College Scholarship** **Pennsylvania State Harrisburg, 2016 - Current**
- **Dean's List & Capital Honors Program** **Pennsylvania State Harrisburg, 2015 -16**
- **Advanced Diploma in Rescue Operations & Incident Management** **Singapore Civil Defense Force, 2014**