**Group Members:**

|  |  |
| --- | --- |
| Koh Jia Yi |  |
| Zha Ming |  |
| Teh Suet Ling |  |

Table of Contents

[1 Background: 3](#_Toc139629260)

[1.1 Aim and Objective: 4](#_Toc139629261)

[2 Related Work 5](#_Toc139629262)

[2.1 Introduction of Candidate Techniques 5](#_Toc139629263)

[2.2 Python Libraries 6](#_Toc139629264)

[3 Edit Distance 9](#_Toc139629265)

[4 Formulation/Design 11](#_Toc139629266)

[5 Implementation 15](#_Toc139629267)

[5.1 Data Preprocessing 15](#_Toc139629268)

[5.2 Non-Word Error 15](#_Toc139629269)

[5.3 Real-Word Error 15](#_Toc139629270)

[5.4 Search Function 16](#_Toc139629271)

[6 Results 18](#_Toc139629272)

[7 References 30](#_Toc139629273)

# Background:

Typographic error is inevitable, it is common in every writing, whether due to the writer’s knowledge or from careless mistakes. The error types are known as non-word error and include:

a) Missing letter error, this error is common when the word consists of more than one same letter next to each other.

Correct:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| B | A | L | L | O | O | N |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| B | A | L | O | O | N |

Incorrect:

b) Extra letter error, this happens when an additional letter was entered.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| F | A | U | L | T | Y |

Correct:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| F | A | U | L | T | T | Y |

Incorrect:

c) Wrong Letter error

This type-error often occurs when the user is typing fast and ended entering the letter on its left or right on keyboard.

|  |  |  |  |
| --- | --- | --- | --- |
| W | O | R | D |

Correct:

|  |  |  |  |
| --- | --- | --- | --- |
| W | O | R | F |

Incorrect:

d) Transposition error

This occurs when the position of letters is accidentally swapped.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| L | E | T | T | E | R |

Correct:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| L | E | T | T | R | E |

Incorrect:

e) Extra/ unwanted spacing error

Another error that occurs often when the user type in fast speed and ended entering unnecessary spacing in word.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| J | O | K | I | N | G |

Correct:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| J | O | K | K | I | N | G |

Incorrect:

On the other hand, real-word error happens when the user has limited knowledge in the language and causing uses word with similar sound but different meaning. For instance:

Mail 🡪 Male

Main 🡪 Mane

Pray 🡪 Prey

The research in Natural Language Processing (NLP) has been growing rapidly in the past decades. Back in 1961, Les Earnest, a computer scientist acknowledged the need to implement a spell-checker amid his work in developing program to recognize bitmap images of cursive writing (Earnest, 2011). Relevant work was then carried out by Ralph Gorin, who developed SPELL, the first true spelling checker program, in 1971 (Earnest, 2011).

Spell-checking system can offer great support for a good writing piece. In general, spell-checker identifies misspelled word and match with the words in a dictionary with close edit distance. The use of n-gram model in language processing has been popular, it was first discussed in 1951 (Chaabi & Ataa Allah). N-gram modelling transforms unstructured data format to a structured format, it is defined as a neighboring sequences of words or tokens in a document. This technique is essential for text data feature extraction.

## Aim and Objective:

The aim of this project is to build a spell-checking system which can detect non-words and real words errors, then suggest correct word choice specifically in the computer science field.

Objectives:

1) To build a spell-checker which can detect non-word error using minimum edit distance technique.

2) To build a spell-checker which can detect real words errors using bigram technique.

# Related Work

## Introduction of Candidate Techniques

Spell-checker is essential not only to ensure accurate message is delivered but also information search. There are several existing systems and techniques for spell-checking. For instance, dictionary-based spell checkers in which this system compares words to a dictionary. The word is considered misspelled if it is not found in the dictionary. Rule-based spell checkers on the other hand uses predefined rules that relies on regular expressions such as repeated letters, missing vowels, or common misspellings (Wiechetek, 2021). Besides that, some spell-checker uses statistical language model to calculate the probability of a word or words sequence present in a context. Based on the model, it can suggest corrections based on probability. Some spell-checkers utilized machine learning technique such as neural networks which can learn from large datasets, predict the likelihood of correct spellings, and suggest corrections (Wiechetek, 2021). In addition to that, contextual spell checkers determine the error based on the analysis of surrounding words, grammar, and syntax.

Spell-checker system is utilized in search engines to ensure that the information retrieved is relevant to the user’s search. Google Suggest feature was first introduced by Google in 2008 to fine-tune queries and reduce errors on search engine. However, limited spell-checker is available for offline search tools as it is less popular. Gowri et al. (2022) proposed a spell check mechanism which converts the query entered by user into binary form. This information is then compared with dictionary which is stored in binary form. After the query evaluation process, any relevant documents retrieved and suggested list will be presented. This method was tested to be capable in increasing accuracy and reducing error of user’s query.

Besides English, spell-checker is also available in other languages. For instance, (Chaabi & Ataa Allah, 2022) developed a spell-checker system for the Amazigh language. In the work of Chaabi & Ataa Allah (2022), the combination of Damerau-Levenshtein algorithm and N-gram was used. As the name suggested, the system is capable in suggesting possible corrections for mispelled Amazigh word. Amazigh is one of the languages used in Africa. It uses Tifinagh-IRCAM alphabet system which consists of thirty-three letters (27 consonants, 2 semi-consonants and 4 vowels). The proposed system can handle large dictionaries, the target word occurs in the top five suggetstion in a frequent manner (87%) (Chaabi & Ataa Allah, 2022).

Targeting on Indonesian Language, Anggoro & Nurfadilah (2022) developed a system using Jaro-Winkler Distance (JWD) algoirthm to correct specific active verbs with the prefix mem- followed by letter ‘p’. Overall, the system achieved a 80%-95% accuracy for common, specific and correctable mistakes. The authors used JWD algorithm for its high Mean Average Precision (MAP) characteristic. In comparison to other string searching algorithm, such as Hamming Distance (MAP= 0.46), Levenshtein Distance (MAP= 0.74) and Damerau Levenshtein Distance (MAP=0.85) algorithms, JWD achieved an overall 0.87 MAP value (Anggoro & Nurfadilah, 2022). According to Anggoro & Nurfadilah (2022), JWD algorithm considers three components: string length calculation, search for two strings with similar characters, search for number of transpositions. Based on the output of JWD Calculations, autocorrect process will take place when the treshold 0.95 is achieved while suggestions will be provided if 0.92 threshold is achieved. The accuracy achieved by using JWD algorithm ranged between 80% to 95% (Anggoro & Nurfadilah, 2022).

On the other hand, Lu et al. (2019) developed a spell-checking tool, CSpell specifically for consumer health inquiries. CSpell is capable of handling different errors including nonword errors, real-word errors, word boundary infractions, punctuation errors, and any of these combinations (Lu et al., 2019). To develop this tool, the authors used dual embedding model to tackle context-dependent problems as well as the combination of dual embedding technique and dictionary-based corrections for a 2-stage ranking system (Lu et al., 2019). Besides that, splitters and handlers were built to handle word boundary infractions problem (Lu et al.,, 2019). As a result, in comparison to the Ensemble method which was chosen by the author as a strong baseline, CSpell outperforms by 14.03% in detecting error and 12.33% in correcting error.

## Python Libraries

Python language was used to build the spell-checker system. Various libaries or packages in Python are available Natural Language Processing (NLP). The summary of relevant libraries used to develop the system are as shown in Table 1.

|  |  |  |
| --- | --- | --- |
| **Library** | **Introduction** | **Functions Used to Build the Spell-Checker System** |
| nltk (Natural Language Toolkit)  (NLTK Documentation, 2023) | This is a very popular library for natural language processing that provides many tools for dealing with text data. | Tokenization: The nltk.tokenize.word\_tokenize method is used to split string text into a list of words  Bigram generation: The nltk.bigrams method is used to generate adjacent pairs (bigrams) of words, which is useful for analyzing the relationship between two consecutive words.  Corpus: nltk.corpus.brown, this specific corpus is a collection of the Brown University Standard Corpus of Present-Day American English, providing a large number of English words helpful for word verification and suggestions. |
| tkinter  (Python Interface to Tcl/Tk, 2007) | tkinter is Python's standard graphical user interface library. | tkinter and its ttk module are used for creating and managing the user interface, such as creating text input fields, buttons, labels, etc. All graphical elements (like text fields and buttons) are defined and controlled through tkinter. |
| re (Regular expression operations)  (Regular Expression Operations, n.d.) | The re module provides support for regular expressions, which are powerful string processing tools capable of complex pattern matching and manipulation. | In this project, the re module is mainly used for recognizing and extracting words from the input text. |
| levenshtein  (python-Levenshtein 0.21.1, n.d.) | This library provides a function to calculate the Levenshtein distance between two strings. The Levenshtein distance measures the minimum number of single-character edits (insertions, deletions, or substitutions) required to change one string into another. | In this project, it's used to calculate the edit distance between the wrong word and words in the dictionary for providing correction suggestions. |
| os  (Miscellaneous Operating System Interfaces, n.d.) | The os library provides many functions for interacting with the operating system, such as file and directory operations. | In this project, the os module is mainly used for dealing with the file paths of the corpus. |
| concurrent.futures  (concurrent.futures- Launching Parallel Tasks, n.d.) | This module provides a high-level interface for asynchronously executing callables (i.e., functions, methods). It allows tasks to be scheduled concurrently using threads or processes in order to efficiently use system resources and improve overall program performance. | In this project, concurrent.futures is used in the get\_min\_edit\_dist\_words function where it parallelizes the calculation of edit distances between the wrong word and all words in the dictionary. By using ThreadPoolExecutor from concurrent.futures, the program can speed up the distance calculation process significantly. Each thread in the pool is assigned a word and calculates the edit distance to the wrong word independently and concurrently with the other threads. |

Table 1 Library Used to Build the Spell-Checker System

# Edit Distance

In the transformation from string W1 into string W2, multiple operations can take place including letter insertion, deletion, transposition, and substitution. Considering the operations to be taken, the lexicographical distance calculates the minimum number of edits required for the transformation to happen. Edit distance, also known as Levenshtein distance is defined as the minimum number of operations required to transform a string (Yulianto et al., 2018). It considers three string operations: substitution, insertion, and deletion. In addition to that, other variations of edit distance consider different string operations can contribute to the development of a spell-checker system. For example, Damerau-Levenshtein distance is another string metric which measure the difference between two strings (Zhao & Sahni, 2017). In addition to substitution, insertion and deletion, it also considers transposition of two adjacent characters. On the other hand, Restricted Damerau-Levenshtein distance, as its name suggests, it covers similar operations as the former variation with the exception that each substring is restricted to only one edit (Chaabi & Ataa Allah,, 2022).

In contrary, Hamming distance measures the similarity between two sequences. Specifically, for two strings with the same length, the hamming distance is referred to as the number of characters between the two strings which are different (Sharma, 2020). The drawback of Hamming distance is that it can only be applied to strings with same length.

Longest common subsequence (LCS) looks for the longest common subsequence that present in the given two strings in the same order, in which subsequence does not require that the characters to be consecutive in both strings (Paterson, 1994). Conversely, approximate substring matching is a fuzzy string searching technique that looks for the closest match between strings (Navarro, 2014).

|  |  |
| --- | --- |
| **Edit Distance Variations** | **Function** |
| Damerau-Levenshtein Distance  (Zhao & Sahni, 2017) | Measure distance between two strings. Consider insertion, deletion, substitution, and transposition. |
| Restricted Damerau-Levenshtein Distance  (Chaabi & Ataa Allah,, 2022) | Restrict each substring to only one edit. Consider insertion, deletion, substitution, and transposition. |
| Hamming distance  (Sharma, 2020) | Measure the similarity between two sequences. Only applicable to strings with same length. |
| Longest common subsequence  (Paterson, 1994) | Looks for the longest common subsequence that present in the given two strings in the same order |
| Approximate substring matching  (Navarro, 2014) | Fuzzy string searching technique, looks for the closest match between strings. |

Table 2 Edit Distance Variations and Function

# Formulation/Design

The foundation of the spelling correction system relies heavily on the principle of Natural Language Processing (NLP) and the concept of edit distance and bigram frequency. The primary objective is to design a system capable of accurately identifying both non-word and real word spelling errors, further suggesting appropriate corrections.

The fundamental design of the system is based on a dictionary formed from two distinct corpuses. One corpus comprises computer science texts while the other encompasses a general English lexicon. A dictionary is constructed from these corpuses, which then serves as the primary reference to detect spelling errors. Furthermore, the design employs the concept of tokenization and normalization to preprocess the data effectively.

This program's GUI portion was thoughtfully crafted, focusing on various integral elements such as main window configuration, text input box formulation, suggestion display setup, search bar creation, and error identification design. Here's a more detailed explanation of each part's design rationale.

**Main Window Configuration**

The main window is initially developed using the tkinter library, within which different components are systematically arranged. The main window, labeled "Word Checking GUI", contains a main frame that hosts all other components. Its grid, consisting of two rows and two columns, is configured to facilitate an organized layout.

**Text Input Box Formulation**

An interactive text box is integrated into the main window to facilitate user input, erroneous word identification, and word suggestion. The first row and column of the main window house this feature, where we've incorporated a label, "Input Sentence:", and a Text widget for user input. To distinguish between error types, 'error' and 'potential\_error' tags are assigned, coloring the background red and yellow respectively. Two events, <KeyRelease-space> and <Button-1>, are tied to the widget, enabling real-time error detection and suggestion display.

**Suggestion Display Setup**

In the first row's second column, another frame is constructed to present suggested words. A Text widget in this frame, labeled "Suggested Words:", displays the suggestions, and is bound to the <Button-1> event, allowing users to replace errors by clicking on a suggested word.

**Search Bar Creation**

The second row's second column contains a search bar where users can look up desired words. Along with a label, "Search Words:", an Entry widget allows for user input, and a Text widget presents the search results. The Entry widget's <KeyRelease> event immediately executes a search upon character input.

**Error Identification Design**

The mark\_words function, called when a word is entered, checks for errors. It first calls the mark\_unknown\_words function to verify the word's presence in the dictionary, tagging unrecognized words as 'error'. It also calls the mark\_real\_words\_error function, adding a 'potential\_error' tag to words that are contextually incorrect. Concurrently, the update\_suggested\_words function refreshes the suggestion list.

**Adaptive Window Size with GRID System**

The GRID system is instrumental in our GUI design for component layout, providing a versatile grid structure to place components as required. A key feature of the GRID system is its adaptive nature, allowing for changes in grid and component sizes in response to window size variations.

The function part of the program involves key elements such as corpus reading, error marking, word searching and suggestion, non-word and real-word handling, and interactive features like corpus word display and highlighting misspelled words. The design approach for these aspects is explained as follows:

**Corpus Reading (read\_corpus):**

This function navigates all files within a specific directory, reading each file's content. The contents are tokenized, standardized (lowercased), and added to the word dictionary, along with creating bigrams for addition to the bigram list. It finally includes words from the Brown corpus to the word dictionary and bigram list. The returned result includes a set with all words and a conditional frequency distribution containing all bigrams.

**Error Marking (mark\_unknown\_words, mark\_real\_words\_error):**

The mark\_unknown\_words function identifies words not present in the dictionary as errors (red background), while the mark\_real\_words\_error function flags infrequently occurring bigrams as potential errors (yellow background).

**Word Search and Suggestion (search\_words, suggest\_words):**

The search\_words function explores the dictionary for words that match the initial letters inputted in the search bar, displaying the results. The suggest\_words function proposes similar dictionary words, differing by no more than three characters in length from the misspelled word and with a top-five edit distance. The suggestions are sorted by edit distance and length, prioritizing edit distance but favoring longer words if distances are equal.

**Non-words and Real-words Handling:**

Minimum Edit Distance and Bigram techniques detect and correct spelling errors, primarily identifying and amending non-existent corpus words (non-words). The mark\_unknown\_words method is invoked when a user inputs text, marking unrecognized words as errors. By clicking on an erroneous word, the show\_suggestions method provides spelling suggestions, calculated by minimum edit distance. Real-word error detection forms another integral part of the system design. For this, the concept of bigram frequency is utilized. The design uses the frequency of word pairs (bigrams) in the corpus to detect potential real-word errors. A word is flagged as a possible error if it forms a zero-frequency bigram with its preceding or following word. The system then suggests corrections based on the highest frequency bigrams, allowing the design to account for the context of the error.

**Interactive Features:**

In order to enhance user experience, the system incorporates a word search feature. A search bar has been included, which allows users to query words based on their initial letters. Upon any keystroke in the search bar, the search\_words function is triggered, generating a dynamically updated, alphabetically sorted list of corpus words that begin with the inputted letters. As the user types, the list of results is continuously refined in real time to match the user's query, providing a seamless and user-friendly browsing experience.

A screenshot of a computer

Description automatically generated

Word Searching Frame

Input Frame

Suggested Word Frame

Figure . Screenshot of the layout of the system

The operational workflow of the system can be represented by the flowchart outlined below:

A diagram of a work flow

Description automatically generated

Figure . Flowchart of the process of non-word & real-word error detection system.

# Implementation

## Data Preprocessing

The system initiates its operation by loading two distinct corpuses. The first corpus is drawn from the computer science domain, incorporating text from ten selected research papers. The second corpus, a generalized English lexicon, is derived from the widely used Brown Corpus provided by the Natural Language Toolkit (NLTK). Both corpuses are subject to an initial tokenization process using the 'NLTK.tokenize' function, succeeded by a normalization procedure that removes any non-alphabetical characters. The subsequent unique words are then stored within a dictionary. Concurrently, the ‘NLTK.bigram' function is utilized to generate bigrams, and the ‘ConditionalFreqDist’ is used to track the frequency of their occurrence within the corpus.

## Non-Word Error

This system is specifically designed to detect both non-word and real-word errors. For non-word errors, the system cross-references the input word with the established dictionary. If a word cannot be found within this set, it is highlighted in red, flagging a potential spelling mistake.

The user is then prompted to select the highlighted word, which triggers the system to suggest five alternative words displayed in the top right frame. This selection process is efficient and tailored, as it only calculates the edit distance (utilizing Levenshtein Library) for words with a letter count that deviates by a maximum of ±3 from the misspelled word, thereby reducing unnecessary computational time. For instance, if the misspelled word is “appl” (consisting of four letters), the candidates will include words with a length between 1 and 7 letters. The system then computes the edit distance between the misspelled word and the candidates, presenting the top five with the smallest edit distance. The user can replace the misspelled word by clicking on the preferred suggestion. An illustration is shown in the diagram below:

A screenshot of a computer

Description automatically generated

A typo of the word essay

Suggestion after clicking on the highlighted word.

Figure . Demonstration of spelling error/non-word detection, and the suggestions for it. The detected error ‘eesay’ is suggested with the word (ranked from lowest edited distance) ‘essay’, ‘say’, ‘delay’, ‘repay’, ‘desal’.

## Real-Word Error

In the case of real-word errors, the system employs bigram frequency. A potential real-word error is flagged if the frequency of a particular bigram within the input sentence is zero. To illustrate, consider the sentence "I want too write an essay". The bigrams ('want', 'too') and ('too', 'write') register a frequency of zero, indicating a possible real-word error. However, the bigrams ('I', 'want') and ('write', 'an') yield a frequency greater than zero, marking them as valid bigrams. As a result, only the word 'too' is highlighted in yellow as a potential real-word error. This intricate process ensures that the system is sensitive to both non-word and real-word errors, facilitating an efficient spelling correction tool for users.

A yellow and black text

Description automatically generated

In the event of such real-word errors, the system offers a mechanism similar to non-word errors to provide potential corrections. Once a user clicks on the highlighted text, the system generates suggestions based on the preceding word. For instance, with 'too' being identified as an error, the word before it, 'want', becomes the basis for candidate generation. The system formulates suggestions according to the top 5 bigrams exhibiting the highest frequency where the first word is 'want' (‘want’, ‘\*next word\*’). Upon selection, the original erroneous word is replaced by the user-selected suggestion.

A screenshot of a computer

Description automatically generated

Suggestion after clicking on the highlighted word.

A typo of the word essay

Figure . Demonstration of the real-word error detection, and the suggestions for the error. The detected error is the word ‘too’, and the suggestions are (ranked from high to low): ‘to’, ‘a’, ‘you’, ‘it’, ‘the’.

## Search Function

Moreover, the system also offers a word searching tool that allows users to search for a particular word in the dictionary. The "search words" functionality is a critical part of the GUI application that assists users in identifying words in the corpus that match or start with the text inputted in the "Search Words" text box.

1. GUI Element Creation: The create\_search\_frame function is called to create a search frame on the right side of the GUI interface. This frame consists of a label "Search Words", an entry box where the user can type in the search term, and a text box where the matching words from the corpus will be displayed.

2. Search Term Input: As the user types into the entry box, the search\_words function is invoked upon every key release event. This function fetches the current search term typed into the entry box.

3. Word Search: The function then iterates over the entire corpus (self.dictionary) to find all the words that start with the current search term. It uses the startswith function of Python's string class to check if a word in the dictionary starts with the search term.

4. Displaying Results: The matching words are sorted in alphabetical order and displayed in the text box below the entry box. If a previously displayed list of words exists, it gets cleared out before displaying the new results. This process ensures that the search results always correspond to the latest search term typed into the entry box.

This feature improves the usability of the application, making it easy for users to find words in the corpus that align with their search requirements. As a real-time feature, it provides instant feedback to users, enhancing the interactive aspect of the application.

A screenshot of a computer

Description automatically generated

Figure . Demonstration of the word-searching functionality.

# Results

The following test cases showed that the proposed spell-checking system is capable of detecting non-words and real-words errors. Non-words error which is due to typing error is highlighted as red while real-words error which happens due to wrong context is highlighted as yellow. For each error, the system will provide a list of suggested words based on edit distance (for non-words error) and real-words error is based on the bigram frequency.

**Case 1:**

A yellow and black text

Description automatically generated

**Non-words error:**

A screenshot of a computer

Description automatically generated

**Real-words error:**

A close up of a computer screen

Description automatically generated

A screenshot of a computer

Description automatically generated

In this example, ‘pythone’ was correctly detected by the system as non-word error and provided the correct words suggestion. Meanwhile, the system also detected ‘programming’ as the real-word error due to the low bigram frequency of ‘level programming’ and ‘programming language’ in the corpus.

**Case 2:**

A screenshot of a computer

Description automatically generated

**Non-words error:**

A screenshot of a computer

Description automatically generated

In this example, the system is able to capture the only non-words spelling error and the expected word ‘hidden’ was also suggested by the system for correction. However, the phrase ‘node layers’ was missed.

**Case 3:**

A screenshot of a computer

Description automatically generated

**Non-words error:**

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**Real-words error:**



A screen shot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Two non-words error were correctly detected by the system with correct words suggestion. However, due to the low bigram frequency of ‘machine learning’ and ‘intelligence which’, the system detected ‘learning’ and ‘which’ as real-words error.

**Case 4:**

A screenshot of a computer

Description automatically generated

**Non-words error:**

A screenshot of a computer

Description automatically generated

A computer screen shot of a error

Description automatically generated

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

**Real-words error:**

A close up of a text

Description automatically generated

A white rectangular object with a black border

Description automatically generated

A close up of text

Description automatically generated

A white rectangular object with a black border

Description automatically generated

A white rectangular object with a black border

Description automatically generated

In this sentence, one non-word spelling error was detected by the system and the expected word ‘accuracy’ was provided as the first suggested word. ‘Error function, if’ was detected as a real-word spelling error as these phrases are not found in the corpus. Meanwhile, the system also correctly detected ‘too access’ as real-word error and the correct word was among the suggested words shown.

**Case 5:**

A screenshot of a computer

Description automatically generated

**Non-words error:**

A white rectangular object with a black border

Description automatically generated

**Real-words error:**

A close up of a white background

Description automatically generated

A screen shot of a computer

Description automatically generated

A computer screen shot of a computer

Description automatically generated

A black text on a white background

Description automatically generated

A screenshot of a computer

Description automatically generated

In this sentence, one non-word spelling error was detected by the system and the expected word ‘focuses’ was among the suggested words. Meanwhile, the system also correctly detected ‘it involve’ as real-word error in addition to other two phrases which was mistakenly detected due to the low appearance of these phrases in the corpus.

**Case 6:**

A screenshot of a computer

Description automatically generated

**Non-words error:**

A white rectangular object with a black border

Description automatically generated

**Real-words error:**



A white rectangular object with a black border

Description automatically generatedA white rectangular object with a blue background

Description automatically generated

In this sentence, an individual’s name was detected as non-word spelling error. Besides that, the system has correctly detected real-word spelling error ‘referred too as’ and suggested the expected correct word ‘to’.

**Case 7:**

A computer screen shot of words

Description automatically generated

**Non-words error:**

A white rectangular object with black text

Description automatically generated

A screenshot of a computer

Description automatically generated

**Real-words error:**



A white rectangular object with a black border

Description automatically generated

A close up of text

Description automatically generated

A white rectangular object with a black border

Description automatically generated

Two non-words error were correctly detected by the system with correct words suggestion. Meanwhile, the system detected mistakenly three real-words error due to the low bigram frequency.

**Case 8:**

A screen shot of a computer

Description automatically generated

**Non-word error:**

A white rectangular object with a blue background

Description automatically generated

**Real-word error:**



A white rectangular object with a black border

Description automatically generated

A close up of a text

Description automatically generated

A screen shot of a computer

Description automatically generated

A white screen with black text

Description automatically generated

A white rectangular object with a blue background

Description automatically generated

In this example, one non-word error was correctly detected with expected correct word provided. However, due to the low bigram frequency of ‘correction schemes, systemic or non-systemic’, these phrases were also detected as real-words error.

A summary of the system’s performance is detailed as below:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Case** | **Non-word Errors** | **Detected Non-word Errors** | **Non-word error suggestions** | **Real-word Errors** | **Detected Real-word Errors** | **Correct Sentence** |
| 1 | pythone | pythone | Python | (none) | programming | Python is a high level programming language, it can be used to build web application. |
| 2 | hiden | hiden | hidden | note layers | (Not detected) | Neural networks are comprised of node layers, containing an input layer, one or more hidden layers, and an output layer. |
| 3 | brunch | brunch | branch | (none) | learning, which | Machine learning is a branch of artificial intelligence which focuses on the use of data and algorithms to imitate the way humans learn. |
| 4 | ccuracy | ccuracy | accuracy | too | error function, if, too, access | An error function evaluates the prediction of the model. If there are known examples, it can make a comparison to assess the accuracy of the model. |
| 5 | focusses | focusses | focuses | it involve, | it involve, mathematical, and | Computer science focuses on the development of software systems. It involves working with mathematical models, data analysis, security, algorithms, and computational theory. |
| 6 | Babage | Babage | cabbage | too | referred too as | Babbage is sometimes referred to as the father of computing. |
| 7 | vission | vission | vision | (none) | computer, image | Computer vision is a field of computer science that focuses on enabling computers to identify objects and people in images and videos. |
| 8 | corection | corection | correction | (none) | systemic or non-systemic | Error detection and correction schemes can be either systematic or non-systematic. |

Table . Summary of the non-words error, detected non-words error, suggested words for non-words error, real-words error, detected real-words error, and the correct sentence.

Non-word error detection accuracy =

Real word error detection accuracy =

**Results:**

Non-word error detection accuracy = 100%

Real word error detection accuracy = 32.5%

**Search Words**

A screenshot of a computer

Description automatically generated

**A screenshot of a computer

Description automatically generated**

Figure 6 Search Words Function

Besides error detection, this system is embedded with a search words function. The user can pass in one letter and a list of possible words will be displayed. As more letters are entered, the displayed words become more specific. In this example, when the word ‘learn’ was entered in the search bar, all possible words derived from the root word ‘learn’ are displayed.

Overall, the system exhibits great proficiency in detecting non-word errors. It is capable in detecting spelling inconsistencies and providing corrections, provided the correct form of the word resides in the corpus-derived dictionary. This validates its robust design, illustrating a solid understanding of the task and its efficient execution, in the field of non-word error detection.

However, the system's current capabilities for real-word error detection are limited. This limitation is due to its heavy reliance on the corpus used. The corpus' size and variety directly impact the accuracy of real-word error detection since it's fundamentally based on the frequency of word pair (bigram) occurrences. In the current state, the corpus appears to lack sufficient diversity and volume (which consists of 41733 unique words, 1095516 bi-gram) thereby inhibiting the system's ability to accurately determine real-word errors based on context.

Looking forward, there are several potential pathways for improvement. To begin with, expanding the size and diversity of the training corpus would improve the system's understanding of word usage in various contexts, hence enhancing its real-word error detection capabilities. Implementing more complex language models such as trigrams or n-grams could also be beneficial, as they consider larger context and can be more accurate in detecting errors based on context.

Moreover, incorporating a more sophisticated Natural Language Processing (NLP) model, such as transformer-based models like BERT or GPT, could boost the system's contextual understanding. These models are designed to understand the semantics of text and can better detect if a word is misplaced or used inappropriately.

Additionally, including a feedback mechanism for users to correct suggestions can help the system learn from its mistakes and improve over time. This self-learning ability can greatly enhance the system's adaptability and accuracy in the long run.

In conclusion, while the system is proficient in non-word error detection, its ability to detect real-word errors could be significantly improved through the mentioned strategies. With careful planning and implementation, the system's overall performance and utility can be significantly enhanced.

# References

Anggoro, D.A. & Nurfadilah, I. (2022). Active Verb Spell Checking Mem- + P in Indonesian Language Using Jaro-Winkler Distance Algorithm. *Iraqi Journal of Science, Vol 63 No. 4*, 1811-1822.

Bryer, E., Rhujittawiwat, T., Rose, J.R., & Wilder, C.F. (2021). Spelling Based Ranked Clustering Algorithm To Clean and Normalize Early Modern European Book Titles. *International Conferences Computer Graphics, Visualization, Computer Vision and Image Processing.*

Chaabi, Y. & Ataa Allah, F. (2022). Amazigh Spell Checker using Damerau-Levenshtein Algorithm and N-gram. *Journal of King Saud University- Computer and Information Sciences 34*, 6116-6124.

*concurrent.futures- Launching Parallel Tasks*. (n.d.). Retrieved from Python Software Foundation: https://docs.python.org/3/library/concurrent.futures.html

Earnest, L. (2011). The First Three Spelling Checkers.

Gowri, S., Sathish Kumar, P.J., Geetha Rani, K., Surendran, R., & Jabez, J. (2022). Usage of a Binary Integrated Spell Check Algorithm for an Upgraded Search Engine Optimization. *Measurement: Sensors 24*, 100451.

Lu, C.J., Aronson, A.R., Shooshan, S.E., & Demner-Fushman, D. (2019). Spell Checker for Consumer Language (CSpell). *Journal of the American Medical Informatics Association 26(3)*, 211-218.

*Miscellaneous Operating System Interfaces*. (n.d.). Retrieved from Python Software Foundation: https://docs.python.org/3/library/os.html

Navarro, G. (2014). Approximate String Matching. *Encyclopedia of Algorithms*, DOI 10.1007/978-3-642-27848-8\_363-2.

*NLTK Documentation*. (2 Jan, 2023). Retrieved from NLTK: https://www.nltk.org/

Paterson, M. & Dancik, V. (1994). Longest Common Subsequences. *International Symposium on Mathematical Foundations of Computer Science*, (pp. 127-142).

*Python Interface to Tcl/Tk*. (2007). Retrieved from Python Software Foundation: https://docs.python.org/3/library/tkinter.html

*python-Levenshtein 0.21.1*. (n.d.). Retrieved from PyPI: https://pypi.org/project/python-Levenshtein/

*Regular Expression Operations*. (n.d.). Retrieved from Python Software Foundation: https://docs.python.org/3/library/re.html

Sharma, P. (25 February, 2020). *Understanding Distance Metrics Used in Machine Learning*. Retrieved from Analytics Vidhya: https://www.analyticsvidhya.com/blog/2020/02/4-types-of-distance-metrics-in-machine-learning/#:~:text=Hamming%20Distance%20measures%20the%20similarity,the%20corresponding%20characters%20are%20different.

Wiechetek, L., Pirinen, F.A., Hamalainen, M., Argese, C. (2021). Rules Ruling Neural Networks- Neural vs. Rule-Based Grammar Checking for a Low Resource Language. *Processings of Recent Advances in Natural Language Processing*, 1526-1535.

Yulianto, M. M., Arifudin, R. & Alamsyah. (2018). Autocomplete and Spell Checking Levenshtein Distance Algorithm to Getting Text Suggest Error Data Searching in LIbrary. *Scientific Journal of Informatics Vol. 5, No.1*.

Zhao, C. & Sahni, S. (2017). String Correction Using the Damerau-Levenshtein Distance. *7th IEEE Internatioal COnference on Computational Advances in Bio and Medical Sciences (ICCAB 2017)* (p. 277). BMC Bioinformatics .