Implementation

1.1 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.

1.2 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:

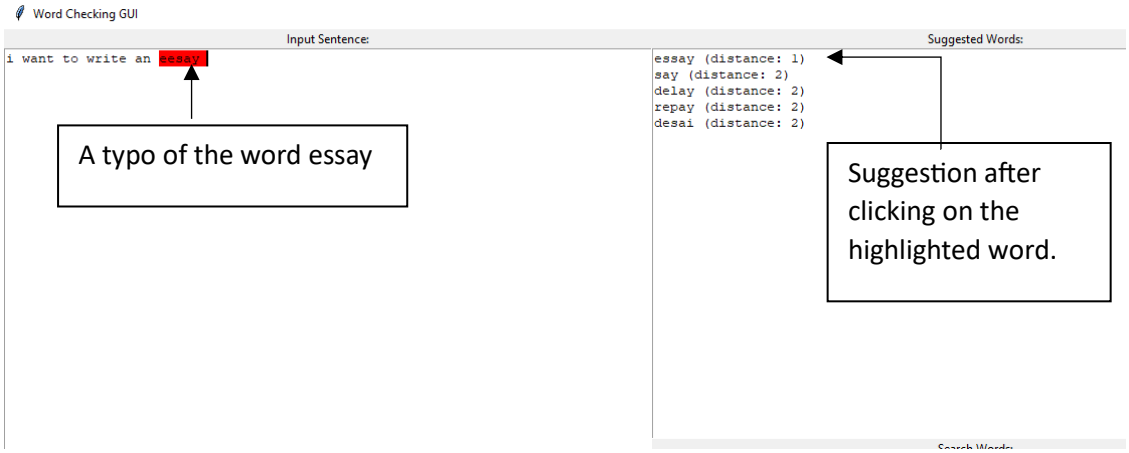
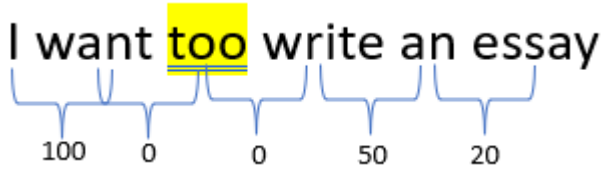


Figure 1. Demonstration of spelling error/non-word detection, and the suggestions for it. The detected error 'eesay' is suggested with the word (ranked from lowest edited distance) 'essay', 'say', 'delay', 'repay', 'desal'.

1.3 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.



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.

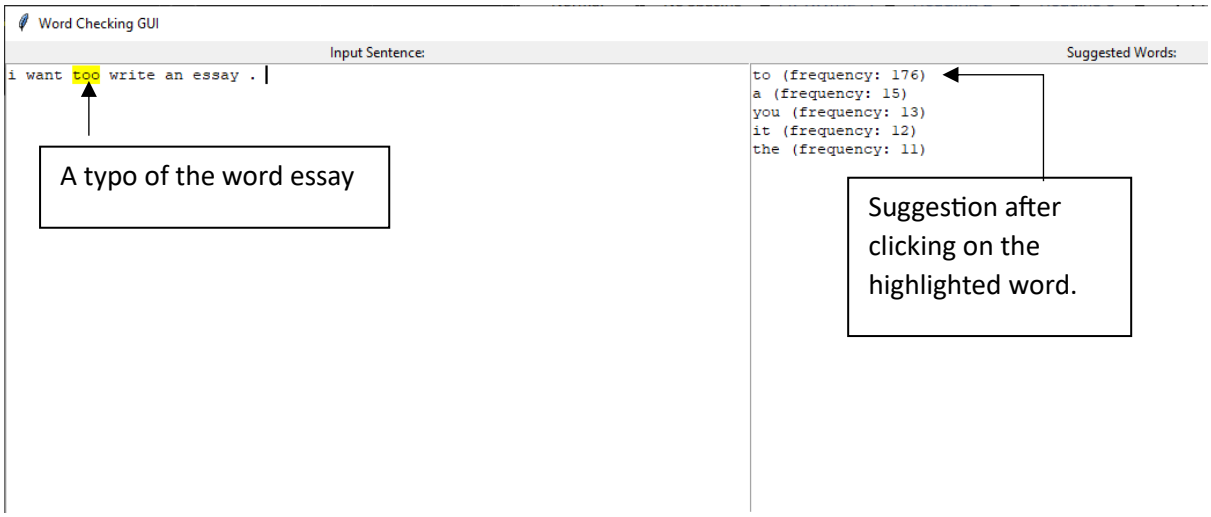


Figure 2. Demonstration of the real-word error detection, and the suggestions for the error. The detected error is the word ‘too’, and the suggestions are (ranked from high to low): ‘to’, ‘a’, ‘you’, ‘it’, ‘the’.

1.4 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.

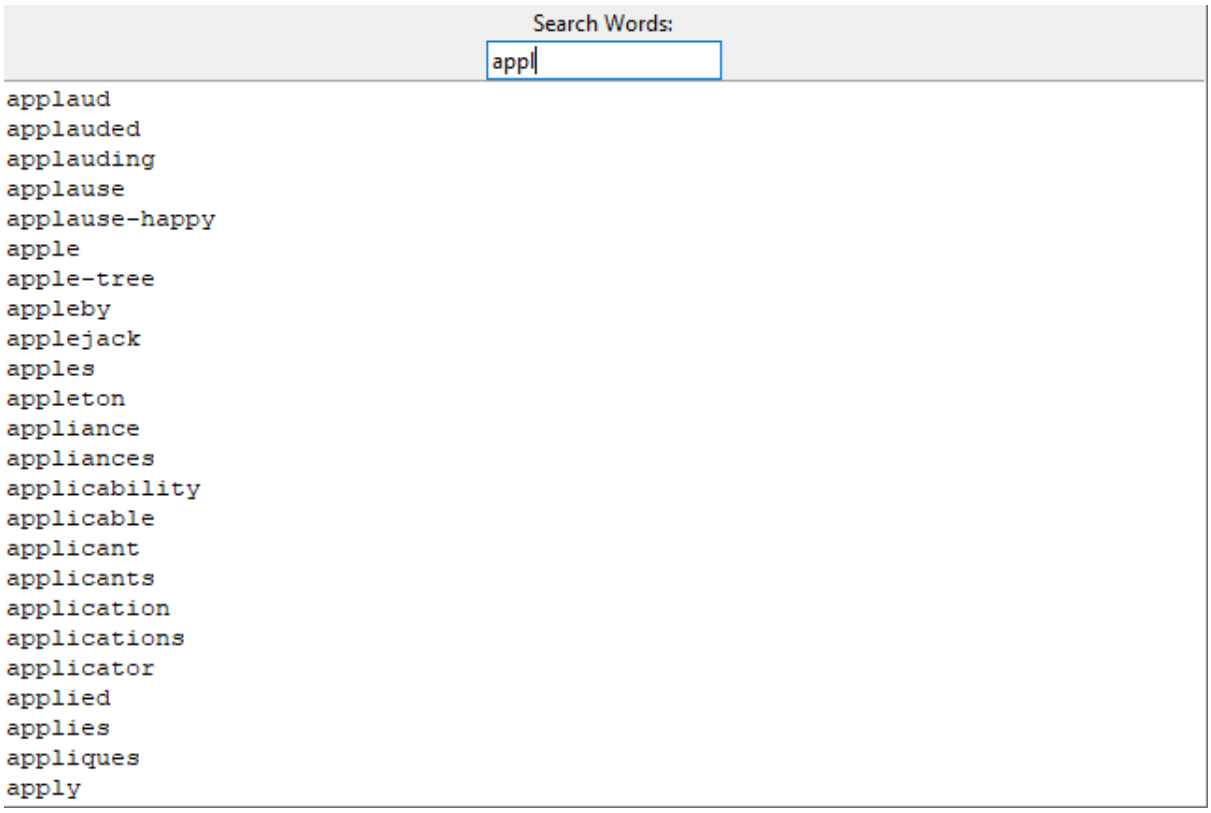


Figure 3. Demonstration of the word-searching functionality.