

ABSTRACT

Natural Language Processing (NLP) is a subfield of artificial intelligence (AI) focused on enabling computers to understand, interpret, and generate human language. It combines linguistics, machine learning, and computational models to allow machines to process text data in a manner that simulates human understanding. The increasing demand for error-free and contextually accurate written communication has led to significant advancements in Natural Language Processing technologies. This report develops a Context-Aware Autocorrector that leverages NLP techniques, specifically through the Natural Language Toolkit (NLTK), to improve both spelling and grammatical accuracy in user-generated text. NLTK, a powerful library in Python for text processing, plays a central role in the system by providing tools for tokenization, word validation, and manipulation of text data. The system's ability to identify misspelled words is powered by NLTK's word corpus, which contains a comprehensive list of valid English words.

The system employs rule-based spelling correction methods, such as letter deletion, switching, replacement, and insertion, to suggest possible corrections for misspelled words. NLP techniques enable the identification of these errors by breaking down text into individual tokens, allowing for more precise spelling corrections. For grammatical issues, a Hugging Face transformer model is used to apply advanced machine learning-based grammar correction, which understands not just the structure but also the context of the sentences. This ensures that grammatical fixes align with the overall meaning of the text, enhancing the accuracy of the output.

The project utilizes NLP's power to bridge the gap between traditional spelling correction algorithms and context-aware grammar correction, making it suitable for various applications such as education, content creation, and professional writing. With a user-friendly Tkinter interface, users can easily input text, view corrected sentences, and receive suggestions for misspelled words, all of which are powered by NLP's ability to process and understand natural language. The combination of NLP, NLTK, and modern transformer models offers a comprehensive and intelligent tool to improve text quality, making it more accurate, coherent, and contextually appropriate.

Table of Contents

Chapter 1.....	3
Introduction	3
Chapter 2.....	5
Methodology.....	5
2.1 Overview of System Architecture	5
2.2 Core Methodology	5
2.3 Algorithms and Techniques	6
2.4 Technologies and Libraries	7
2.5 Other Libraries.....	7
Chapter 3.....	9
Implementation.....	9
3.1 Preloading NLTK data and Model.....	9
3.2 Grammar Correction Model Initialization.....	9
3.3 Spelling Correction Methods	11
3.4 Text Correction Workflow	14
3.5 GUI Setup	15
Chapter 4.....	19
Results	19
Chapter 5.....	26
Conclusion	26
References.....	27

Chapter 1

Introduction

In the digital era, written communication plays a crucial role across professional, academic, and personal domains. Errors in spelling or grammar can hinder effective communication and compromise the quality of written content. Addressing these challenges, this project introduces the Context-Aware Autocorrector, an advanced system designed to enhance the accuracy and fluency of written text by combining traditional spelling correction techniques with cutting-edge Natural Language Processing (NLP) and machine learning models.

The Context-Aware Autocorrector is built to identify and correct both spelling and grammatical errors while considering the contextual meaning of sentences. At its core, the system leverages the Natural Language Toolkit (NLTK), a comprehensive Python library for linguistic analysis and text processing. NLTK facilitates tokenization, word validation, and spelling correction through rule-based algorithms such as letter deletion, switching, replacement, and insertion. This ensures precise identification and correction of misspelled words.

To handle grammatical corrections and sentence structure improvements, the system incorporates a transformer-based model from Hugging Face, which understands the overall context of a sentence. This model provides intelligent corrections for complex grammatical issues, including subject-verb agreement and phrase reordering, ensuring that the corrected text is both grammatically accurate and contextually coherent. The transformer model operates on the principle of attention mechanisms, allowing it to focus on relevant parts of a sentence while generating corrections. By understanding the relationships between words and phrases, the model ensures that the suggested corrections align with the intended meaning of the text. This approach is far more sophisticated than traditional rule-based grammar correction, as it considers nuances such as word order, syntactic dependencies, and contextual relationships.

developed using Tkinter, serves as the bridge between users and the advanced functionalities of the Context-Aware Autocorrector system. Tkinter, being Python's standard GUI toolkit, provides a simple yet versatile framework to design and implement intuitive desktop applications. The graphical interface includes interactive text input fields, buttons for autocorrect and clearing text, and sections to display both the corrected output and spelling suggestions. These features are organized within a clean and responsive layout, ensuring that the user experience remains smooth and engaging. The design prioritizes accessibility, allowing users from all technical backgrounds to harness the system's powerful text correction capabilities without needing specialized skills. Tkinter's cross-platform nature and ease of integration make it the perfect choice for creating a standalone application that combines functionality and simplicity, catering to a wide audience. This design ensures that users can access powerful text correction capabilities without requiring extensive technical knowledge.

By integrating advanced NLP techniques and an intuitive interface, the Context-Aware Autocorrector offers a holistic solution for improving text quality, making it a valuable tool for professionals, students, and anyone aiming to enhance their written communication.

System Workflow Diagram

The high-level system workflow of the Context-Aware Autocorrector demonstrates the sequential process by which input text is analyzed and corrected. The process begins with Input Text, where users provide raw text data through the user interface. This text is then passed to the Preprocessing stage which includes some critical steps which we will discuss later in the report in detail. The next step, Spelling Correction, applies advanced algorithms. Once spelling errors are addressed, the corrected text undergoes Grammar Correction, powered by a transformer-based NLP model that ensures context-aware fixes for grammatical issues. Finally, the corrected and polished text is presented as the Output, ready for use. This workflow, depicted in a structured diagram, highlights the seamless integration of rule-based and machine-learning-driven techniques to deliver accurate and contextually relevant text corrections.

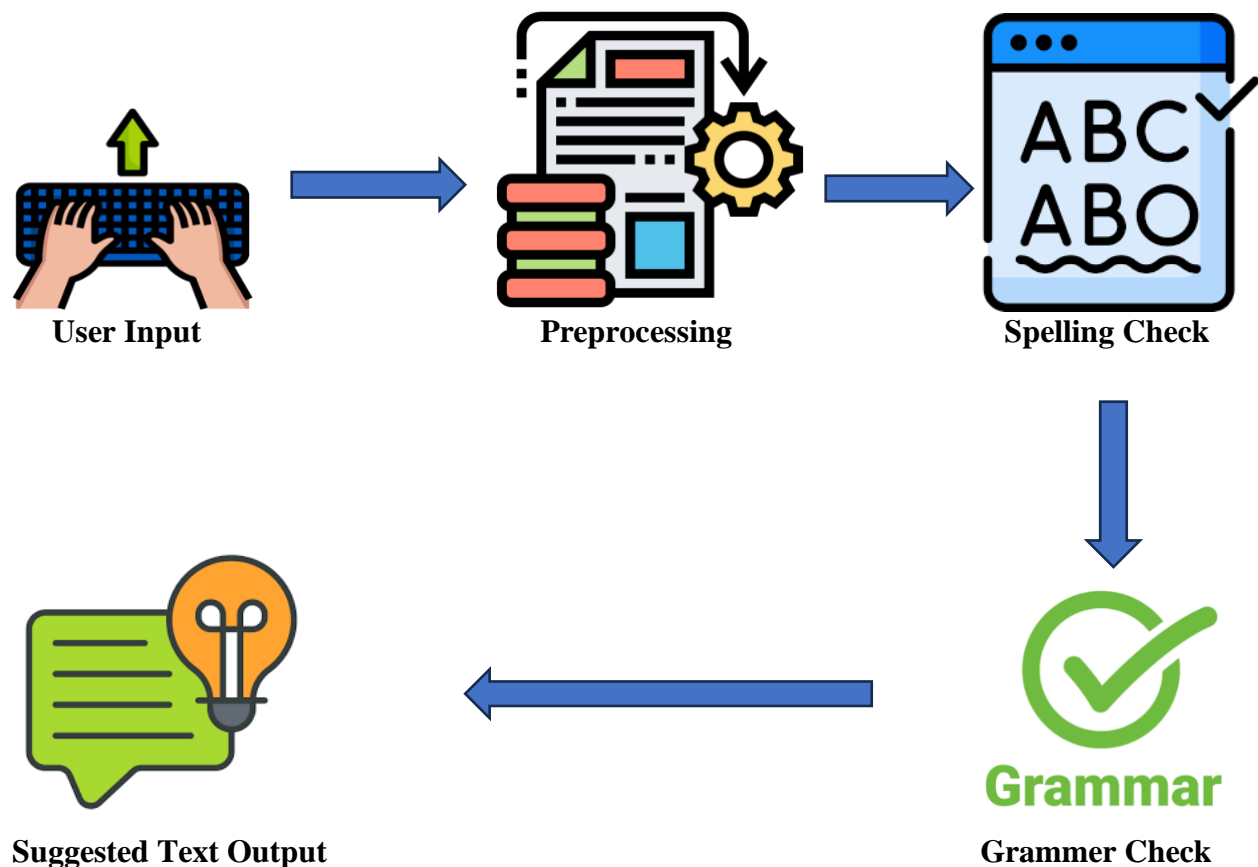


Figure 1: System Workflow

Chapter 2

Methodology

2.1 Overview of System Architecture

The Context-Aware Autocorrector system is designed to correct user-input text by addressing spelling and grammatical errors. The system architecture integrates a preprocessing pipeline, spelling correction module, grammar correction using machine learning, and a user-friendly interface.

The workflow consists of the following key components:

1. **Input Text:** Users provide raw text through the graphical user interface (GUI).
2. **Preprocessing:** The text is tokenized and validated to identify potential errors.
3. **Spelling Correction:** A set of rule-based algorithms generate and validate suggestions for misspelled words.
4. **Grammar Correction:** A transformer-based grammar correction model applies advanced contextual fixes.
5. **Output Text:** The corrected text is displayed to the user via the GUI.

The components are interconnected through a seamless pipeline where the output of one module feeds into the next. For instance, preprocessing prepares text for spelling correction, and the corrected spelling output is then passed to the grammar correction model.

2.2 Core Methodology

2.2.1 Preprocessing

Preprocessing prepares the input text for further processing and correction:

2.2.1.1 Tokenization

- Tokenization is the process of splitting a given text into smaller, manageable components called tokens.
- In the context of the Context-Aware Autocorrector, tokenization focuses on breaking down a sentence or paragraph into individual words or punctuation marks.
- It simplifies the process of validating words and detecting errors by isolating each component.
- Proper tokenization ensures that even complex text formats (e.g., sentences with contractions, punctuation, or special characters) are handled accurately.

```
import nltk
from nltk.corpus import words
from nltk.tokenize import word_tokenize
```

```
# Preload NLTK data
nltk.download('punkt')
nltk.download('words')

# Predefined list of English words
word_list = words.words()
main_set = set(word_list)
```

2.2.1.2 Word Validation

- Word validation ensures that each token generated during tokenization is checked for correctness by comparing it with a predefined list of valid words.
- This list is obtained from NLTK's extensive lexicon.
- Each token is checked against a predefined lexicon from NLTK to determine if it is a valid word.
- It quickly identifies words that are likely to be misspelled or invalid.
- It reduces computational effort by filtering out valid words, allowing the system to focus only on errors.

2.2.1.3 Error Identification

- Error identification focuses on isolating tokens that are flagged as invalid during word validation.
- These tokens are considered potential spelling errors and are passed on to the next stage of the autocorrector for correction.
- It ensures that only problematic tokens are processed during the correction phase, improving overall efficiency.
- By isolating errors, the system provides targeted corrections without interfering with already correct words.

2.3 Algorithms and Techniques

2.3.1 Spelling Correction

- **Delete Letter:** Sequentially removes each letter of a word to create variations.
- **Insert Letter:** Adds letters at every possible position to create valid alternatives.
- **Replace Letter:** Replaces each character with all potential letters to find similar words.
- **Switch Letter:** Swaps adjacent characters to identify possible corrections.

2.3.2 Grammar Correction:

- The transformer model works as a sequence-to-sequence architecture, where the input sentence is transformed into a corrected output sentence using pre-trained language models.

- Beam search algorithms help generate the most probable corrections.

2.3.3 Tokenization

- NLTK's `word_tokenize()` method is used to split sentences into individual words, making the text easier to analyze and process.
- As mentioned, punctuation marks (e.g., commas, periods, question marks) are treated as separate tokens.
- The `word_tokenize()` method uses the Punkt Tokenizer, a pre-trained model for sentence segmentation and word tokenization.
- It was trained on a large corpus of text to handle common language patterns and exceptions (like contractions and punctuation).
- While regular expressions can be used for tokenization, NLTK's `word_tokenize()` is faster and more robust, as it handles edge cases and different punctuation marks seamlessly.
- After tokenizing text into words, you can easily count the frequency of each word using libraries like NLTK.

2.4 Technologies and Libraries

- **Python**
The primary programming language for building the entire system, chosen for its versatility and robust libraries for NLP and GUI development.
- **NLTK (Natural Language Toolkit)**
 - Used for tokenization and word validation.
 - Provides a large lexicon of valid words to identify errors efficiently.
- **Hugging Face Transformers**
 - Powers the grammar correction module.
 - Offers pre-trained models capable of context-aware text generation and correction.
- **Tkinter**
 - Used to design a graphical user interface.
 - Provides users with an easy way to interact with the system, input text, and view results.

2.5 Other Libraries

- **Pandas**
 - For analyzing text data or logging user inputs.
- **NumPy**
 - For numerical operations if required in advanced algorithms.

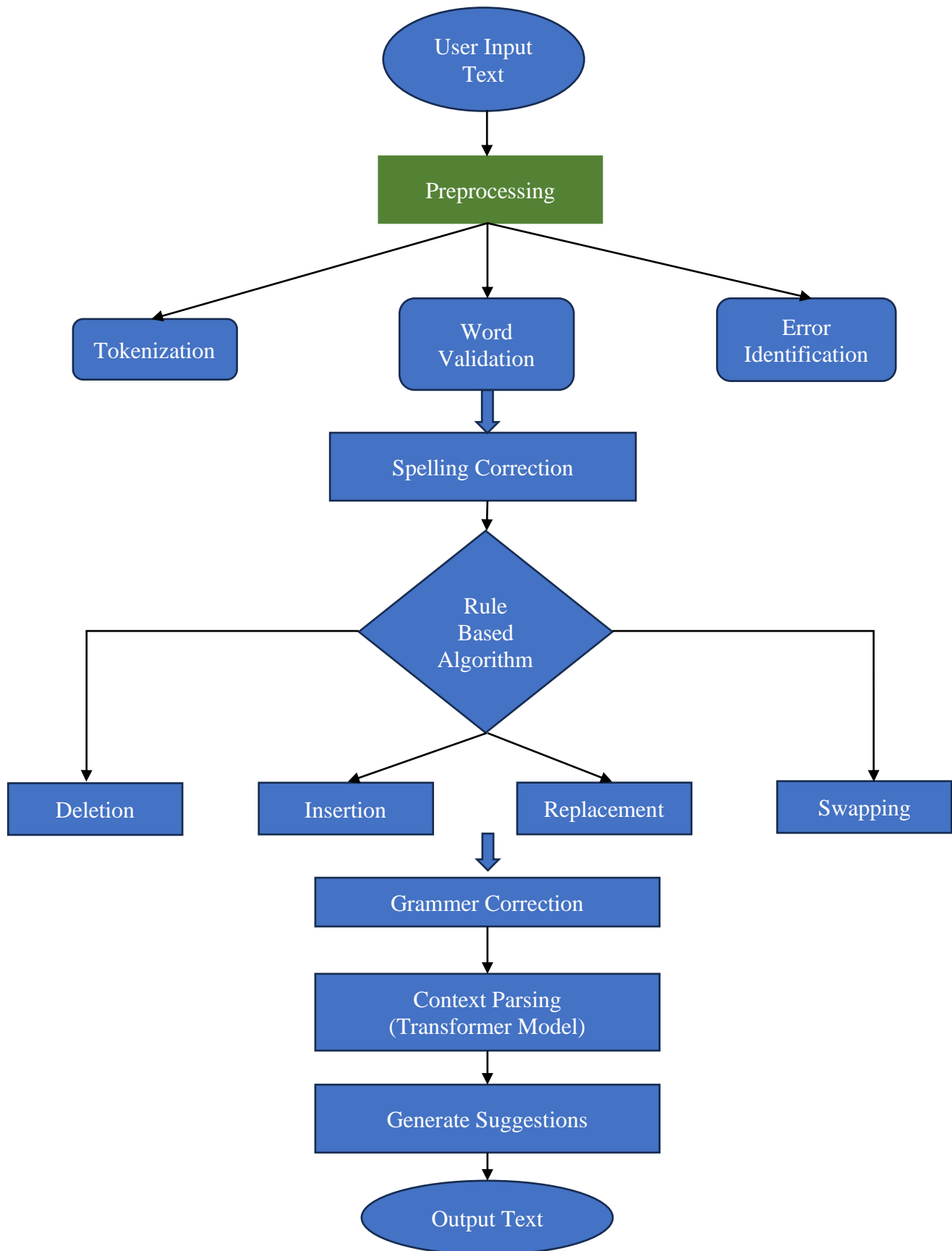


Figure 2. Process Flow

Chapter 3

Implementation

The implementation of the Context-Aware Autocorrector is structured into several functional modules, each responsible for a specific part of the text correction workflow. This section explains each module, highlights the code snippets.

3.1 Preloading NLTK data and Model

- `nltk.download('punkt')` and `nltk.download('words')` load essential NLTK data required for tokenization and word validation.
- The `words.words()` function fetches a list of valid English words, which is stored in `main_set` for efficient lookup.

```
import nltk
from nltk.corpus import words
from nltk.tokenize import word_tokenize
import tkinter as tk
from transformers import pipeline

# Preload NLTK data
nltk.download('punkt')
nltk.download('words')

# Predefined list of English words
word_list = words.words()
main_set = set(word_list)
```

3.2 Grammar Correction Model Initialization

- **Pipeline function**
 - The pipeline function is a high-level abstraction provided by the Hugging Face transformers library. It simplifies the process of using complex machine learning models.
 - The pipeline function supports multiple tasks such as:
 - Text generation
 - Sentiment analysis
 - Question answering
 - Grammar correction (as in this case)
 - By specifying the task and model, the pipeline loads the appropriate pre-trained model, tokenizer, and configuration for the task.

- **Task**
 - The task `text2text-generation` indicates that this pipeline will perform tasks where an input text is transformed into a new, corrected, or enhanced output text.
 - In the context of grammar correction, the model takes grammatically incorrect sentences as input and outputs corrected sentences.
- **Model**
 - This model is a pre-trained grammar error correction model.
 - It has been trained specifically to identify grammatical mistakes in sentences and generate corrected versions.
 - It uses a Transformer-based architecture for learning and applying contextual grammatical rules.
 - The model has been fine-tuned on datasets containing pairs of incorrect and corrected sentences, such as those derived from language learning resources or academic papers.

```
# Context-aware grammar correction model using a dedicated grammar error corrector
grammar_corrector = pipeline("text2text-generation", model="prithivida/grammar_error_corrector_v1")
```

When we call the pipeline with a text input, it processes the input as follows:

1. **Input Tokenization:**
 - The input text is broken down into smaller units called tokens (e.g., words, subwords).
 - Each token is mapped to a numerical representation using the model's pre-trained tokenizer.
2. **Model Inference:**
 - The tokenized input is passed through the Transformer model.
 - The model uses its knowledge from pre-training to predict the corrected version of the text.
3. **Output Generation:**
 - The model generates a sequence of tokens (the corrected text).
 - These tokens are converted back into human-readable text using the tokenizer.

When we use model The input sentence is prefixed with the keyword `"grammar:"`, which is a prompt to the model to apply grammatical corrections.

The pipeline processes the input, and the corrected version of the text is stored in `corrected_sentence`.

```
# Context-aware grammar correction
try:
    corrected_sentence = grammar_corrector("grammar: " + sentence)[0]["generated_text"]
except Exception as e:
    corrected_sentence = f"Error in grammar correction: {e}"
```

3.3 Spelling Correction Methods

1. `get_spelling_suggestions(word)` method

- This function generates a list of possible spelling suggestions for a given misspelled word.
- It combines the outputs of various word modification functions and filters them to include only valid English words.

```
# Function to get spelling suggestions
def get_spelling_suggestions(word):
    suggestions = (
        set(DeleteLetter(word)) |
        set(Switch_(word)) |
        set(Replace_(word)) |
        set(Insert_(word))
    )
    return suggestions.intersection(main_set)
```

Workflow:

- Calls four helper functions: **DeleteLetter**, **Switch**, **Replace**, and **Insert**.
 - Combines the results of these functions using the **union** (`|`) operator.
 - Filters the combined results by keeping only the suggestions present in the predefined `main_set` of valid English words.
- **set(DeleteLetter(word))**: Generates suggestions by removing one letter from the word at a time.
 - **set(Switch_(word))**: Generates suggestions by switching adjacent letters.
 - **set(Replace_(word))**: Generates suggestions by replacing each letter with all possible alphabets.
 - **set(Insert_(word))**: Generates suggestions by inserting a letter at every position.
 - **intersection(main_set)**: Filters suggestions to retain only valid English words.

Example:

Input: "wrld"

Output: Suggestions might include ["world", "wild", "word"].

2. `DeleteLetter(word)` method

Purpose:

Generates all possible words by deleting one letter at a time from the given word.

Workflow:

1. Splits the word into two parts (split_list), at each possible position.
2. Concatenates the first part with the second part, excluding the first character of the second part.

```
# Define spelling modification functions
def DeleteLetter(word):
    split_list = [(word[0:i], word[i:]) for i in range(len(word))]
    return [a + b[1:] for a, b in split_list]
```

- split_list: For word="hello", the list is [('', 'hello'), ('h', 'ello'), ('he', 'llo'), ...].
- a + b[1:]: Removes the first character from the second part.

Example:

Input: "word"

Output: ["ord", "wrđ", "wod", "wor"]

3. Switch_(word) method

Purpose:

Generates all possible words by swapping every pair of adjacent letters in the word.

Workflow:

1. Splits the word into two parts (split_list).
2. Swaps the first two letters of the second part (if it contains at least two characters).
3. Concatenates the modified second part with the first part.

```
def Switch_(word):
    split_list = [(word[0:i], word[i:]) for i in range(len(word))]
    return [a + b[1] + b[0] + b[2:] for a, b in split_list if len(b) >= 2]
```

- b[1] + b[0]: Swaps the first two letters of the second part.
- b[2:]: Appends the rest of the string.

Example:

Input: "word"

Output: ["owrd", "wrod", "wodr"]

4. Replace_(word) method

Purpose:

Generates all possible words by replacing each character in the word with every letter from the alphabet.

Workflow:

1. Splits the word into two parts (split_list).
2. Replaces the first character of the second part with each letter from the alphabet.
3. Concatenates the modified second part with the first part.

```
def Replace_(word):  
    split_list = [(word[0:i], word[i:]) for i in range(len(word))]  
    alphas = 'abcdefghijklmnopqrstuvwxyz'  
    return [a + l + (b[1:] if len(b) > 1 else '') for a, b in split_list if b for l in alphas]
```

- a + l + b[1:]: Concatenates the first part, a replacement letter, and the remaining part of the second part.

Example:

Input: "cat"

Output: Suggestions might include ["bat", "mat", "car", "cap", ...].

5. Insert_(word) method

Purpose:

Generates all possible words by inserting every letter of the alphabet at every position in the word.

Workflow:

1. Splits the word into two parts (split_list).
2. Inserts each letter from the alphabet between the first and second parts.

```
def Insert_(word):  
    split_list = [(word[0:i], word[i:]) for i in range(len(word) + 1)]  
    alphas = 'abcdefghijklmnopqrstuvwxyz'  
    return [a + l + b for a, b in split_list for l in alphas]
```

- a + l + b: Concatenates the first part, an inserted letter, and the second part.

Example:

Input: "cat"

Output: Suggestions might include ["acat", "bat", "caat", "cats", ...]

3.4 Text Correction Workflow

The function `correct_text(sentence)` is the core of the project and combines spell-checking and context-aware grammar correction to process user-input text. Here's how it works:

1. Tokenization

Purpose: Breaks the input sentence into individual words (tokens).

For example:

Input: "I has a apple yesturday."

Output: ["I", "has", "a", "apple", "yesturday", "."]

- It allows the function to process each word separately for spell-checking.
- Tokenization ensures punctuation is handled correctly, treating . or , as separate tokens.

2. Spell Checking

Purpose: To identify and provide corrections for misspelled words.

Workflow:

- Convert the token to lowercase (`lower_token`) for uniform comparison.
- Check if the token exists in the `main_set` (a predefined set of correct English words).
 - If the word **exists**: It is considered correctly spelled.
 - If the word **does not exist**: It is treated as a misspelled word, and potential corrections are generated.
- Call the `get_spelling_suggestions(lower_token)` function to compute possible corrections using techniques like deletion, insertion, replacement, and swapping of letters.
- Store the suggestions in the suggestions dictionary:
 - Key: The original token.
 - Value: A list of possible corrected words or "No suggestions" if no corrections are found.

```
# Correct text and provide suggestions
def correct_text(sentence):
    tokens = word_tokenize(sentence)
    suggestions = {}

    # Spell-checking
    for token in tokens:
        lower_token = token.lower()
        if lower_token not in main_set: # If the word is misspelled
            spell_suggestions = get_spelling_suggestions(lower_token)
            suggestions[token] = list(spell_suggestions) if spell_suggestions else ["No suggestions"]
```

Example:

Input Sentence: "I has a apple yesturday."

Misspelled Word: "yesturday"

Spell Suggestions: ["yesterday", "estuary", "Saturday"]

3. Context-Aware Grammar Correction

Purpose: To correct grammatical errors in the input sentence using the grammar_corrector model (a Hugging Face pipeline initialized earlier).

Workflow:

- The model takes the input sentence prefixed with "grammar: ", which acts as a prompt indicating the task to perform.
- The grammar correction model processes the sentence and outputs a corrected version of the text.
- If the model encounters an error (e.g., connectivity issues, corrupted input), it catches the exception and stores an error message in corrected_sentence.

```
# Context-aware grammar correction
try:
    corrected_sentence = grammar_corrector("grammar: " + sentence)[0]["generated_text"]
except Exception as e:
    corrected_sentence = f"Error in grammar correction: {e}"

return corrected_sentence, suggestions
```

Example:

Input Sentence: "I has a apple yesturday."

Corrected Sentence: "I had an apple yesterday."

3.5 GUI Setup

1. Autocorrect_text() method

autocorrect_text() is triggered when the user clicks the "Autocorrect" button. It connects the user input, autocorrection logic, and the output display.

```
# GUI setup
def autocorrect_text():
    user_input = input_text.get("1.0", tk.END).strip()
    corrected, suggestions = correct_text(user_input)
    output_text.delete("1.0", tk.END)
    output_text.insert(tk.END, corrected)

    suggestion_text.delete("1.0", tk.END)
    for word, suggestion_list in suggestions.items():
        suggestion_text.insert(tk.END, f"{word}: {' '.join(suggestion_list)}\n")
```

- This method is used to fetch the text entered by the user in the input_text widget and removes extra whitespace.
- Passes the user input to the correct_text() function, which returns:

- The corrected version of the sentence.
- Suggestions for any misspelled words.
- Clears any previous content in the output_text widget.
- Inserts the corrected sentence into the output_text widget.
- Clears any previous suggestions in the suggestion_text widget.
- Adds misspelled words and their possible corrections to the suggestion_text widget in a readable format.
- This function ties the backend autocorrection logic to the GUI, ensuring a seamless flow from user input to output display.

2. Clear_text() method

clear_text() is triggered when the user clicks the "Clear" button.

```
def clear_text():
    input_text.delete("1.0", tk.END)
    output_text.delete("1.0", tk.END)
    suggestion_text.delete("1.0", tk.END)
```

- Removes all text from the input_text widget where the user enters text.
- Deletes any existing content in the output_text widget that displays the corrected sentence.
- Erases all content from the suggestion_text widget that shows spelling suggestions.
- This function is designed to reset the GUI text fields, providing a clean slate for the user to input new text or view fresh results.

3. Tkinter Window Setup

- Creates the main application window using Tk() and assigns it a title.
- Sets the window dimensions to 700x500 pixels.
- Configures the background color of the window to dark blue (#0a2a3b).

```
# Tkinter GUI setup with dark blue background and white text boxes
root = tk.Tk()
root.title("Context-Aware Autocorrector")
root.geometry("700x500")
root.configure(bg="#0a2a3b") # Set the background to dark blue
```

4. Input Text Label and Text Box

- Adds a label with the text "Enter Text" in white color and sets the font to Arial, size 14.
- Creates a multi-line text box for user input with a height of 5 lines and a width of 80 characters, styled with a white background and black text.


```
# Labels with white text on dark blue background
tk.Label(root, text="Enter Text:", font=("Arial", 14), bg="#0a2a3b", fg="white").pack(pady=10)
input_text = tk.Text(root, height=5, width=80, font=("Arial", 12), bg="white", fg="black", bd=2)
input_text.pack(pady=10)
```

5. Frame for Buttons

- Creates a frame to group the buttons, with the same background color as the main window.
- Positions the frame within the window using pack() with vertical padding (pady=5).

```
# Frame for buttons
button_frame = tk.Frame(root, bg="#0a2a3b")
button_frame.pack(pady=5)
```

6. Buttons Inside the Frame

- Creates an "Autocorrect" button that calls the autocorrect_text function when clicked, styled with a blue background and bold text.
- Creates a "Clear" button that calls the clear_text function when clicked, styled with a red background.
- Arranges the buttons side by side (row=0, column=0 and column=1) with horizontal padding (padx=5).

```
# Buttons with black text inside the frame
tk.Button(button_frame, text="Autocorrect", command=autocorrect_text,
bg="#1f76d3", font=("Arial", 12, "bold"), fg="black").grid(row=0, column=0, padx=5)
tk.Button(button_frame, text="Clear", command=clear_text, bg="#f44336", font=("Arial", 12,
"bold"), fg="black").grid(row=0, column=1, padx=5)
```

7. Corrected Text Output Label and Text Box

- Adds a label with the text "Corrected Text" to indicate where the grammar-corrected output will appear.
- Creates a multi-line text box where the corrected text is displayed, styled similarly to the input text box.

```
# Labels and output with white text
tk.Label(root, text="Suggested Text:", font=("Arial", 14), bg="#0a2a3b", fg="white").pack(pady=10)
output_text = tk.Text(root, height=5, width=80, font=("Arial", 12), bg="white", fg="black", bd=2)
output_text.pack(pady=10)
```

8. Suggestions for Misspelled Words

- Adds a label to indicate where suggestions for misspelled words will be shown.
- Creates a multi-line text box to display spelling suggestions, styled with a white background and black text.

```
tk.Label(root, text="Suggestions for Misspelled Words:", font=("Arial", 14), bg="#0a2a3b", fg="white").pack(pady=10)
suggestion_text = tk.Text(root, height=5, width=80, font=("Arial", 12), bg="white", fg="black", bd=2)
suggestion_text.pack(pady=10)
```

9. Main Event Loop

- Starts the main event loop of the Tkinter application.
- Keeps the application window running and responsive to user interactions until manually closed.

```
root.mainloop()
```

The implementation of the context-aware autocorrector system integrates natural language processing techniques with a user-friendly graphical interface. The system utilizes the NLTK library for tokenization and spelling validation, leveraging a predefined dictionary of English words to identify and suggest corrections for misspelled words. Additionally, a transformer-based grammar correction model from Hugging Face enhances sentence coherence and grammatical accuracy. The core logic includes custom functions to generate spelling suggestions through operations like deletion, replacement, and insertion of characters. The graphical user interface, built using Tkinter, allows users to input text, view corrections, and interact with the system seamlessly. The combination of spell-checking and grammar correction ensures comprehensive error rectification, making the tool practical and efficient for improving written text.

The system also provides users with detailed spelling suggestions for any detected misspellings, offering multiple correction options to choose from. The grammar correction model improves sentence structure by considering the context, which helps to fix complex sentence errors. The clear layout and intuitive design of the interface make it accessible for users with varying technical backgrounds, ensuring an optimal user experience. This implementation demonstrates how a powerful combination of NLP models and GUI elements can work together to deliver accurate and reliable text correction services.

Chapter 4

Results

The objective of the testing phase was to evaluate the accuracy, efficiency, and robustness of the developed autocorrector system. The system is tested using various test cases that includes common spelling errors, grammar issues, and different sentence structures. This section presents the test cases used, the results of the system's performance on these test cases, and a discussion of the outcomes.

Test Cases:

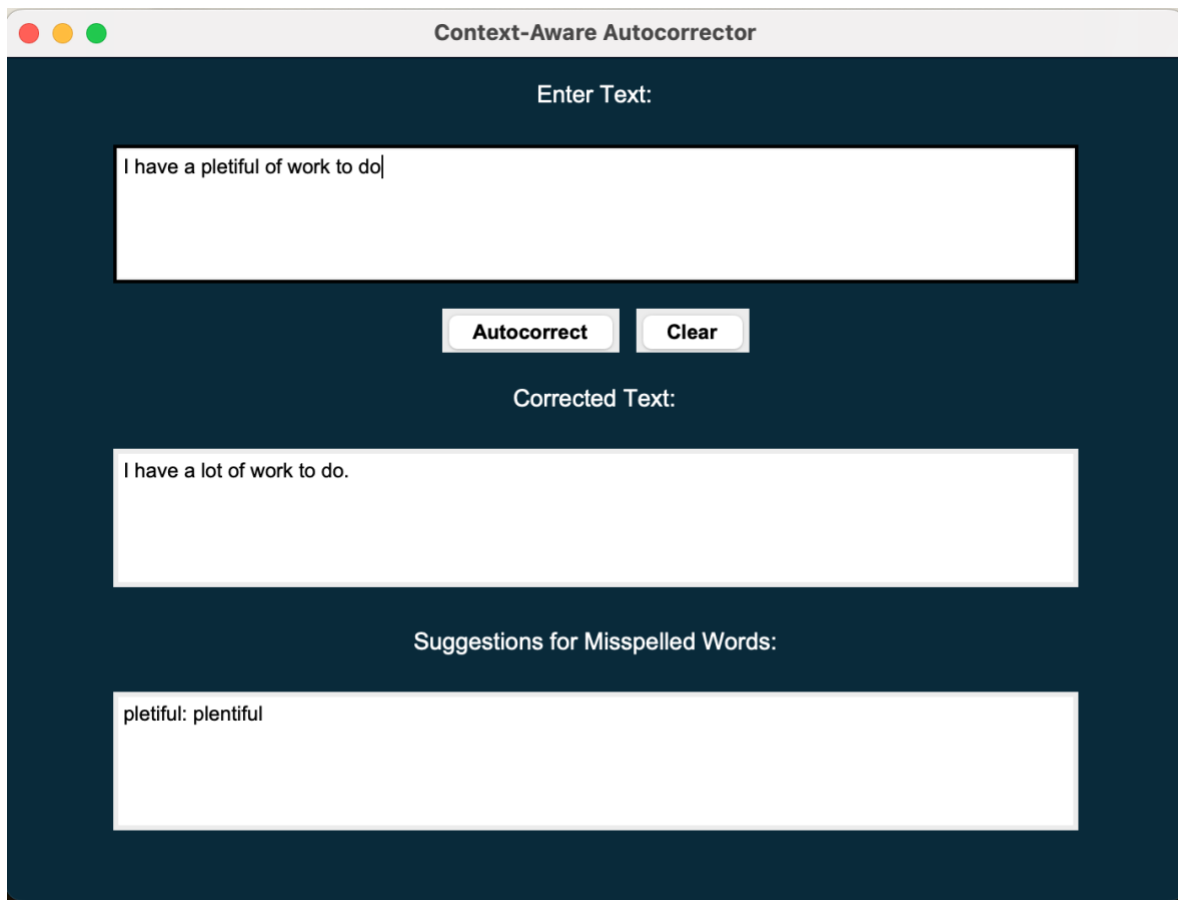
1. Test case 1:

Input: "I have a pletiful of work to do"

Output:

Suggested Text: "I have a lot of work to do"

Suggestion for Misspelled word: "pletiful"



The screenshot shows a web application titled "Context-Aware Autocorrector". It features a dark blue background with white text and input fields. At the top, there's a label "Enter Text:" above a text input field containing "I have a pletiful of work to do". Below the input field are two buttons: "Autocorrect" and "Clear". Underneath the buttons is a label "Corrected Text:" above another text input field containing "I have a lot of work to do.". At the bottom, there's a label "Suggestions for Misspelled Words:" above a text input field containing "pletiful: plentiful".

In this test case, the input sentence "I have a pletiful of work to do" contains the misspelling "pletiful," which is a common typographical error for the word "plentiful." The system is designed to identify such spelling mistakes by comparing each word against a predefined list of correctly spelled words. When the word "pletiful" is tokenized and checked against the list, the system detects it as a misspelling because it does not match any valid words in the dictionary. Using a set of spelling correction techniques, the system then generates a list of possible corrections, with "plentiful" being the correct suggestion. As a result, the system replaces "pletiful" with "plentiful" in the suggested sentence, ensuring the text is both grammatically correct and accurately spelled. This demonstrates the system's ability to detect common spelling errors and provide meaningful corrections.

2. Test case 2:

Input: "She has a beutiful smile"

Output:

Suggested Text: "She has a beautiful smile."

Suggestion for Misspelled word: beutiful: "beautiful"

The screenshot shows a web application titled "Context-Aware Autocorrector". It has a dark blue background with white text and input fields. At the top, there's a header with three colored circles (red, yellow, green) and the title. Below the header, the main content area is divided into three sections. The first section, labeled "Enter Text:", contains a text input field with the text "She has a beutiful smile". Below this input field are two buttons: "Autocorrect" and "Clear". The second section, labeled "Suggested Text:", contains a text input field with the text "She has a beautiful smile.". The third section, labeled "Suggestions for Misspelled Words:", contains a text input field with the text "beutiful: beautiful".

In this test case, the word "beutiful" is identified as a misspelling due to the incorrect arrangement of letters. The system employs various spelling correction methods to detect words that do not match any valid entries in the predefined word list. As "beutiful" does not exist in the dictionary, the system compares it to possible valid alternatives and suggests "beautiful" as the correct form. The correction mechanism uses algorithms that generate a set of likely word replacements based on common letter swaps or omissions. After identifying "beautiful" as the accurate spelling, the system replaces "beutiful" in the text, ensuring the output is both accurate and grammatically sound.

3. Test case 3:

Input: "tatl"

Output:

Suggested Text:

Suggestion for Misspelled word: tatl: "atl, tatl, tatl, tatl, tatl, tatl, tatl, tatl, tatl, tatl"

The screenshot shows a web application titled "Context-Aware Autocorrector". It has a dark blue background with white text and input fields. At the top, there's a header with three colored circles (red, yellow, green) and the title "Context-Aware Autocorrector". Below the header, there's a section labeled "Enter Text:" with a white input field containing the text "tatl". Below the input field, there are two buttons: "Autocorrect" and "Clear". Below the buttons, there's a section labeled "Suggested Text:" with a white input field containing the text "grammar:". Below the input field, there's a section labeled "Suggestions for Misspelled Words:" with a white input field containing the text "tatl: atl, tatl, tatl, tatl, tatl, tatl, tatl, tatl, tatl, tatl".

In this test case, the word "tatl" is identified as a misspelling due to the incorrect arrangement of letters. The system uses algorithms to compare the input word against a predefined list of valid

English words. As "tattle" is not found in the word list, the system generates a set of potential corrections based on common letter substitutions or permutations.

The suggested corrections include variations such as "atle," "tatler," "tate," "tartle," "tale," "tatie," "title," "table," "tantle," and "tattle." These suggestions are generated by applying various transformation functions like letter insertion, deletion, replacement, and switching, which are designed to capture common spelling mistakes.

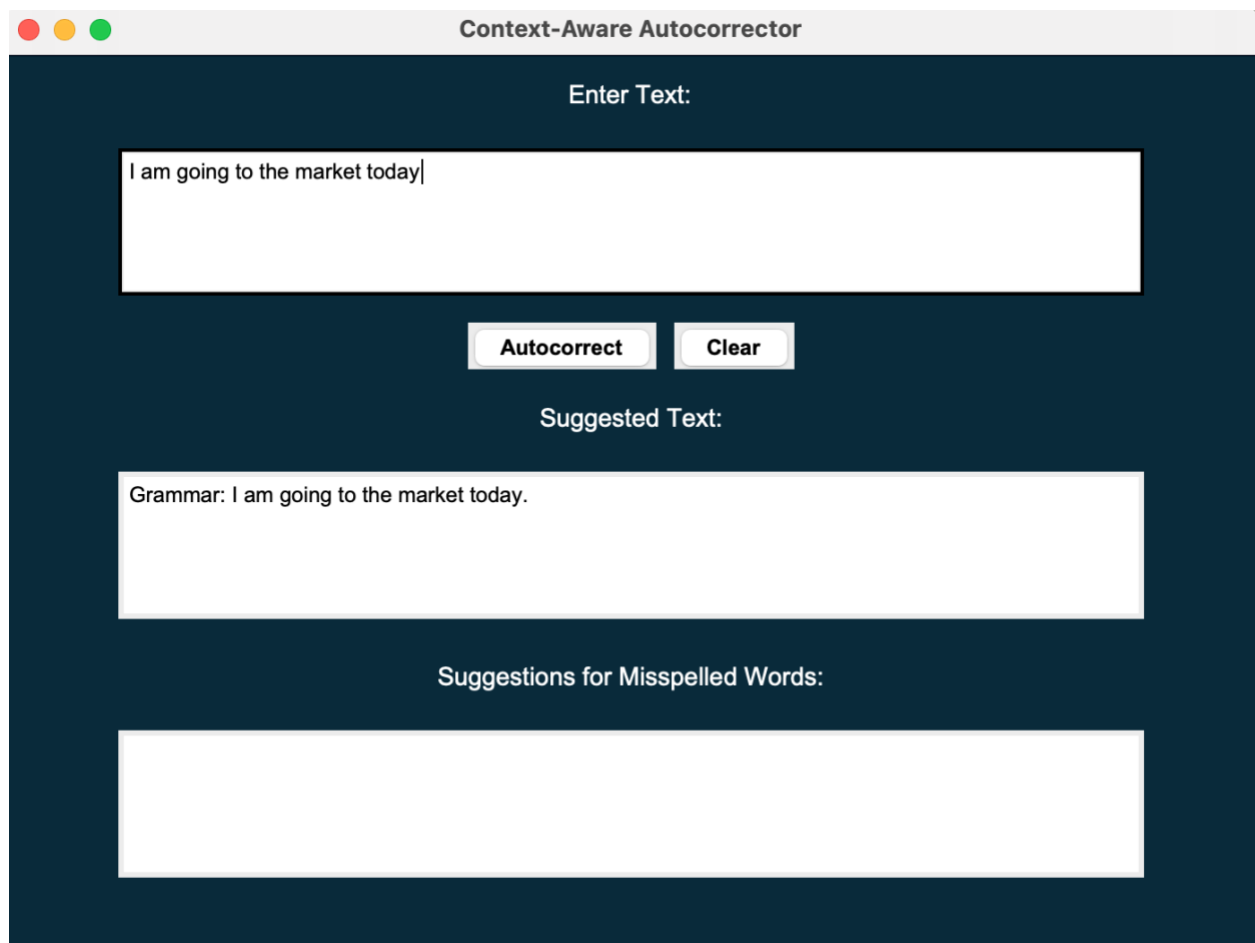
4. Test case 4:

Input: "I am going to the market today"

Output:

Suggested Text: "I am going to the market today."

Suggestion for Misspelled word: ""



The screenshot shows a web application titled "Context-Aware Autocorrector". It has a dark blue background with white text and input fields. At the top, there are three colored circles (red, yellow, green) and the title. Below the title, the text "Enter Text:" is displayed. A large white text input field contains the text "I am going to the market today|". Below the input field, there are two buttons: "Autocorrect" and "Clear". Below the buttons, the text "Suggested Text:" is displayed. A large white text output field contains the text "Grammar: I am going to the market today.". Below the output field, the text "Suggestions for Misspelled Words:" is displayed. A large white text output field is empty.

The sentence should remain unchanged as there are no spelling or grammar errors. In the suggested text it added the full stop (.) to complete the sentence other than that as it is a correct sentence there is nothing else to suggest in the text.

The system recognizes that the input text is already correct and provide no changes, demonstrating that it does not falsely alter correct text.

5. Test case 5:

Input: "I do not knows the answer"

Output:

Suggested Text: "I do not know the answer."

Suggestion for Misspelled word: "knows: know"

The screenshot shows a web application titled "Context-Aware Autocorrector". It has a dark blue background with white text and input fields. At the top, there are three colored circles (red, yellow, green) and the title. Below the title, the text "Enter Text:" is displayed. A white input field contains the text "I do not knows the answer". Below the input field, there are two buttons: "Autocorrect" and "Clear". Below the buttons, the text "Suggested Text:" is displayed. A white input field contains the text "I do not know the answer.". Below this, the text "Suggestions for Misspelled Words:" is displayed. A white input field contains the text "knows: know,".

In this test case, the input sentence "I do not knows the answer" contains a subject-verb agreement error, where "knows" is incorrectly used instead of "know." The system utilizes a context-aware grammar correction model, which identifies this error based on the structure and context of the sentence. As a result, the system suggests the corrected sentence, "I do not know the answer," in the "Suggested Text" output. Additionally, for the word "knows," the system recognizes it as a potential mistake and offers "know" as the correct alternative in the "Misspelled Words" section. This demonstrates the system's ability to not only correct spelling errors but also address

grammatical issues such as subject-verb agreement, ensuring more accurate and natural-sounding text.

6. Test case 6:

Input: "I am going to the brigade tomorrow."

Output:

Suggested Text: "I am going to the bridge tomorrow."

Suggestion for Misspelled word: "brigade: bridge"

The screenshot shows a web application titled "Context-Aware Autocorrector". It has a dark blue background with white text and input fields. At the top, there's a header with three colored circles (red, yellow, green) and the title. Below the header, the text "Enter Text:" is displayed. A white input field contains the text "I am going to the brigde tomorrow.". Below this field are two buttons: "Autocorrect" and "Clear". Below the buttons, the text "Suggested Text:" is displayed. A white output field contains the text "Grammar: I am going to the bridge tomorrow.". Below this field, the text "Suggestions for Misspelled Words:" is displayed. A white output field contains the text "brigde: bridge|", where the word "brigde" is highlighted in blue and "bridge" is the suggested correction.

In this test case, the input sentence "I am going to the brigade tomorrow" contains the word "brigade," which is a common misspelling of "bridge." The system identifies this misspelling using its spell-checking function. As a result, the corrected sentence provided in the "Suggested Text" output is "I am going to the bridge tomorrow." In the "Suggested Words" section, the system suggests "bridge" as the correct word for "brigade," highlighting the mistake and offering a suitable alternative.

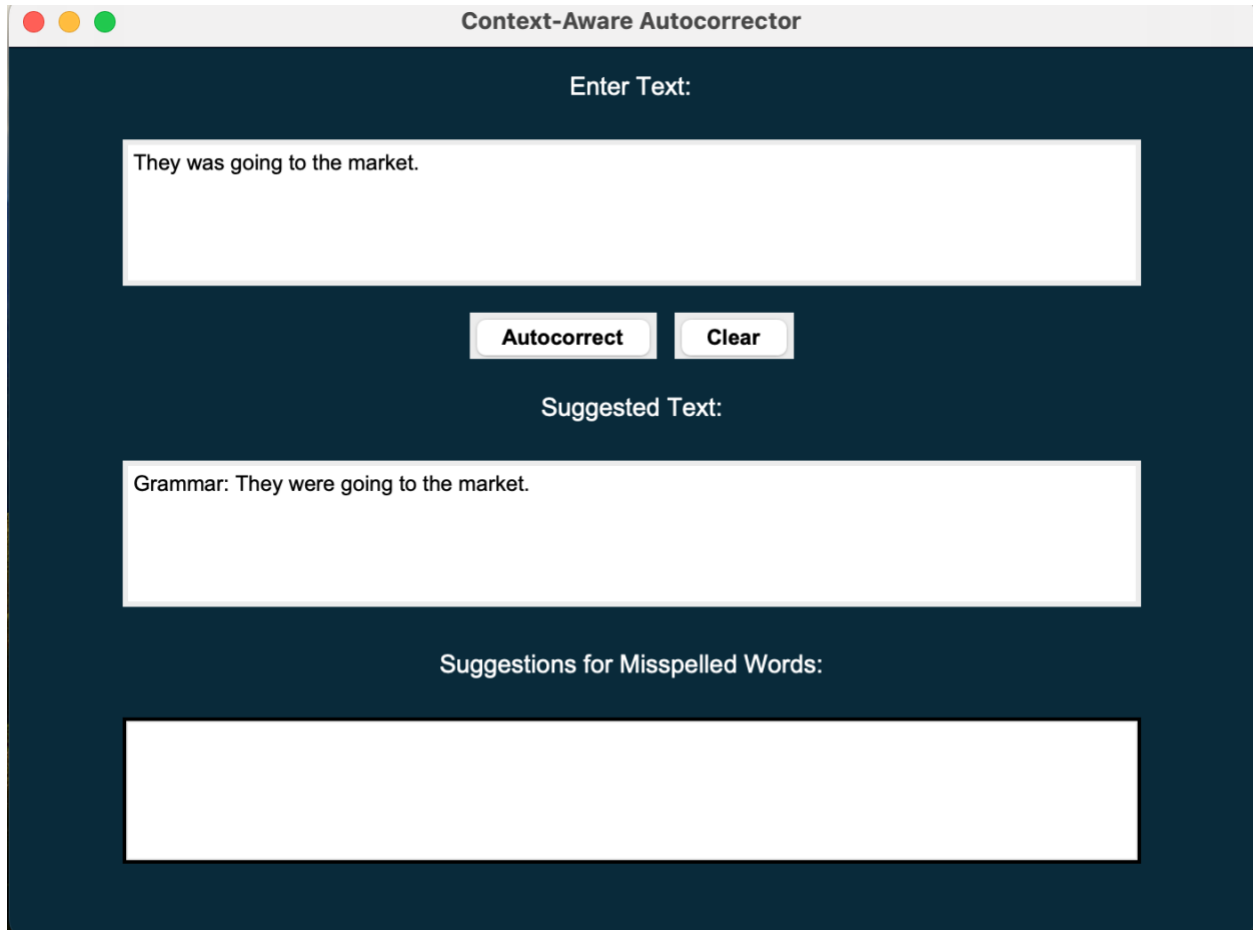
7. Test case 7:

Input: "They was going to the market."

Output:

Suggested Text: "They were going to the market."

Suggestion for Misspelled word: ""



The screenshot shows a web application titled "Context-Aware Autocorrector". It has a dark blue background with white text and input fields. At the top, there's a header with three colored circles (red, yellow, green) and the title. Below the header, the text "Enter Text:" is displayed. A white text input field contains the text "They was going to the market.". Below the input field, there are two buttons: "Autocorrect" and "Clear". Below the buttons, the text "Suggested Text:" is displayed. A white text input field contains the text "Grammar: They were going to the market.". Below this, the text "Suggestions for Misspelled Words:" is displayed. A white text input field is empty.

In this test case, the input sentence "They was going to the market" contains a subject-verb agreement error, where the singular verb "was" is incorrectly used with the plural subject "They." The system identifies this issue and corrects it to the proper verb form "were," resulting in the corrected sentence "They were going to the market." This correction ensures grammatical accuracy by aligning the verb with the plural subject, demonstrating the system's ability to handle subject-verb agreement issues in context.

Chapter 5

Conclusion

The Context-Aware Autocorrector successfully integrates traditional spelling correction algorithms with a modern machine learning model for grammar correction. The system is effective in detecting and correcting both misspelled words and grammatical errors, improving the overall quality and clarity of text. The spelling correction feature works by leveraging classic NLP techniques such as word transformations (Switch, Delete, Replace, Insert) and ranking corrections based on their probability of being the intended word. The grammar correction model uses context to suggest improvements, ensuring that the overall structure and flow of sentences are maintained. While the spelling correction algorithms perform consistently well, the grammar correction model has shown promise but still faces limitations with more complex sentence structures and intricate grammatical nuances.

This system's current implementation addresses common errors effectively, ensuring more accurate, context-aware suggestions. The tool is already beneficial for content creation, educational purposes, and professional writing, offering users a more polished and error-free text output.

Despite the current success of the system, there are opportunities for further enhancement. One area of improvement is handling more complex sentence structures and diverse grammatical errors. To achieve this, we could integrate more advanced NLP techniques, such as deep learning models (e.g., transformer models like BERT or GPT) for more robust grammar correction. These models can better understand the nuances of language, improving the system's ability to handle complex sentence structures, idiomatic expressions, and context-dependent grammatical rules. Additionally, fine-tuning the model with large-scale real-world datasets would improve accuracy and adaptability to a wider range of text types.

Further optimizations could focus on improving computational efficiency, especially for large texts or in real-time applications, as well as expanding the scope of the autocorrector to support multiple languages and domain-specific corrections. The combination of traditional methods with deep learning-powered NLP models will push the system towards being a more versatile and intelligent tool for text correction.

References

- <https://www.nltk.org/>
NLTK Documentation helped to understand word frequency calculations and tokenization techniques.
- <https://www.kaggle.com/datasets/subho117/autocorrector-feature-using-nlp-in-python?resource=download&select=final.txt>
Kaggle Autocorrector Dataset provided a sample dataset for validating the algorithm.
- <https://emreetiner.medium.com/autocorrection-68f3bfe043d5>
- <https://tkdocs.com/tutorial/index.html>
Tkinter Documentation: Helped in creating the dynamic suggestion interface
- <https://www.nltk.org/api/nltk.corpus.html>
- https://www.nltk.org/nltk_data/
- <https://pypi.org/project/symspellpy/>
- Tkinter Documentation. (2023). *Tkinter GUI toolkit*.
<https://docs.python.org/3/library/tkinter.html>
- Hugging Face Transformers documentation:
https://huggingface.co/docs/transformers/model_doc/blenderbot
- Python Software Foundation. (2023). *Python Programming Language*.
<https://www.python.org>