WriteRight: Simplifying Spelling Corrections with NLP

Project Description

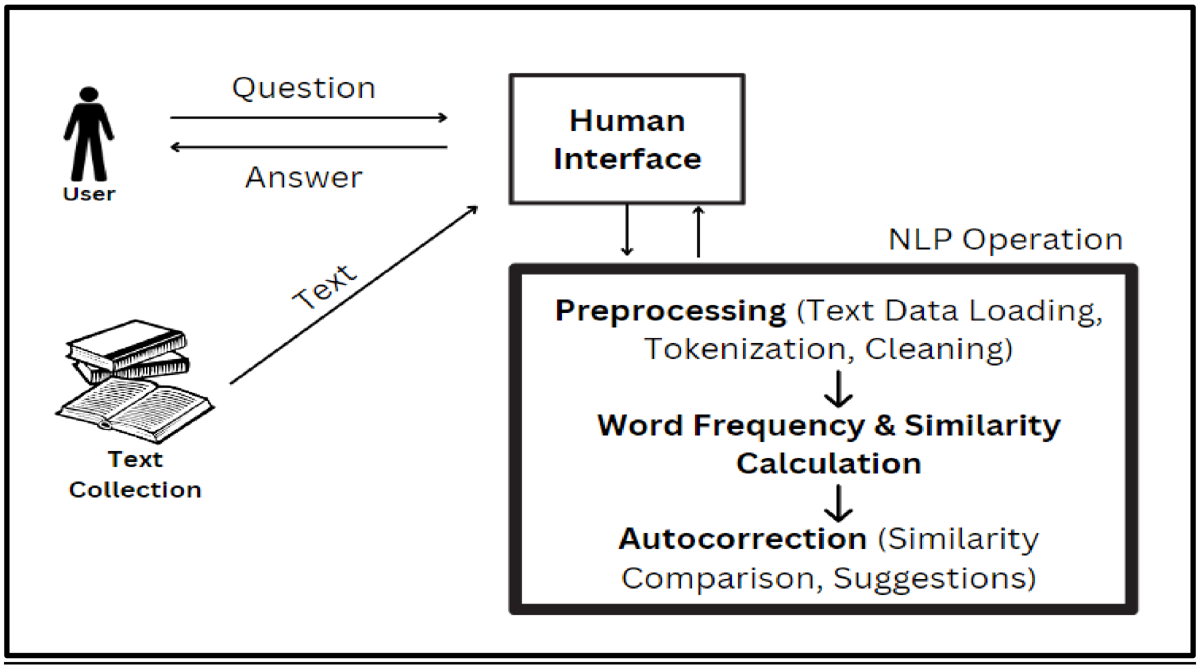
WriteRight is an innovative solution powered by Natural Language Processing (NLP) technology, aimed at simplifying the process of correcting spelling errors in written content. This advanced system addresses the limitations of traditional autocorrection features by leveraging NLP to intelligently identify and rectify spelling mistakes, ensuring that users can convey their messages with precision and professionalism effortlessly.

Scenario 1: Professional Email Composition
In the fast-paced corporate world, professionals often rely on email communication for business correspondence. However, errors in spelling and grammar can undermine the professionalism of these emails. With WriteRight, professionals can compose emails confidently, knowing that spelling errors will be promptly corrected. For instance, WriteRight will automatically detect and rectify spelling mistakes in critical emails to clients or colleagues, ensuring that the communication reflects the sender's professionalism and attention to detail.

Scenario 2: Academic Writing Assistance
Students and researchers face the challenge of maintaining accuracy and clarity in their academic writing. However, errors in spelling can detract from the credibility of their work. WriteRight provides valuable assistance by identifying and correcting spelling errors in academic papers, essays, and research articles. For example, WriteRight will automatically correct spelling mistakes in a student's thesis, ensuring that the final document meets the highest standards of academic integrity and professionalism.

Scenario 3: Social Media Engagement
In today's digital age, social media platforms serve as vital channels for personal and professional communication. However, errors in spelling can detract from the effectiveness of social media posts and comments. With WriteRight, users can enhance their social media engagement by ensuring that their posts are free from spelling errors. For instance, WriteRight will automatically correct spelling mistakes in a user's Facebook status or Instagram caption, helping them convey their message clearly and professionally to their audience.

Technical Architecture


Question
Answer
User
Human Interface
Text
Text Collection
NLP Operation
Preprocessing (Text Data Loading, Tokenization, Cleaning)
Word Frequency & Similarity Calculation
Autocorrection (Similarity Comparison, Suggestions)

Project Flow:

- User interacts with the Streamlit UI to enter the input sentence.
- Input sentences are tokenized and each word is checked against the vocabulary.
- Misspelled words are processed by the autocorrect function, which calculates similarity scores using the Jaccard metric.
- Suggestions for misspelled words are returned and displayed in the frontend.

To accomplish this, we have to complete all the activities listed below:

- **Requirements Specification**
    - Create a requirements.txt file to list the required libraries.
    - Install the required libraries
- **Interfacing with NLP Technique**
    - Implement an autocorrect function
- **Model Deployment**
    - Integrate with Web Framework
    - Host the Application

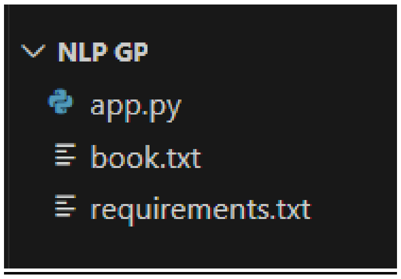
Prior Knowledge:

You must have the prior knowledge of the following topics to complete this project.

- Natural Language Processing Concepts
- NLP: https://www.tutorialspoint.com/natural_language_processing/index.htm
- textdistance: https://pypi.org/project/textdistance/
- Jaccard Distance: https://www.geeksforgeeks.org/find-the-jaccard-index-and-jaccard-distance-between-the-two-given-sets/
- Streamlit: https://www.geeksforgeeks.org/a-beginners-guide-to-streamlit/

Project Structure

Create the Project folder which contains files as shown below:



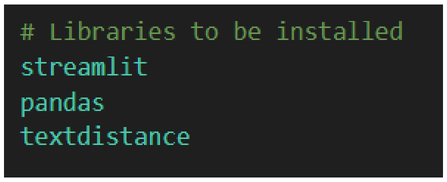
```
NLP GP
  app.py
  book.txt
  requirements.txt
```

- app.py: It serves as the primary application file housing both the model and Streamlit UI code.
- book.txt: It is used to train the vocabulary and word frequency model for the auto correction functionality.
- requirements.txt: It enumerates the libraries necessary for installation to ensure proper functioning.
- Additionally, ensure proper file organization and adhere to best practices for version control.

Milestone 1: Requirements Specification

Specifying the required libraries in the requirements.txt file ensures seamless setup and reproducibility of the project environment, making it easier for others to replicate the development environment.

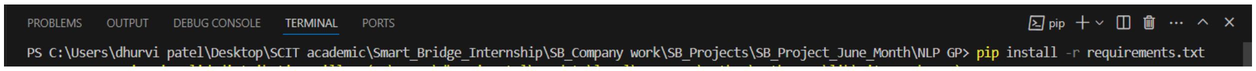
Activity 1: Create a requirements.txt file to list the required libraries.



```
# Libraries to be installed
streamlit
pandas
textdistance
```

- <u>streamlit</u>: Streamlit is a powerful framework for building interactive web applications with Python.
- <u>pandas:</u> Used for data manipulation and analysis, particularly for handling data structures like DataFrames.
- <u>textdistance:</u> Used for calculating similarity and distance metrics between sequences of text.

Activity 2: Install the required libraries



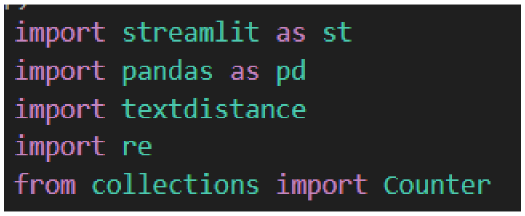
```
PS C:\Users\dhurvi patel\Desktop\SCIT academic\Smart_Bridge_Internship\SB_Company work\SB_Projects\SB_Project_June_Month\NLP GP> pip install -r requirements.txt
```

- Open the terminal.
- Run the command: pip install -r requirements.txt
- This command installs all the libraries listed in the requirements.txt file.

<u>Milestone 2: Interfacing with NLP Technique</u>

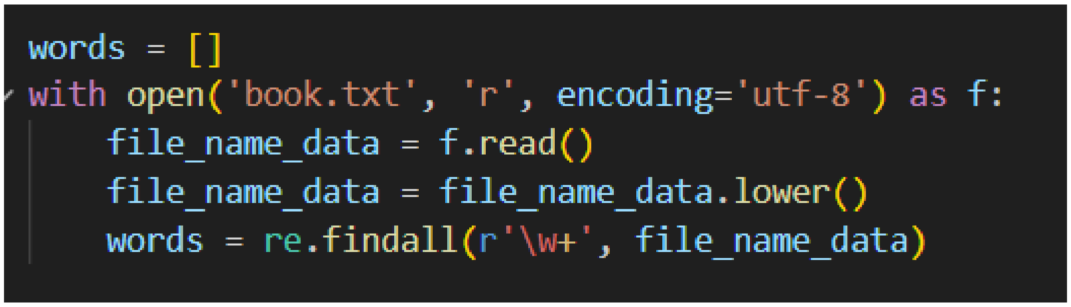
The NLP technique employed for the project is Jaccard distance. The Jaccard distance is an NLP technique used to measure the dissimilarity between two sets. It is calculated as one minus the Jaccard index, which is the size of the intersection divided by the size of the union of the sets. This metric is particularly useful for comparing text sequences based on their unique and common elements.

Activity 1: Implement an autocorrect function



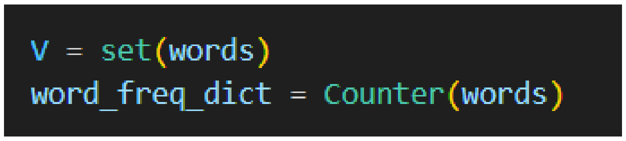
```
import streamlit as st
import pandas as pd
import textdistance
import re
from collections import Counter
```

- **Streamlit, Pandas, Textdistance, re, Counter Imports:** Importing necessary libraries for building the application, processing text data, calculating text similarity, and performing regular expression operations.



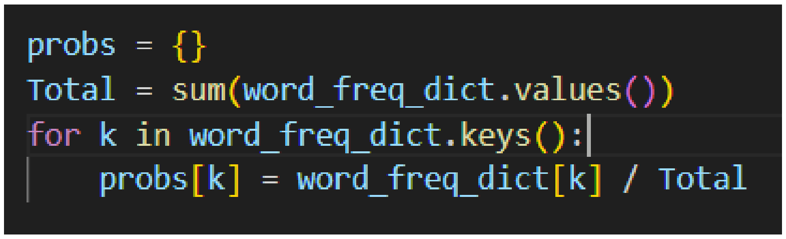
```python
words = []
with open('book.txt', 'r', encoding='utf-8') as f:
    file_name_data = f.read()
    file_name_data = file_name_data.lower()
    words = re.findall(r'\w+', file_name_data)
```

- Reading and Preprocessing Text Data:
    - The code reads the contents of the file book.txt.
    - Converts all text to lowercase to ensure case-insensitive matching.
    - Tokenizes the text into individual words using regular expressions (re.findall).
    - Stores the words in a list called words.



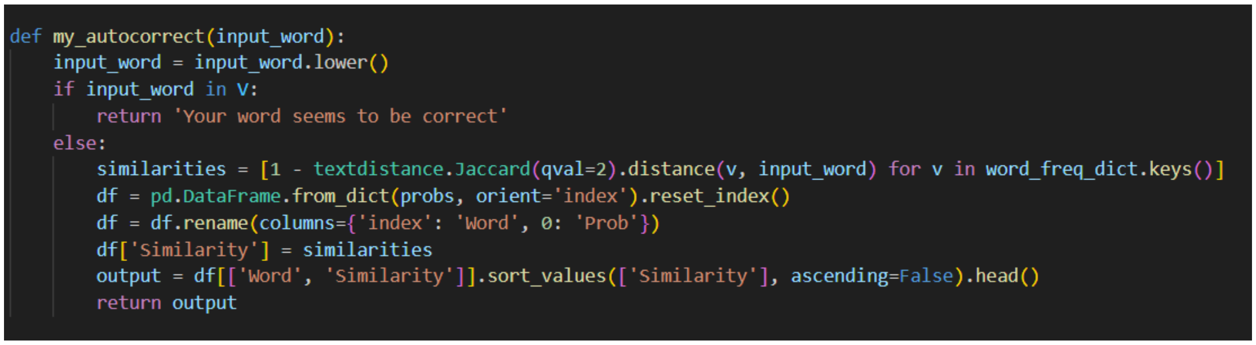
```python
V = set(words)
word_freq_dict = Counter(words)
```

- Creating Vocabulary and Word Frequency Dictionary:
    - Converts the list of words into a set to create a vocabulary (V) containing unique words.
    - Uses Counter to create a dictionary (word_freq_dict) that maps each word to its frequency in the text.



```python
probs = {}
Total = sum(word_freq_dict.values())
for k in word_freq_dict.keys():
    probs[k] = word_freq_dict[k] / Total
```

- Calculating Relative Frequencies:
    - Calculates the relative frequency of each word in the text.
    - Divides the frequency of each word by the total number of words to obtain probabilities (probs dictionary).



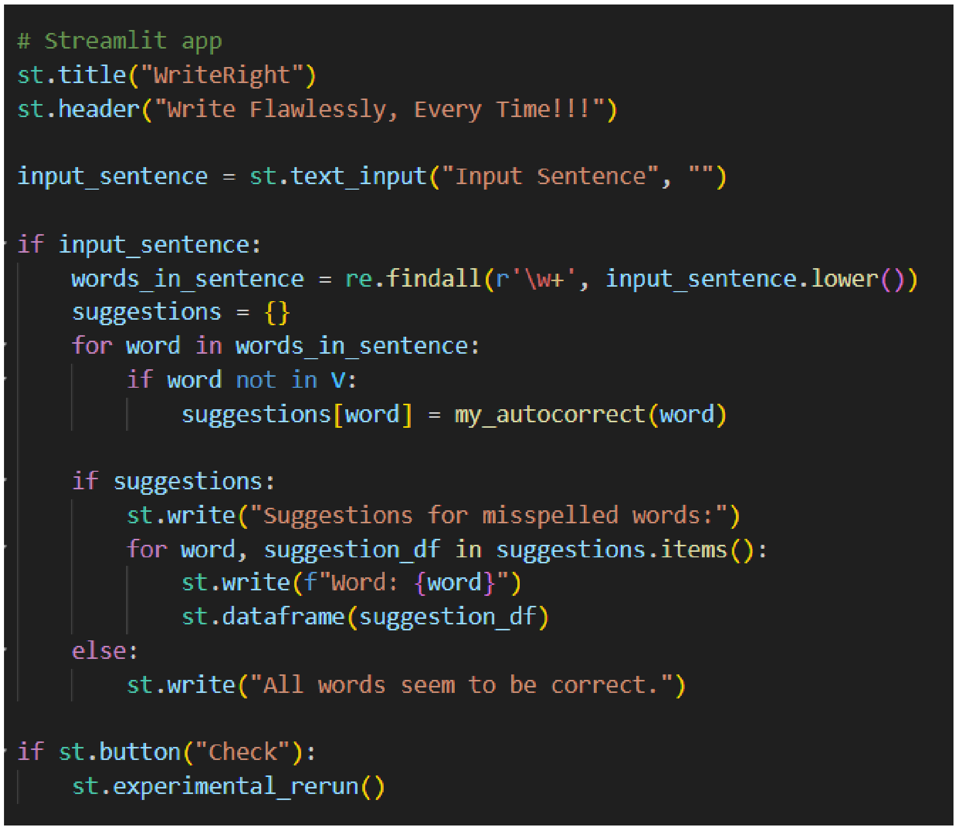
```python
def my_autocorrect(input_word):
    input_word = input_word.lower()
    if input_word in V:
        return 'Your word seems to be correct'
    else:
        similarities = [1 - textdistance.Jaccard(qval=2).distance(v, input_word) for v in word_freq_dict.keys()]
        df = pd.DataFrame.from_dict(probs, orient='index').reset_index()
        df = df.rename(columns={'index': 'Word', 0: 'Prob'})
        df['Similarity'] = similarities
        output = df[['Word', 'Similarity']].sort_values(['Similarity'], ascending=False).head()
        return output
```

- **Input Normalization**: Converts the input word to lowercase for case-insensitive matching.
- **Checking Word Existence**: If the input word exists in the vocabulary (V), it is considered correct, and a message indicating this is returned.
- **Calculating Similarity Scores**: If the input word is not found in the vocabulary, the function calculates Jaccard similarity scores between the input word and all words in the vocabulary.
    - It uses a list comprehension to calculate the Jaccard similarity score for each word (v) in the vocabulary.
    - The Jaccard similarity score is computed using the textdistance. Jaccard method with a q-value of 2, measuring the similarity between the input word and each vocabulary word.
- **Creating DataFrame**: It constructs a DataFrame (df) from the probabilities (probs dictionary) and renames columns for clarity.
    - The DataFrame includes columns for words (Word), their probabilities (Prob), and their similarity scores (Similarity).
- **Output Generation**: It returns the top similar words along with their similarity scores sorted in descending order.
    - The output is a DataFrame containing the top similar words and their similarity scores, sorted by similarity in descending order.

Milestone 4: Model Deployment

We deploy our model using the Streamlit framework, a powerful tool for building and sharing data applications quickly and easily. With Streamlit, we can create interactive web applications that allow users to interact with our models in real-time, providing an intuitive and seamless experience.

Activity 1: Integrate with Web Framework



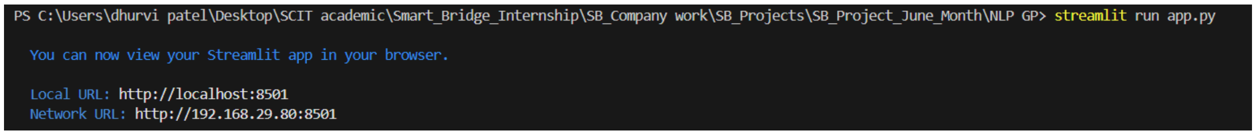
```python
# Streamlit app
st.title("WriteRight")
st.header("Write Flawlessly, Every Time!!!")

input_sentence = st.text_input("Input Sentence", "")

if input_sentence:
    words_in_sentence = re.findall(r'\w+', input_sentence.lower())
    suggestions = {}
    for word in words_in_sentence:
        if word not in V:
            suggestions[word] = my_autocorrect(word)

    if suggestions:
        st.write("Suggestions for misspelled words:")
        for word, suggestion_df in suggestions.items():
            st.write(f"Word: {word}")
            st.dataframe(suggestion_df)
    else:
        st.write("All words seem to be correct.")

if st.button("Check"):
    st.experimental_rerun()
```

- The provided Streamlit app, titled "WriteRight," allows users to input sentences for autocorrection.
- Upon entering a sentence, the app tokenizes it into words and checks each word against a predefined vocabulary.
- If a word is not found in the vocabulary, it suggests corrections using the my_autocorrect function.
- These suggestions are displayed for misspelled words. The user can click the "Check" button to rerun the auto correction process.
- Overall, the app aims to help users write flawlessly by providing auto correct suggestions in real-time.

Activity 2: Host the Application

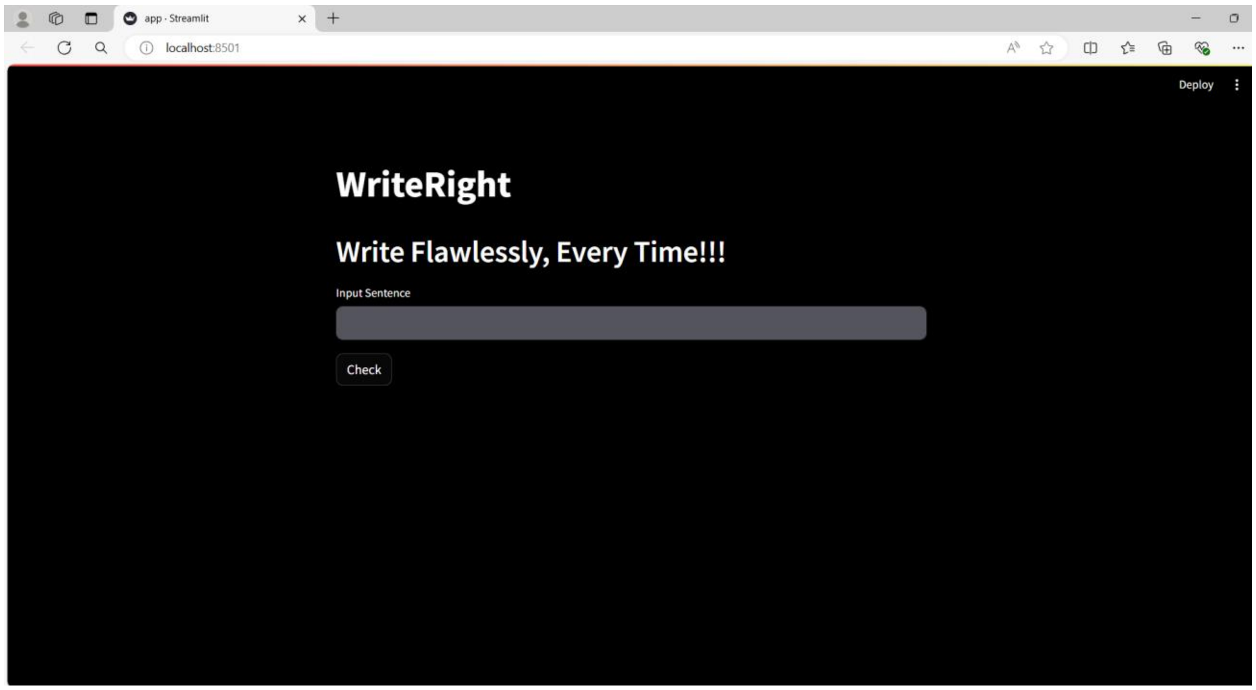
Launching the Application:

- To host the application, go to the terminal, type - streamlit run app.py
- Here app.py refers to a python script.



```
PS C:\Users\dhurvi patel\Desktop\SCIT academic\Smart_Bridge_Internship\SB_Company work\SB_Projects\SB_Project_June_Month\NLP GP> streamlit run app.py

  You can now view your Streamlit app in your browser.

  Local URL: http://localhost:8501
  Network URL: http://192.168.29.80:8501
```

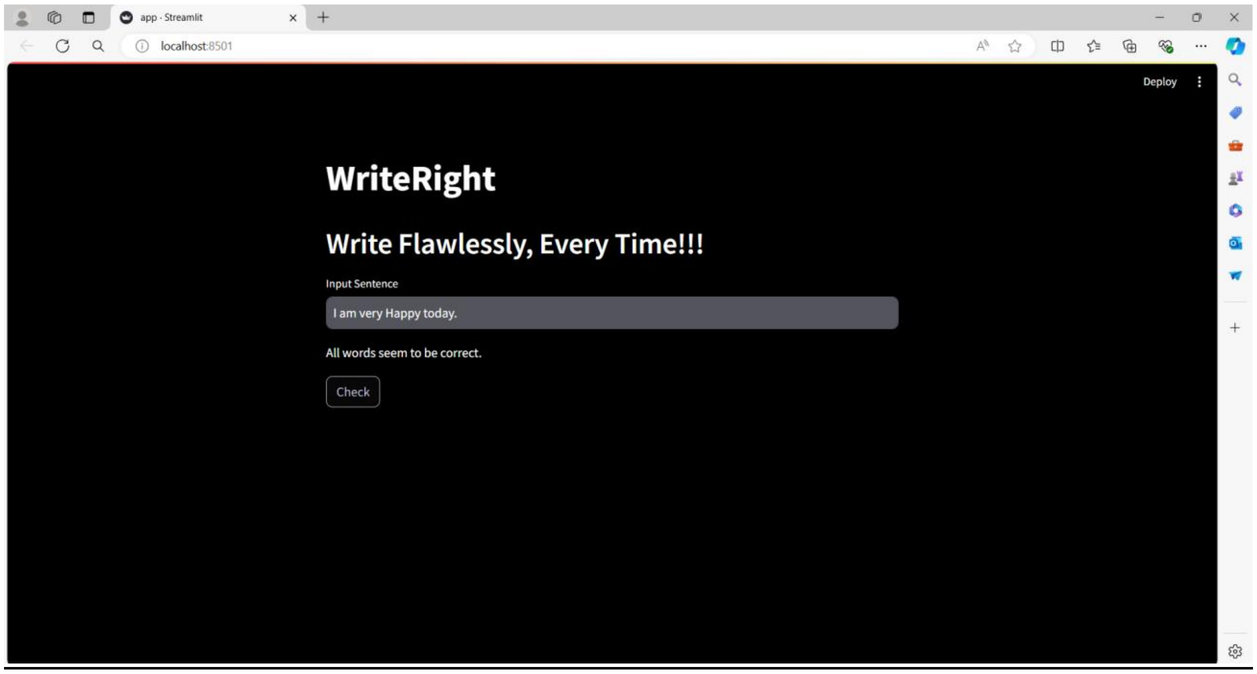
Run the command to get the below results



WriteRight

Write Flawlessly, Every Time!!!

INPUT 1



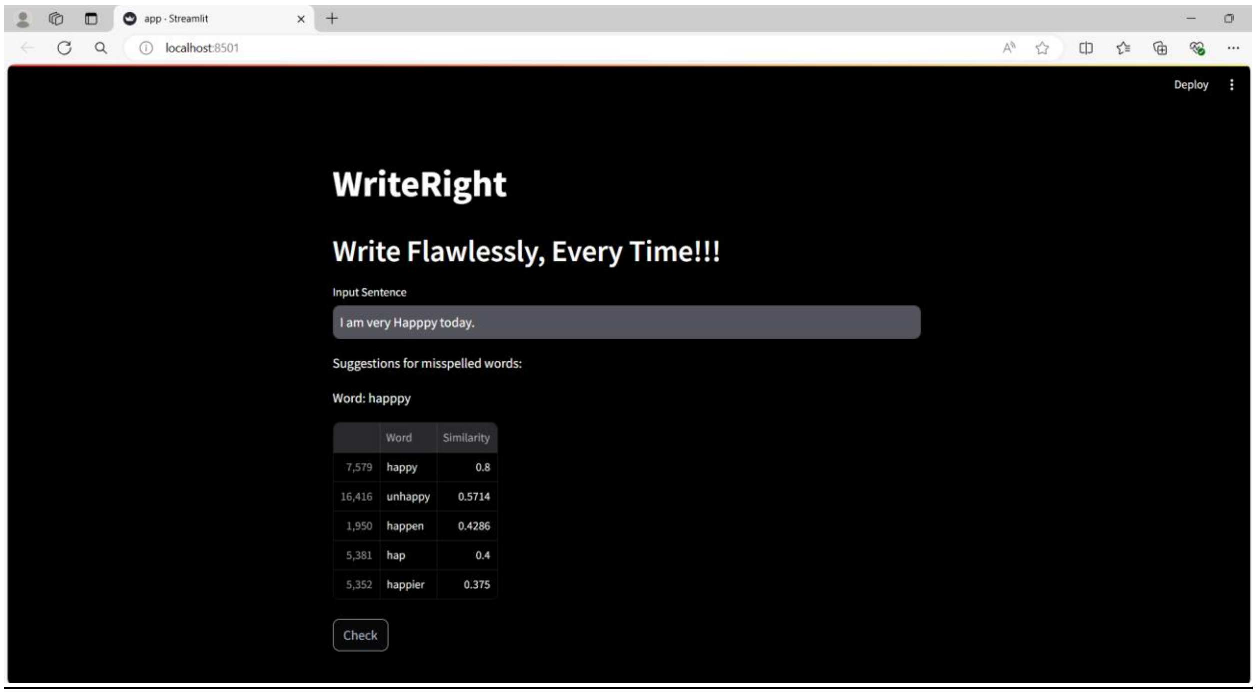
WriteRight

Write Flawlessly, Every Time!!!

I am very Happy today.

All words seem to be correct.

INPUT 2



WriteRight

Write Flawlessly, Every Time!!!

Input Sentence

I am very Happpy today.

Suggestions for misspelled words:

Word: happpy

	Word	Similarity
7,579	happy	0.8
16,416	unhappy	0.5714
1,950	happen	0.4286
5,381	hap	0.4
5,352	happier	0.375