MINI-PROJECT REPORT

AutoCorrect and Keyword Suggestion System Using NLP with both text and speech input system

Problem Statement:

In today's digital era, generating context-specific keywords is essential for content creation, optimizing search engines, and enhancing user engagement. Manually identifying the correct spelling and relevant keywords can be time-consuming and prone to error. There is a need for an intelligent system that not only autocorrects user inputs but also provides relevant, contextually appropriate keyword suggestions using NLP techniques.

This project addresses this problem by developing an intelligent system that uses text analysis, tokenization, and part-of-speech (POS) tagging to suggest relevant keywords, enhancing the overall content creation and search optimization process.

Objective:

The primary objective of this project is to develop an NLP-powered tool that:

- Automatically corrects spelling errors.
- Suggests relevant keywords based on context and part-of-speech tagging.
- Enhances user experience and reduces manual efforts in content generation.
- Improves search optimization by categorizing keywords according to their grammatical usage.

Scope:

The system offers the following functionalities:

- **Spelling Correction**: Uses a corpus-based spell-checking algorithm to suggest corrections.
- **Keyword Suggestion**: Utilizes tokenization and word embedding techniques to suggest contextually appropriate keywords.
- **POS-Based Search**: Categorizes search results based on part-of-speech (POS) tagging.
- **Voice Input**: Integrates a speech recognition system to convert voice inputs into text and process them for spell correction and keyword suggestion.

Tools and Technologies:

- **Programming Language**: Python
- **Framework**: Streamlit (for the user interface)
- **NLP Libraries**: Hugging Face Transformers, NLTK
- **Speech Recognition**: Google Speech API, SpeechRecognition package
- **Other Libraries**: Autocorrect, re (Regular expressions), collections (for Counter)
- **Corpus Data**: A text corpus file (`big.txt`) for spell correction and word probability calculations.

System Design:

Architecture Overview:

1. **User Input**:
    - Text Input: Users can type in a word or phrase to be corrected or analyzed.
    - Voice Input: Users can provide voice input, which is transcribed into text for processing.
2. **Spell Checker**:
    - A spelling correction module processes the input text and suggests corrections using word-level edits and corpus-based probability calculations.
3. **Keyword Suggestion**:
    - Based on the corrected word, the system suggests relevant keywords using tokenization and context analysis.
4. **POS Categorization**:
    - The results from the corpus search are categorized according to POS tags (Nouns, Verbs, Adjectives, etc.) using a pre-trained POS tagger from the Hugging Face library.

Modules:

- **AutoCorrect**: Corrects misspelled words by generating potential word edits and ranking them based on probability from the corpus.
- **Search Corpus**: Searches for words in the corpus and returns the surrounding context.
- **POS Tagger**: Applies part-of-speech tagging to categorize the search results

Implementation:

1. Data Loading and Preprocessing:

The project loads a large corpus file (big.txt) to be used for spell checking and word probability calculation.

 # Load the corpus and compute probabilities

```
def load_corpus(filename):
    corpus = read_corpus(filename)
    vocab = set(corpus)
    words_count = Counter(corpus)
    total_words_count = float(sum(words_count.values()))
    word_probabs = {word: words_count[word] / total_words_count for word in words_count.keys()}
    return corpus, vocab, word_probabs
```

2. Spelling Correction:

The correct_spelling function generates possible corrections for misspelled words using techniques such as deletion, insertion, replacement, and swapping of letters.

 # Spell corrector function

```
def correct_spelling(word, vocab, word_probabs):
    if word in vocab:
        return f"'{word}' is already correct."

    suggestions = level_one_edits(word) or level_two_edits(word) or [word]
    best_guesses = [w for w in suggestions if w in vocab]

    if not best_guesses:
```

```
        return f"Sorry, no suggestions found for '{word}'."

    # Rank suggestions based on probabilities
    suggestions_with_probabs = [(w, word_probabs[w]) for w in best_guesses]
    suggestions_with_probabs.sort(key=lambda x: x[1], reverse=True)

    return f"Suggestions for '{word}':\n" + "\n".join([f"{w}: {p:.4f}" for w, p in suggestions_with_probabs])
```

3. POS Tagging and Categorization:

Using Hugging Face's pre-trained POS tagger, the search results are categorized into different POS tags such as Noun, Verb, Adjective, etc.

```
# Load Hugging Face POS tagger model
pos_tagger = pipeline("token-classification", model="vblagoje/bert-english-uncased-finetuned-pos", aggregation_strategy="simple")

# Categorize search results by POS
for res in search_results:
    pos_tags = pos_tagger(res)
    # Categorize and display results by POS
```

4. Speech Recognition:

The system allows voice input for hands-free usage, which is transcribed into text for further processing.

```
def transcribe_audio():
    recognizer = sr.Recognizer()
    with sr.Microphone() as source:
        audio = recognizer.listen(source)
    try:
```

```
        text = recognizer.recognize_google(audio)
        return text
    except sr.UnknownValueError:
        return ""
```

APP.PY

```
import streamlit as st
from autocorrect.corrector import correct_spelling, load_corpus
from search.search_engine import search_corpus
from transformers import pipeline
import speech_recognition as sr

# Load corpus data
corpus, vocab, word_probabs = load_corpus('data/big.txt')

# Load Hugging Face POS tagger model
pos_tagger = pipeline("token-classification", model="vblagoje/bert-english-uncased-finetuned-pos", aggregation_strategy="simple")

# Set page config for better appearance
st.set_page_config(page_title="AutoCorrect Dashboard", layout="wide")

# Function to transcribe audio input
def transcribe_audio():
    recognizer = sr.Recognizer()
    with sr.Microphone() as source:
        st.sidebar.text("Listening...")
```

```
        audio = recognizer.listen(source)
    try:
        text = recognizer.recognize_google(audio)
        st.sidebar.text(f"Transcribed text: {text}")
        return text
    except sr.UnknownValueError:
        st.sidebar.error("Could not understand audio.")
        return ""
    except sr.RequestError:
        st.sidebar.error("Could not request results from Google Speech Recognition service.")
        return ""

# Sidebar for user input
st.sidebar.header("AutoCorrect Settings")
input_method = st.sidebar.selectbox("Choose Input Method:", ["Text Input", "Voice Input"])

if input_method == "Voice Input":
    word = transcribe_audio()
else:
    word = st.sidebar.text_input("Enter a word to check spelling:", "")

# Main title
st.title("🔍 AutoCorrect & POS Categorized Search System")

# If the user has entered a word, show suggestions and search results
if word:
```

```
    # Correct spelling
    result = correct_spelling(word.lower(), vocab, word_probabs)

    # Display the original and corrected word
    st.subheader("Spelling Correction")
    if ":" in result:
        corrected_word = result.split(":")[1].strip()
        st.markdown(f"Original Word: {word}")
        st.markdown(f"Corrected Suggestion: {corrected_word}")
    else:
        st.markdown(f"Original Word: {word}")
        st.markdown("No corrections found.")
        corrected_word = word  # Use the original word if no correction is found

    # Perform search using corrected spelling (first suggestion)
    search_results = search_corpus(corrected_word, corpus)

    # Display search results categorized by POS
    st.subheader("Search Results Categorized by POS")

    # Define the POS categories you want to include
    pos_categories = ["NOUN", "VERB", "ADJ", "ADV", "PROPN", "PRON",
"NUM", "INTJ"]

    # Create a dictionary to hold results for each POS category
    pos_results = {pos: [] for pos in pos_categories}

    if search_results:
```

```
        for res in search_results:
            # Apply POS tagging to each result
            pos_tags = pos_tagger(res)

            # Check which POS the corrected word appears as
            for tag in pos_tags:
                entity = tag['entity_group']
                if entity in pos_results:  # Check if entity is in pos_results
                    if tag['word'].lower() == corrected_word:
                        pos_results[entity].append(res)
                        break  # Exit the loop after finding the first match

        # Display results for each POS category
        for pos in pos_categories:
            st.subheader(f"Results where '{corrected_word}' is used as a {pos}:")
            if pos_results[pos]:
                for i, res in enumerate(pos_results[pos], start=1):
                    st.markdown(f"Result {i}: {res}")
            else:
                st.markdown(f"No results found where '{corrected_word}' is used as a {pos}.")
    else:
        st.warning(f"No search results found for {corrected_word}.")

# Footer for additional info
st.markdown("---")
st.markdown("### About This Tool")
st.write(
```

"This AutoCorrect & POS Categorized Search System helps you find corrections for your misspelled words and search relevant information from a corpus, while categorizing the results based on the grammatical usage of the word."

)

SEARCH_ENGINE.PY

```
# Basic search function that searches for a word in the corpus and returns surrounding words (context)
def search_corpus(word, corpus, context_size=5):
    results = []
    for i, w in enumerate(corpus):
        if w == word:
            # Extract context around the found word
            start = max(i - context_size, 0)
            end = min(i + context_size + 1, len(corpus))
            context = " ".join(corpus[start:end])
            results.append(context)

    return results if results else None
```

CORRECTOR.PY

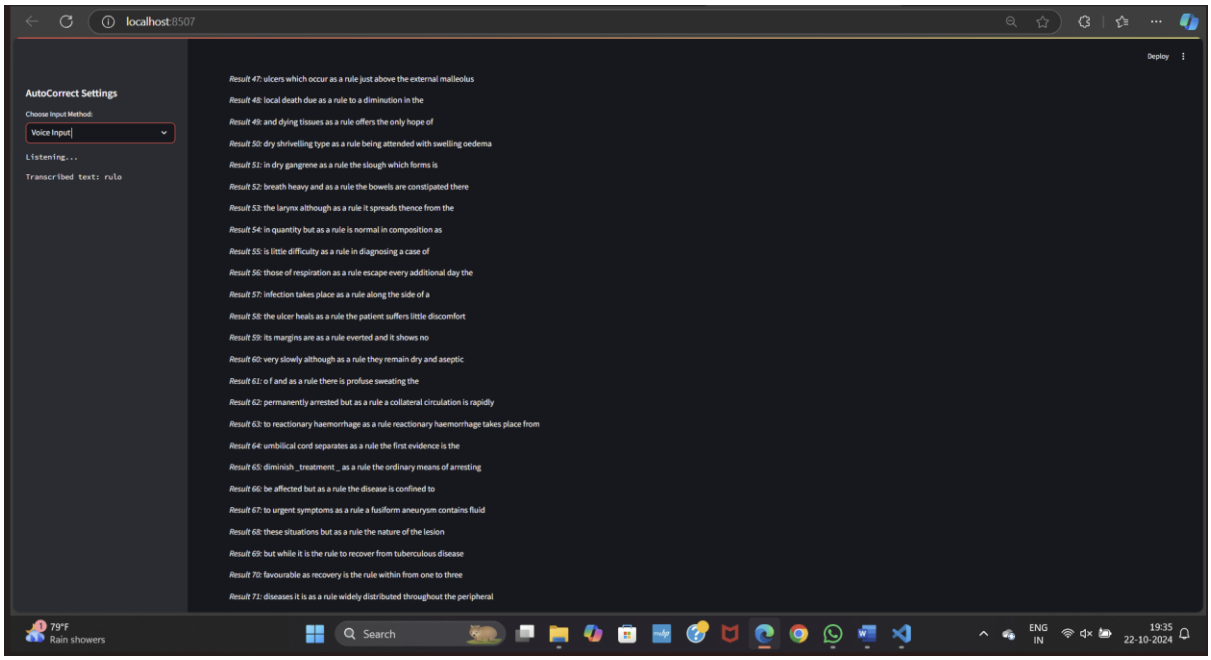
```
import re
import string
from collections import Counter

# Function to read the corpus from a text file and extract words
def read_corpus(filename):
    with open(filename, 'r', encoding='utf-8') as file:
```

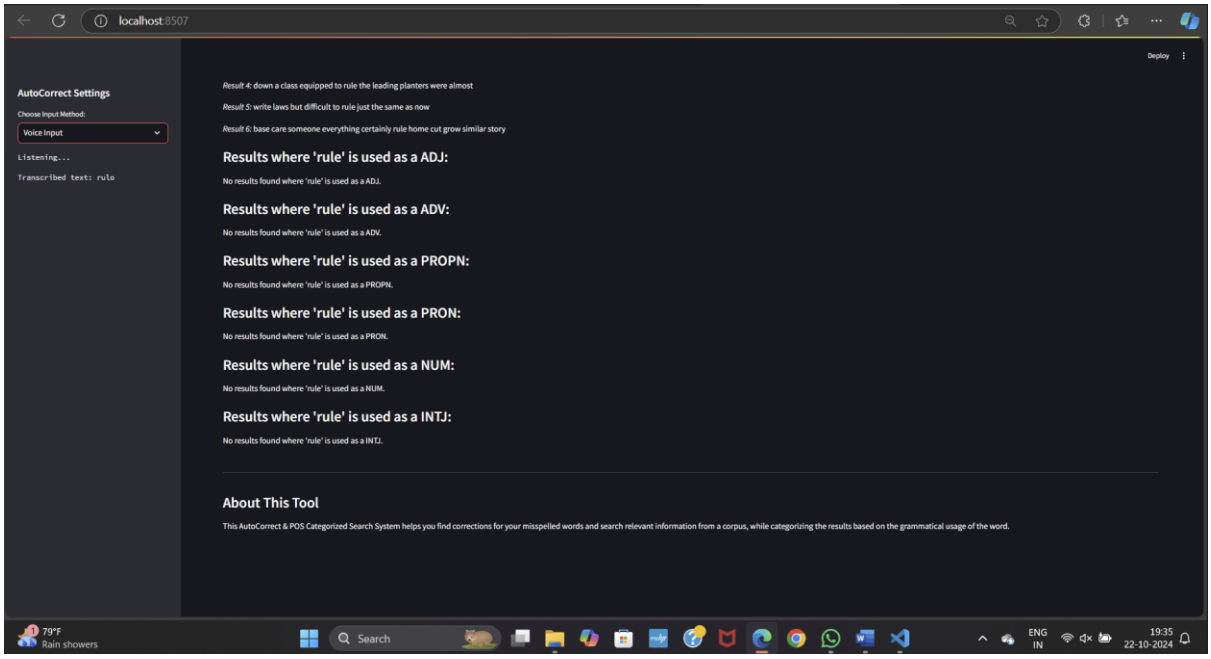
```
        lines = file.readlines()
        words = []
        for line in lines:
            words += re.findall(r'\w+', line.lower())
    return words

# Load the corpus and compute probabilities
def load_corpus(filename):
    corpus = read_corpus(filename)
    vocab = set(corpus)
    words_count = Counter(corpus)
    total_words_count = float(sum(words_count.values()))
    word_probabs = {word: words_count[word] / total_words_count for word in
words_count.keys()}
    return corpus, vocab, word_probabs

# Helper functions for generating word edits
def split(word):
    return [(word[:i], word[i:]) for i in range(len(word) + 1)]

def delete(word):
    return [left + right[1:] for left, right in split(word) if right]

def swap(word):
    return [left + right[1] + right[0] + right[2:] for left, right in split(word) if
len(right) > 1]

def replace(word):
```

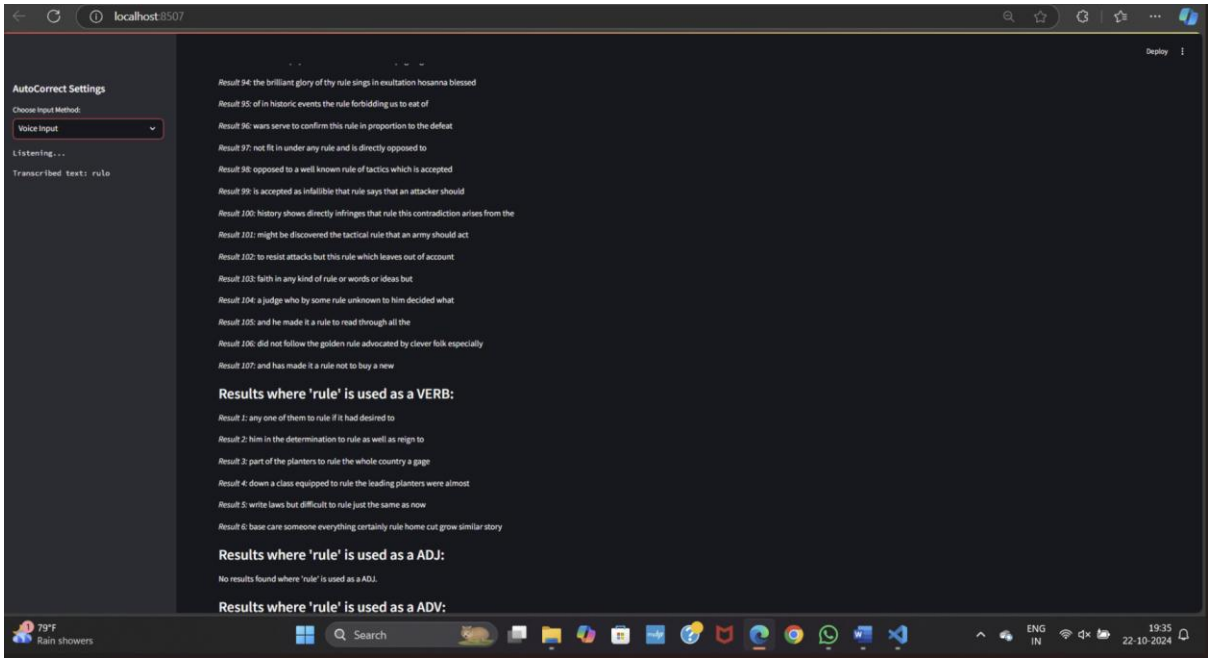
```
    return [left + char + right[1:] for left, right in split(word) if right for char in string.ascii_lowercase]

def insert(word):
    return [left + char + right for left, right in split(word) for char in string.ascii_lowercase]

# Function to generate level one edits
def level_one_edits(word):
    return set(delete(word) + swap(word) + replace(word) + insert(word))

# Function to generate level two edits
def level_two_edits(word):
    return set(e2 for e1 in level_one_edits(word) for e2 in level_one_edits(e1))

# Spell corrector function
def correct_spelling(word, vocab, word_probabs):
    if word in vocab:
        return f"'{word}' is already correct."

    # Generate suggestions
    suggestions = level_one_edits(word) or level_two_edits(word) or [word]
    best_guesses = [w for w in suggestions if w in vocab]

    if not best_guesses:
        return f"Sorry, no suggestions found for '{word}'."

    # Rank suggestions based on probabilities
```

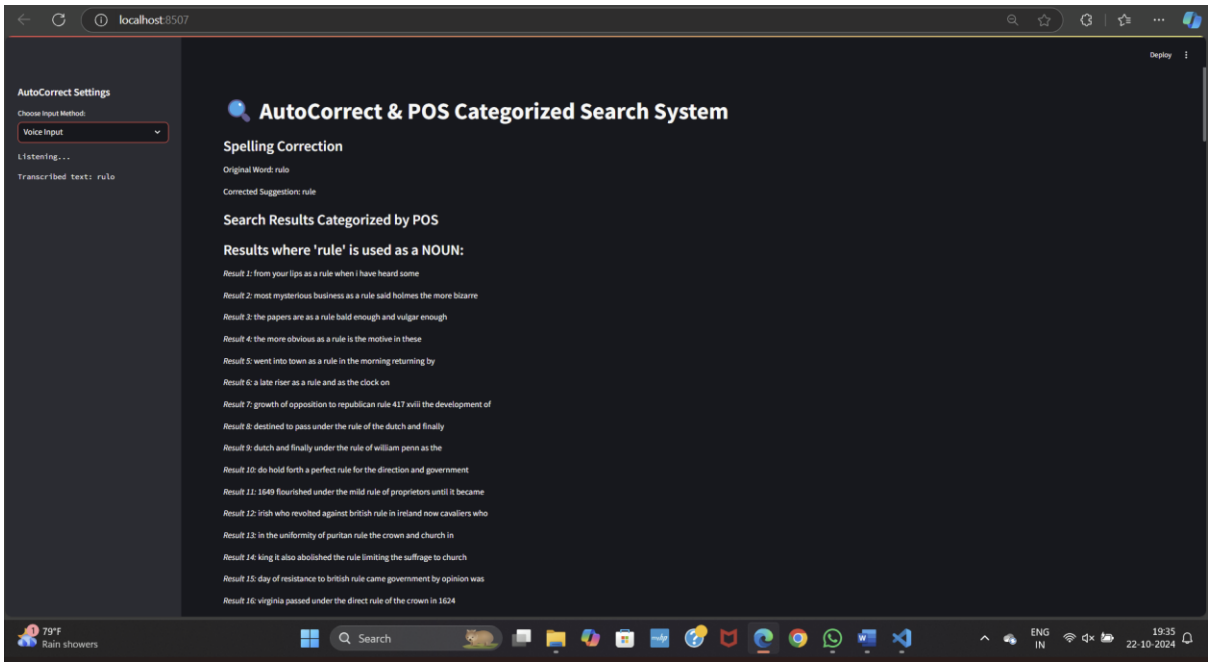
```
    suggestions_with_probabs = [(w, word_probabs[w]) for w in best_guesses]
    suggestions_with_probabs.sort(key=lambda x: x[1], reverse=True)

    return f"Suggestions for '{word}':\n" + "\n".join([f"{w}: {p:.4f}" for w, p in suggestions_with_probabs])
```

OUTPUT:






localhost:8507
Deploy
AutoCorrect Settings
Choose Input Method:
Voice Input
Listening...
Transcribed text: rulo
Result 94: the brilliant glory of thy rule sings in exultation hosanna blessed
Result 95: of in historic events the rule forbidding us to eat of
Result 96: wars serve to confirm this rule in proportion to the defeat
Result 97: not fit in under any rule and is directly opposed to
Result 98: opposed to a well known rule of tactics which is accepted
Result 99: is accepted as infallible that rule says that an attacker should
Result 100: history shows directly infringes that rule this contradiction arises from the
Result 101: might be discovered the tactical rule that an army should act
Result 102: to resist attacks but this rule which leaves out of account
Result 103: faith in any kind of rule or words or ideas but
Result 104: a judge who by some rule unknown to him decided what
Result 105: and he made it a rule to read through all the
Result 106: did not follow the golden rule advocated by clever folk especially
Result 107: and has made it a rule not to buy a new
Results where 'rule' is used as a VERB:
Result 1: any one of them to rule if it had desired to
Result 2: him in the determination to rule as well as reign to
Result 3: part of the planters to rule the whole country a gage
Result 4: down a class equipped to rule the leading planters were almost
Result 5: write laws but difficult to rule just the same as now
Result 6: base care someone everything certainly rule home cut grow similar story
Results where 'rule' is used as a ADJ:
No results found where 'rule' is used as a ADJ.
Results where 'rule' is used as a ADV:
79°F
Rain showers
Search
ENG
IN
19:35
22-10-2024


localhost:8507
Deploy
AutoCorrect Settings
Choose Input Method:
Voice Input
Listening...
Transcribed text: rulo
AutoCorrect & POS Categorized Search System
Spelling Correction
Original Word: rulo
Corrected Suggestion: rule
Search Results Categorized by POS
Results where 'rule' is used as a NOUN:
Result 1: from your lips as a rule when i have heard some
Result 2: most mysterious business as a rule said holmes the more bizarre
Result 3: the papers are as a rule bald enough and vulgar enough
Result 4: the more obvious as a rule is the motive in these
Result 5: went into town as a rule in the morning returning by
Result 6: a late riser as a rule and as the clock on
Result 7: growth of opposition to republican rule 417 xviii the development of
Result 8: destined to pass under the rule of the dutch and finally
Result 9: dutch and finally under the rule of william penn as the
Result 10: do hold forth a perfect rule for the direction and government
Result 11: 1649 flourished under the mild rule of proprietors until it became
Result 12: irish who revolted against british rule in ireland now cavaliers who
Result 13: in the uniformity of puritan rule the crown and church in
Result 14: king it also abolished the rule limiting the suffrage to church
Result 15: day of resistance to british rule came government by opinion was
Result 16: virginia passed under the direct rule of the crown in 1624
79°F
Rain showers
Search
ENG
IN
19:35
22-10-2024

Testing and Validation:

The system was tested with various inputs, including misspelled words, and was validated to ensure that the spelling correction was accurate and keyword suggestions were contextually appropriate. Additionally, POS tagging was checked to ensure correct categorization of search results.

Conclusion:

The AutoCorrect & Keyword Suggestion System successfully automates the process of correcting spelling errors and suggesting relevant keywords. By incorporating NLP techniques such as tokenization, word embedding, and POS tagging, the system significantly improves content creation workflows and search optimization efforts. The addition of speech-to-text functionality enhances accessibility and user experience, making the system versatile for a wide range of use cases.