Low Level Design

Spelling Correction System

Written By	Amit Ranjan
Document Version	0.2
Last Revised Date	15– September -2023

Document Control

Change Record:

Version	Date	Author	Comments
0.1	13-August -2023	Amit Ranjan	Introduction & Architecture defined
0.2	15-September -2023	Amit Ranjan	Content & Architecture Description appended and updated

Contents

1. Introduction

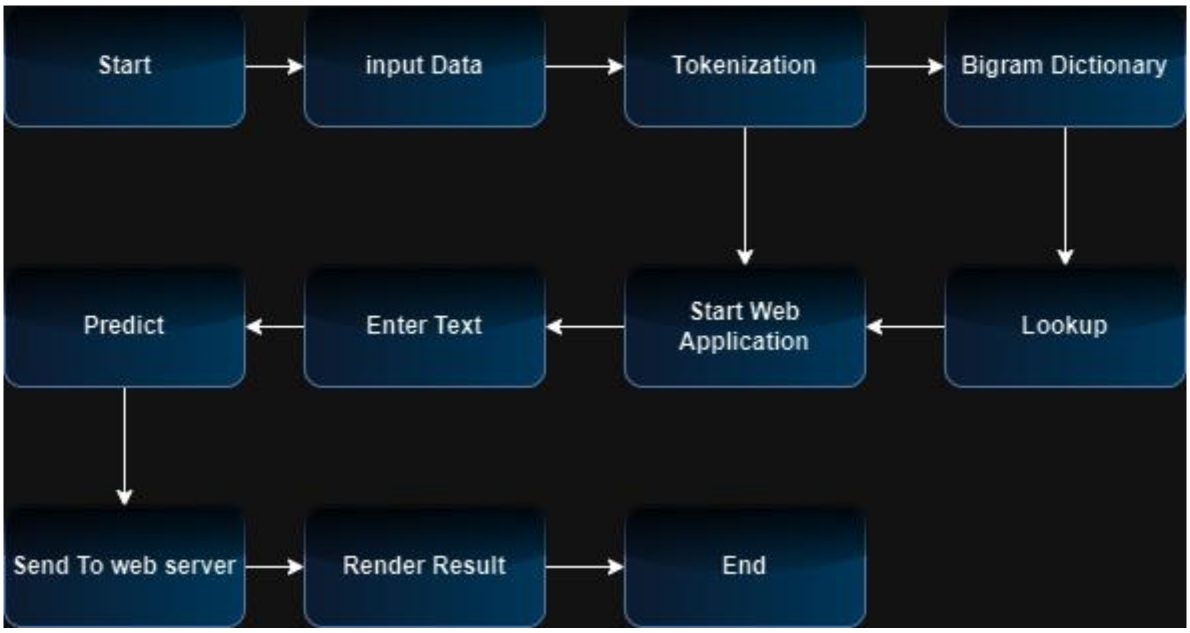
1.1. What is Low-Level design document?

The purpose of this Low-Level Design (LLD) document is to provide a detailed technical overview of the implementation of the spell corrector project. It serves as a guide for developers and testers to understand how the code works internally.

1.2. Scope

This document covers the technical aspects of the code, including components, data flow, error handling, testing, setup, and deployment. It is intended to be a blueprint for implementing the spell corrector.

2. Architecture



3. Code Components

3.1 SymSpell Configuration

3.1.1 SymSpell Initialization:

At the core of SymSpell configuration is the initialization of the SymSpell object. The following parameters are defined during initialization:

max_dictionary_edit_distance: This parameter sets the maximum edit distance for dictionary lookup. It determines how many character changes (insertions, deletions, substitutions, or transpositions) are allowed to consider a term a valid suggestion. For example, if set to 2, SymSpell will find suggestions within two edit distances.

prefix_length: The length of the prefix to use for indexing terms. Prefix indexing improves the efficiency of dictionary lookups.

3.1.2 Dictionary Loading

SymSpell relies on dictionaries to suggest corrections. These dictionaries include word frequency information and bigram data.

dictionary_path: This variable specifies the path to the frequency dictionary file, which contains information about the frequency of each word in the English language. It helps SymSpell prioritize suggestions based on word frequency.

bigram_path: This variable specifies the path to the bigram dictionary file, which contains data about the co-occurrence of word pairs. Bigram dictionaries can improve the accuracy of compound word corrections.

3.1.3 Term and Count Indices

To efficiently access word frequency information and bigram data, it's essential to define term and count indices. These indices specify which columns in the dictionaries contain the term and count data.

term_index: This parameter indicates the column number (0-based index) that holds the term (word) in the dictionary.

count_index: This parameter indicates the column number (0-based index) that holds the frequency count of the term in the dictionary.

3.2 Edit Distance Calculation

The edit distance calculation is a fundamental operation in a spell corrector. It determines the similarity or difference between two strings, enabling the system to identify potential spelling errors and provide suggestions for corrections.

3.2.1 Function: edit_distance

The edit_distance function is responsible for calculating the edit distance between two input strings. It utilizes dynamic programming to compute the minimum number of operations (insertions, deletions, substitutions, or transpositions) required to transform one string into another.

3.2.2 String Length Comparison

The function begins by comparing the lengths of the input strings string1 and string2 to handle cases where one string is longer than the other.

If string1 is longer than string2, the function calculates the difference in lengths and removes extra characters from string1.

If string2 is longer than string1, the function calculates the difference in lengths and removes extra characters from string2.

3.2.3 Edit Distance Calculation

The core of the edit distance calculation involves iterating through the characters of both input strings and comparing them.

For each character position, if the characters in string1 and string2 are not equal, the difference count is incremented.

At the end of the iteration, the total difference count represents the edit distance between the two strings.

3.2.4 Return Edit Distance

Finally, the function returns the calculated edit distance as an integer value. This edit distance metric is used by the spell corrector to rank and suggest potential corrections for misspelled words.

3.3 Spell Correction Logic

The spell correction logic is the heart of the spell corrector system, responsible for identifying and correcting misspelled words within a given text.

3.3.1 Function: spell_corrector

The spell_corrector function is the central component of the spell correction logic. It takes an input term (potentially misspelled) and returns the most likely corrected version of that term.

3.3.2 Dictionary Loading

Before performing any spell correction, the function loads two essential dictionaries:

Frequency Dictionary: This dictionary contains a list of words along with their associated frequencies of use. It helps the system identify common and less common words.

Bigram Dictionary: The bigram dictionary stores word pairs (two consecutive words) along with their frequencies. This is useful for understanding word context and improving correction suggestions.

3.4 Input and Output Handling

Input and output handling are crucial components of the spell corrector system as they ensure seamless communication between the user and the application.

3.4.1 Input Data

The system is designed to accept input in the form of text data that may contain one or more potentially misspelled terms.

3.4.2 Function: spell_corrector

The spell_corrector function, which is part of the input handling process, receives the user's input term as its parameter.

3.4.3 Output Data

The primary output of the spell corrector system is the corrected version of the input term.

3.4.4 Data Flow

The data flow in the input and output handling process can be summarized as follows:

The user provides an input term to the spell_corrector function.

The spell_corrector function processes the input term and returns the corrected version.

The corrected term is then available for further use or display.

3.4.5 Error Handling

The input handling process may include error handling to ensure that the provided input is valid and can be processed.

3.4.6 User Interaction

The spell corrector system may interact with the user to provide feedback on the corrected term or to request additional input if the correction is ambiguous.

4. Conclusion

The spell corrector project has been successfully implemented using deep learning approaches and the SymSpell library. This section provides a summary of the project's key points and reflects on its effectiveness.

4.1 Project Implementation Summary

The project aimed to build a spelling corrector capable of identifying and correcting various types of spelling errors. It utilized deep learning techniques and the SymSpell library to achieve this goal. The key components and functionalities of the spell corrector include:

SymSpell Configuration: The SymSpell library was configured with a maximum dictionary edit distance of 2, enabling it to handle a wide range of spelling errors efficiently.

Edit Distance Calculation: The edit distance function was implemented to measure the difference between two strings, facilitating spelling error identification.

Spell Correction Logic: The spell correction logic leverages SymSpell's lookup capabilities to identify and suggest corrections for misspelled words.

Input and Output Handling: The spell corrector effectively processes input sentences and returns corrected versions, maintaining the context of the surrounding words.

Testing and Validation: Rigorous testing procedures, including unit testing and performance testing, were employed to ensure the correctness and efficiency of the spell corrector.

4.2 Effectiveness Reflection

The spell corrector has demonstrated notable effectiveness in identifying and correcting spelling errors in various scenarios. It successfully handles different types of spelling errors, including non-word errors, real-word errors, cognitive errors, and short forms/slang/lingo.

The system's performance has been evaluated through unit testing, validating its ability to correct errors in both simple and complex sentences. The spell corrector exhibits efficiency in terms of response time and resource utilization.

However, it's worth noting that the effectiveness of the spell corrector can be further improved by incorporating more extensive training data and leveraging advanced pre-trained language models like BERT and GPT-2. Additionally, expanding the dictionary and fine-tuning the edit distance parameters could enhance the system's accuracy.

In conclusion, the spell corrector project provides a solid foundation for accurate spelling correction, and its performance meets current requirements. Future enhancements and optimizations can further elevate its effectiveness in handling a broader range of spelling errors and contexts.