WRITING AID TO AVOID REPETITIONS

MR.ANNOP PORNSAWATDIPAK
MR.THANAPATR AMNUAYWIT

A PROJECT SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR
THE DEGREE OF BACHELOR OF ENGINEERING (COMPUTER ENGINEERING)
FACULTY OF ENGINEERING
KING MONGKUT'S UNIVERSITY OF TECHNOLOGY THONBURI
2022

Writing Aid to Avoid Repetitions

Mr.Annop Pornsawatdipak
Mr.Thanapatr Amnuaywit

A Project Submitted in Partial Fulfillment
of the Requirements for
the Degree of Bachelor of Engineering (Computer Engineering)
Faculty of Engineering
King Mongkut's University of Technology Thonburi
2022

Project Committee

.... (Assoc.Prof. Nuttanart Muansuwan Facundes, Ph.D.)	Project Advisor
.... (Asst.Prof. Committee2, Ph.D.)	Committee Member
.... (Asst.Prof. Committee3, Ph.D.)	Committee Member

Project Title	Writing Aid to Avoid Repetitions
Credits	3
Member(s)	Mr.Annop Pornsawatdipak Mr.Thanapatr Amnuaywit
Project Advisor	Assoc.Prof. Nuttanart Muansuwan Facundes, Ph.D.
Program	Bachelor of Engineering
Field of Study	Computer Engineering
Department	Computer Engineering
Faculty	Engineering
Academic Year	2022

Abstract

The computer-assisted writing aims to help improve the quality of a piece of text with the help of computer. Most of the tools focus on fixing grammatical accuracy and misspelled words, only few of them offer repetition detectors. In this project we will be creating a program to keep track of the words and phrases frequency. Moreover, our program will suggest alternative words that fits in original context, thus, the writer can replace the word marked as repetitive with a new suggested word instead. Our proposed solutions consisted of two main parts, the detector, and the suggestor. In the first part, the program tries to detect only relevant repetitions and filter out very specific words or very trivial words. Next, the suggestor module will generate multiple substitution candidates by feeding the sentence into the Transformer model. The transformer model such as BERT is trained with Masked Language Modelling, which we can use to predict the mask position, in this case the target word to replace. The generated candidates are verified using SentenceBERT and similarity score to ensure each of them won't alter the meaning of the original sentence.

Keywords: Overused word detector / repetitive word detector / BERT / NLP

หัวข้อปริญญานิพนธ์	โปรแกรมช่วยเขียนโดยหลีกเลี่ยงคำซ้ำ
หน่วยกิต	3
ผู้เขียน	นายอรรณพ พรสัวสดิภักดิ์ นายธนภัทร อำนวยวิทย์
อาจารย์ที่ปรึกษา	ผศ.ดร.ณัฐนาถ เหมือนสุวรรณ
หลักสูตร	วิศวกรรมศาสตรบัณฑิต
สาขาวิชา	วิศวกรรมคอมพิวเตอร์
ภาควิชา	วิศวกรรมคอมพิวเตอร์
คณะ	วิศวกรรมศาสตร์
ปีการศึกษา	2565

บทคัดย่อ

การเขียนโดยใช้คอมพิวเตอร์เป็นตัวช่วยมีจุดมุ่งหมายเพื่อช่วยปรับปรุงคุณภาพของข้อความโดยใช้คอมพิวเตอร์เข้ามาช่วยเหลือ เครื่องมือส่วนใหญ่มุ่งเน้นไปที่การแก้ไขความถูกต้องทางไวยากรณ์และคำที่สะกดผิด มีเพียงไม่กี่เครื่องมือเท่านั้นที่มีระบบตรวจจับคำซ้ำ ในโครงการนี้ เราจะสร้างเครื่องมือเพื่อติดตามความถี่ของคำและวลี นอกจากนี้ โปรแกรมของเราจะแนะนำคำทดแทนที่เหมาะกับบริบทเดิม ดังนั้นผู้เขียนสามารถแทนที่คำที่ถูกทำเครื่องหมายว่าซ้ำด้วยคำใหม่ที่แนะนำแทน แนวทางที่เรานำเสนอประกอบด้วยสองส่วนหลัก ได้แก่ ตัวตรวจจับ และตัวแนะนำ ในส่วนแรก โปรแกรมจะพยายามตรวจหาเฉพาะคำซ้ำที่เกี่ยวข้องและกรองคำเฉพาะหรือคำที่ไม่สำคัญออก ถัดไป โมดูลผู้แนะนำจะสร้างชุดคำทางเลือกสำหรับทดแทน โดยป้อนประโยคลงในโมเดล Transformer โมเดลทรานสฟอร์เมอร์ เช่น BERT ได้รับการฝึกฝนด้วย Masked Language Modeling ซึ่งเราสามารถใช้มันเพื่อทำนายตำแหน่งของมาสก์ ในกรณีนี้ คำเป้าหมายที่จะแทนที่คำทดแทนที่สร้างขึ้นจะได้รับการตรวจสอบโดยใช้ SentenceBERT และคะแนนความคล้ายคลึงกันเพื่อให้แน่ใจว่าแต่ละคำจะไม่เปลี่ยนความหมายของประโยคเดิม

คำสำคัญ: การตรวจจับคำที่ใช้บ่อย / การตรวจจับคำซ้ำ / BERT / NLP

CONTENTS

PAGE

LIST OF FIGURES

FIGURE **PAGE**

CHAPTER 1 INTRODUCTION

1.1 Problem statement and Approach

Writing a long text such as an essay, or an academic report can be challenging, especially for non-native English speakers. One of the most common problems is the overuse of certain vocabulary, which can distract the readers and make the article less professional. With limited lexical resources, the writing may sound boring and can potentially weaken the message the writer wants to convey. Particularly in academic writing where the readers are highly educated, repetition is an undesirable trait and can be seen as disrespectful to the readers (Fosu, 2020[1]). This project will focus on helping non-native users of English to reduce repetitions in their writing which will improve the overall quality of their work.

1.2 Objectives

To detect overused words or phrases in writing and suggest alternatives based on context.

1.3 Project Scope

- Overused word detection for detecting high-frequency words in the document.
- Word suggestion for recommending a substitution word that fits in the context.
- Suggestion does not include grammatical corrections, such as verb tense, spelling, etc.
- Supports English language only

1.4 Project Schedule

Term 1

Task/Week	August				September				October				November				December			
	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
Learn about the topic																				
Discuss topic idea with advisor		🟦																		
Background and theory reseach		🟦	🟦																	
Project IDEA document			🟦																	
Research for information																				
Further research related theories			■	■																
Explore different tools and libraries			■	■																
Project proposal document				■	■															
Proposal presentation						■														
Make a report																				
Project report document																🟩				
Report presentation																🟩				
Design & Prototype																				
Gather requirements						🟨	🟨	🟨												
System architecture and pipeline									🟨	🟨	🟨	🟨								
UI mockup									🟨	🟨	🟨	🟨								
ML dataset collection								🟨	🟨	🟨	🟨	🟨								
Implement text preprocessor (beta)													🟨	🟨	🟨					
Term 1 Presentation																				
Give a presentation																	🟥			
Scope adjustment																	🟥	🟥		

Term 2

Task/Week	January				February				March				April				May			
	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
Core feauture: detection																				
Implement Text preprocessing	🟨																			
Implement Name entity extraction		🟨	🟨	🟨																
Implement Keyword extraction		🟨	🟨	🟨																
Core Feauture: suggestion																				
ML dataset collection					🟧	🟧														
ML model training						🟧	🟧													
Implement word suggestion							🟧	🟧												
Core Feature: finalization																				
Combine detection and suggestion									🟪											
Implement application UI										🟪										
Optimize the pipeline											🟪	🟪								
Testing																				
Test code				🟪				🟪					🟪	🟪	🟪	🟪				
Find and fix bugs													🟪	🟪	🟪	🟪				
Delivery																				
Project completed																	🟩			
Final presentation																	🟩			

Term 1

1. Learn about the topic
   - Discuss topic idea
   - Define problem definition, scope and method
   - Make the IDEA document
2. Research for more information
   - Further research related theories
   - Explore different tools and libraries
   - Make a final proposal and presentation
3. Design & prototyping
   - Gather requirements
   - Design system architecture and pipeline
   - Create an UI mockup
   - Try implementing text preprocessor
4. Term 1 final report and presentation

Term 2

1. Core feature: detection
   - Implement Text Preprocessing
   - Implement Name Entity Extraction

- Implement Keyword extraction

2. Core feature: suggestion
   - Finish up ML datasets collecting process
   - ML model training
   - Implement word suggestion
3. Finalize the application
   - Combine detection and suggestion
   - Optimize the pipeline
   - Implement application UI
4. Testing
   - Application testing
   - Discover problems and errors
   - Fixing bugs
5. Delivery
   - Complete the project
   - Final report and presentation

1.5 Deliverables

Term 1

1. Proposal
2. Project report(unfinished) and presentation
3. Requirement list
4. Text preprocessor (beta)
5. UI mockup
6. System architecture

Term 2

1. Completed application
2. Completed project report

CHAPTER 2 BACKGROUND THEORY AND RELATED WORK

2.1 Background

Writing an academic article in English poses a number of challenges to the writer, particularly those whose English is not their first language for example ESL students (English as Second Language) and EFL students (English as Foreign Language). There are a number of difficulties that often come up when writing a long piece of tex;t namely sentence structural problems, grammatical mistakes, coherence and cohesion. Nonetheless, one of the most prominent struggle writers face is having limited lexical resources which leads to the repetitive use of a certain word. The research regarding connector usage of Japanese EFL learners found that learners significantly overuse some connecting devices, especially sentence-initial positions. It also revealed that both native users and Japanese EFL students share a common set of high-frequency linking devices (Narita et al., n.d.). The researchers suggested some of the reasons for this problem include the lack of familiarity with the word and inadequate knowledge. Another investigation (Hama, 2021) supports the claim by pointing out a familiarity issue with certain connecting words which ultimately resulted in repetitive use of linking words.

Repetition in an academic context is especially critical due to the audience being highly knowledgeable. As Fosu (2021)[1] said, “Not only is repeating things distracting, but it's also somewhat insulting to a person's intelligence”. In certain scenarios, the issue arises due to the mindlessness of the writer rather than a lack of skill. As the text becomes longer, it is harder to keep track of how often a certain vocabulary has been used in the text, thus the repetition becomes more difficult to control. The writer can certainly read through the entire document to identify overuse words, however, this is a daunting task and time-consuming to re-read multiple times.

Natural Language Processing (NLP) has been greatly advanced in recent years both in terms of efficiency and accuracy. A state-of-art machine learning-based NLP has emerged thanks to tremendous textual data human generates each day. Currently, there are numerous services focused on computer-assited writing, for instance, Grammarly, Prowritingaid, Gramara, Microsoft Word's Editor, Writer.com, and more. Most of them focus on fixing grammatical errors, typos, and punctuation, only a few of them offer overuse word detection. As mentioned above, we decided to create an application to help writers avoid repetition by keeping track of their word frequency as well as suggesting alternative words to use. The target user of this project is non-native English student writers in an academic context.

2.2 Related Theory

2.2.1 Natural Language Processing

Natural language Processing (NLP) is a sub-field of computer science (also a branch of Artificial intelligence depending on the technique used) defined as a way to give computers the ability to understand text and spoken language in the same way human beings can (IBM, 2020 [2]). According to IBM, NLP is a multidisciplinary combining computational linguistics, rule-based modeling of human language, statistics, machine learning, and deep learning models. All of these enable computers to interpret natural language in both textual and verbal forms. NLP has various use cases both for personal and business use such as machine translators, voice-activated speakers, virtual assistants, virtual agents, customer service chatbots, and social media analysis respectively. Early, NLP applications often rely on a rule-based approach which means the programmer must hard code all the linguistic rules to let the computer perform NLP tasks. However, human language is full of ambiguity and exceptions which makes traditional methods cannot scale to capture all the nuance of everchanging human language. Thus, most modern-day NLP applications let the computer learn patterns by themself using Machine Learning instead.

2.2.1.1 Part-of-speech Tagging

A process of assigning the part of speech to different words in a piece of text. Part of speech is critical as some words can be both verb and noun depending on the usage for example “ship” in the product has been shipped or “ship” in the ship will not float.

2.2.1.2 Word sense disambiguation

A method to determine which meaning to use in a certain context in case the word has more than one meaning. For example, “park” could mean a large public green area or bring a vehicle to a halt and leave it temporarily. This can be done through semantic analysis.

2.2.1.3 Name entity recognition

A technique used to automatically identify name and location in the given text, for instance, Steve Jobs was kicked out of Apple, here, Steve Jobs is a person, and Apple is a company.

2.2.1.4 Sentiment analysis

An attempt to interpret the tone and attitude of the text. Useful for detecting emotions, sarcasm, confusion, and suspicion in the text.

2.2.1.5 Natural Language Generation

A process where computers create textual information that humans can understand.

2.2.1.6 Keyword Extraction

A method used to automatically find the most relevant words and phrases from a piece of document. Allowing the computer to know the main idea of the text. Some of the most well-known techniques (Pradeep, 2022) are KeyBERT, Rapid Automatic Keyword Extraction (aka. Rake), Yet Another Keyword Extractor (aka. YAKE), and TextRank.

2.2.2 Pointwise Mutual Information

Pointwise Mutual Information (PMI) is a technique used to analyze the association between words. PMI asks how much more the two words co-occur in our corpus than we would have a prior expected them to appear by chance (Jurafsky, 2021)[3]. PMI permits the ability to detect word pairs that occur together, thus, is useful when performing a cooccurrence analysis of words in the corpus. The formula is as follows.

2.2.3 Likelihood ratio

According to packtpub.com(n.d.), likelihood ratio or log-likelihood ratio is a measure of how two events are unlikely to be independent but occur together more than by chance. This is used to measure the level of association between tokens, aka. bigram, trigram. A higher score indicates a significant co-occurrence between those tokens. Unlike PMI, this metric is not biased toward low-frequency words.

2.2.4 Byte-pair encoding (BPE)

A type of tokenizer where the word is broken down into tokens each containing a single character. A token can be merged to form a larger group. Similarly, each group can also be joined together to form a word. Example: This is tokenizing -> T h i s i s t o k e n i z i n g -> this is token izing

2.2.5 Attention

Traditional sequence to sequence approaches such as LSTM and RNN tend to suffer from the vanishing gradient problem, where the model failed to capture long term dependencies in a lengthy text. The attention model comes to solve this problem as it permits the model to focus on only the important information of the input. Instead of encoding the entire input sentence into a fixed-sized vector, the model learned to attend to parts of the sentence that are relevant to the output it will produce. The decoder performs additional before outputting the result which is 1) look at each hidden state received from the encoder, 2) assign each of them a score, 3) apply softmax to each score, thus, higher scores will be amplified, the lower scores will be suppressed.

2.2.6 Transformer

According to the paper called "Attention is all you need", published in 2017, the author purposed an encoder-decoder architecture based on attention layers which was later known as the transformer[4]. Initially, the transformer is intended to be used in the translation field, but due to its performance, scalability and versatility, this method is widely adopted by the research community and enterprise to solve challenging NLP tasks. Unlike LSTM or RNN, where it can only process words in a sentence sequentially as dictated by the design. The encoder-decoder architecture processes input in parallel manners.

2.2.6.1 Input Embedding [1]

Textual input must be converted into a vector or in NLP, we called it an embedding. Every word is represented as a vector of values corresponding to its meanings.

2.2.6.2 Positional Encoding [2]

Due to its parallelism property, each word in a sentence will get through the model simultaneously which means it doesn't know what order they are in. The positional encoding assigns each embedding a vector denoting its position.

2.2.6.3 Multi-headed attention [3]

This mechanism allows the model to pay attention to a specific token in the input. The attention vector captures the relationship between each word to the other words in the sentence. In self-attention, the model will give more weight to itself. However, this is not ideal because we are interested in obtaining the interaction between words. So instead, there are multiple attention heads per word which will be averaged out to get the final attention vector for every word.

2.2.6.4 Add and Norm [4]

Normalization is performed on each output layer-wise.

2.2.6.5 Feed Forward [5]

A feed-forward neural network is applied to every attention vector. This will transform the attention vector into a suitable format for the decoder/encoder block. All the vectors are passed to the network parallelly.

2.2.6.6 Masked multi-headed attention [6]

The multi-headed attention block generates attention vectors for every word in the sentence and the take average to get the final attention vector denotes the relationship between words. This resembles multi-headed attention but now included masks. The reason is to prevent the model from having access to the future token that has not appeared yet in the natural ordering of the sentence. Mask is essentially to replace those positions with zero. Thus, the attention score will be computed in accordance with the previous token only and not the future one.

2.2.6.7 Multi-head attention [7]

The attention vectors and the vector from the encoder are passed to the block similar to the previous one except now the input is coming from the encoder. So, we may refer to it as an encoder-decoder attention block. This block contains vectors for both languages and this is where the mapping between the 2 languages takes place. The output is in a form of attention vectors representing the relationship of words from both languages.

2.2.6.8 Linear [8]

A kind of feed-forward network expands the multi-headed attention output to the dimension of size equal to the vocabulary in the target language.

2.2.6.9 Softmax [9]

Calculate the probability score ranging from 0 to 1 for every class (words in the language). The class with the highest probability will be selected as the final output.

2.3 Technologies and Development Tools

2.3.1 Python

Python is an object-oriented programming language that supports a wide selection of libraries including PyTorch, Spacy, NLTK, etc.

2.3.2 PyTourch

A machine learning library written in python.

2.3.3 Natural Language Toolkit (NLTK)

A toolkit provides some of the basic operations in NLP including stemming, tokenization, cooccurrence analysis, etc.

2.3.4 Spacy

A more advanced NLP library shipped with several ready-to-use tools such as tokenizer, NER, POS, Lemmatizer, and Training pipeline.

2.3.5 Tkinter

A GUI builder for Python applications. Provide basic tools to create GUI elements such as a window, a button, an input field, etc.

2.3.6 Visual Studio Code

A code editing software from Microsoft. Supports a wide range of plugins and add-ons which help streamline the development process. Collaborative code editing allows teammates to work on the same project remotely anytime anywhere.

2.3.7 Matplotlib

A library to help visualize results as a graphical representation such as confusion matrix, correlation plot, performance analysis, etc.

2.3.8 Huggingface Transformer

An open-source library dedicated to the development of NLP applications using transformers. It is compatible with both Tensorflow and PyTorch. The package includes numerous state-of-the-art models including BERT, Roberta, XLNet, T5, and more. It also supports various NLP pipelines for performing different tasks out-of-the-box some of them are audio classification, question answering, summarization, and translation. Moreover, this library provides a range of utility functions for training, finetuning, inferencing, etc.

2.4 Related Research

2.4.1 Embedding Dropout

As mentioned, BERT has been trained using masked language modeling, it is capable of predicting a masked token of the sentence. However, if we feed the sentence where a word is replaced with <masked> token, the predicted token will likely fit in the context but may not retain its original meaning. According to Zhou, et al. [5], the embedding dropout can be used to overcome this issue. It is the technique where the random index of the token will be set to zero, thus, it allows the model to get partial information about the target word but help avoid overfitting. This approach helps improve the performance of the prediction according to the researcher. Further evaluation of performance when implemented in our project is needed.

2.4.2 Sentence Concatenation

According to the article, A simple BERT-Based Approach for Lexical Simplication purposed by Jipeng et al[6]. The research focuses on simplifying complex words, although not directly related, this method is applicable to our project. The authors state that considering the complex word w in sentence S, the word w is masked to create a new sentence S' and feed into the model. By doing this, the model will not consider the influence of the complex word. To give some clue about the target word, the sentence with a masked token will be concatenated with the original sentence that has no mask. Both sentences are then passed into the model to get the prediction. This allows the model to get some contexts about the target words, therefore, the output will be relatively close to the target word.

2.4.3 BERT

Based on Bidirectional Encoder Representations from Transformer (aka. BERT), A machine learning-based approach purposed by Google AI Language (Horev, 2018)[7]. It achieved superior performance by reading the entire sequence of words at once, instead of left to right or right to left like most of the previous works. This allows the model to learn the context of words using clues from their surroundings. BERT has been trained using 2 objectives in mind. The first is called next-sentence prediction where it tries to predict whether the given sentence will follow the previous sentence or not. Next, the masked language model where the random part of the corpus is masked and the model tries to fill in the gap. This research has become one of the best-performing approaches in the NLP world. Despite that, the model itself is inefficient and slow to run due to the complexity, so in this project, we decided to use another model named Roberta. At its core, Roberta is similar to BERT but without the Next sentence prediction objective, which noticeably improves the execution speed compared to the original BERT.

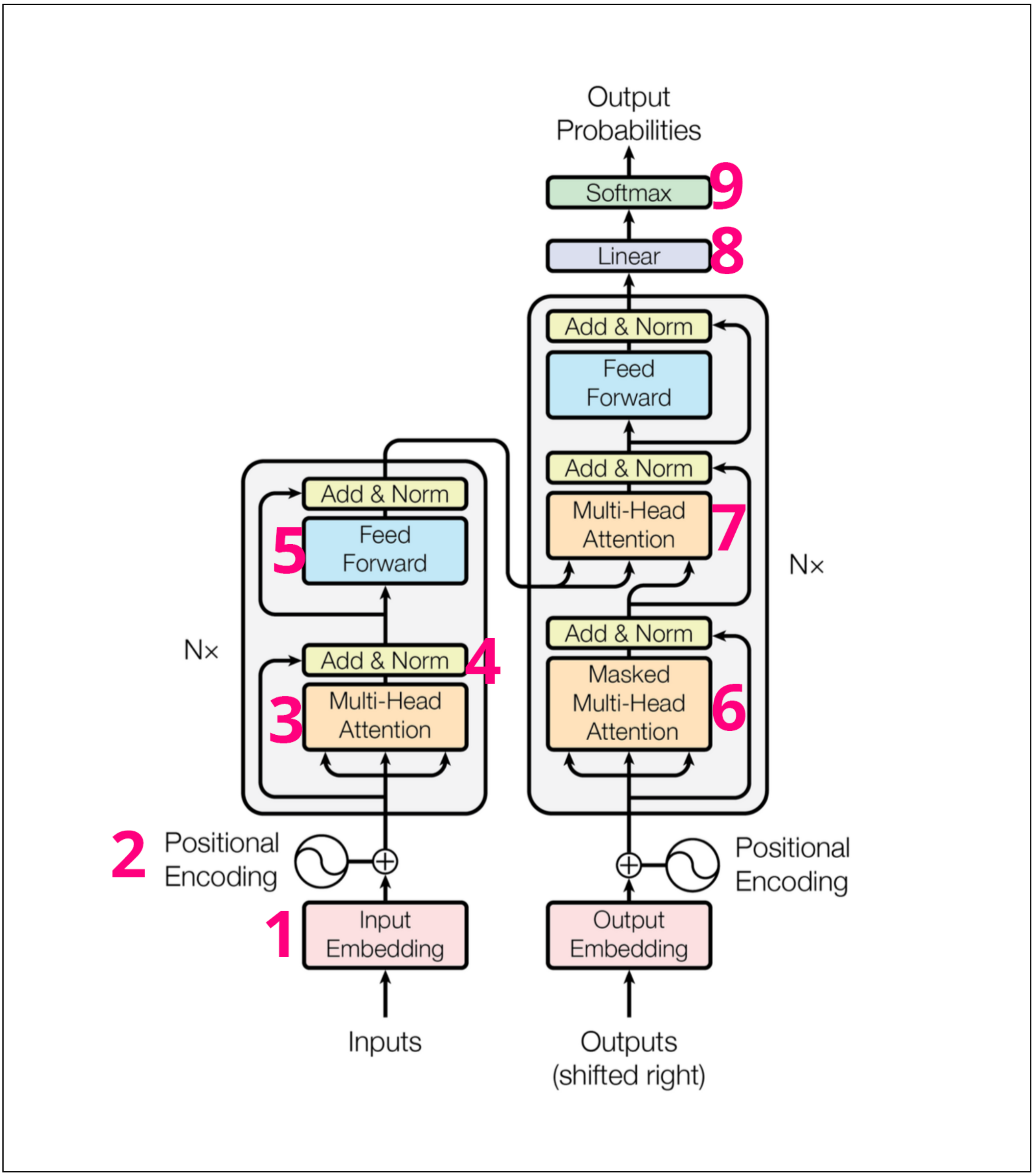


Figure 2.1 The network model

CHAPTER 3 CHAPTER 3 DESIGN AND METHODOLOGY

3.1 System Architecture

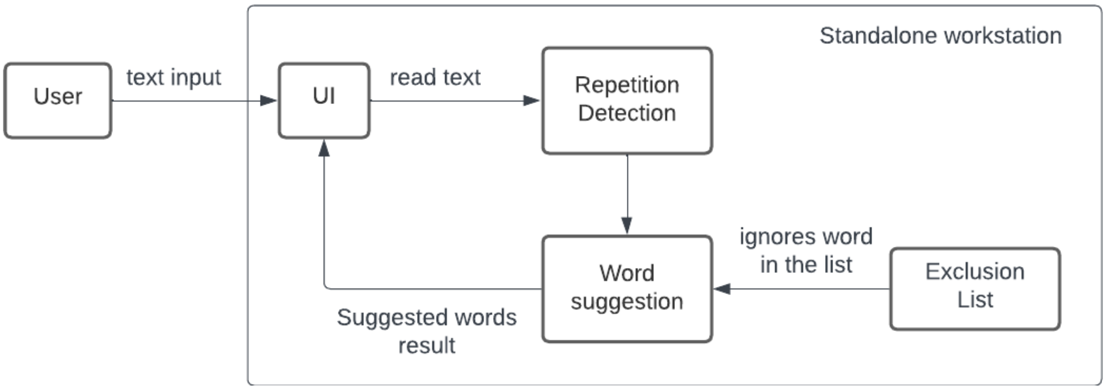


Figure 3.1 System Architecture of an application

Figure 3.1 shows an application architecture in a diagram. The system is a standalone program, There's basically just user interaction with the application side, and some general process of repetition search and word suggestion logic. After the user inputs text by typing and/or importing from other sources by copy and paste the text into the application text input area. The program will read those texts and start a repetition search by counting word occurrences. After searching, the text data will be sent to another software's part which is the word suggestion part, working along with a trained text model. And return matched words excluded from the exclusion list given from the user, that can be replaced with those repetition whether it has or not.

3.2 Feature Lists

3.2.1 Editor

Our program's basic function is to allow users to type in texts or paste them from other sources, so the program can use these texts with other functions.

3.2.2 Repetition highlights

As the name said. The program should highlight the repetitions for the entire text input that user gives, and to not let the user get dazzled when seeing those repetitions in case there are many repeated words in the document. Only the first word or every repetition word will be highlighted, and only shows all the repetition locations when the user hovers or clicks on it.

3.2.3 Repetition replacement suggestion

When a user clicks on any repetition words found, a list of similar or the word that can be replaced will be shown on the UI for the user to make a decision.

3.2.4 Word exception list

In case that user doesn't want this specific word to be detected as a repetition whether in this document or others. They can select the word as an exception. And this list of exceptions can be decided to use in other documents, use for individual words, or not used at all.

3.3 Use case Diagram and Sequence Diagram

3.3.1 Use case diagram

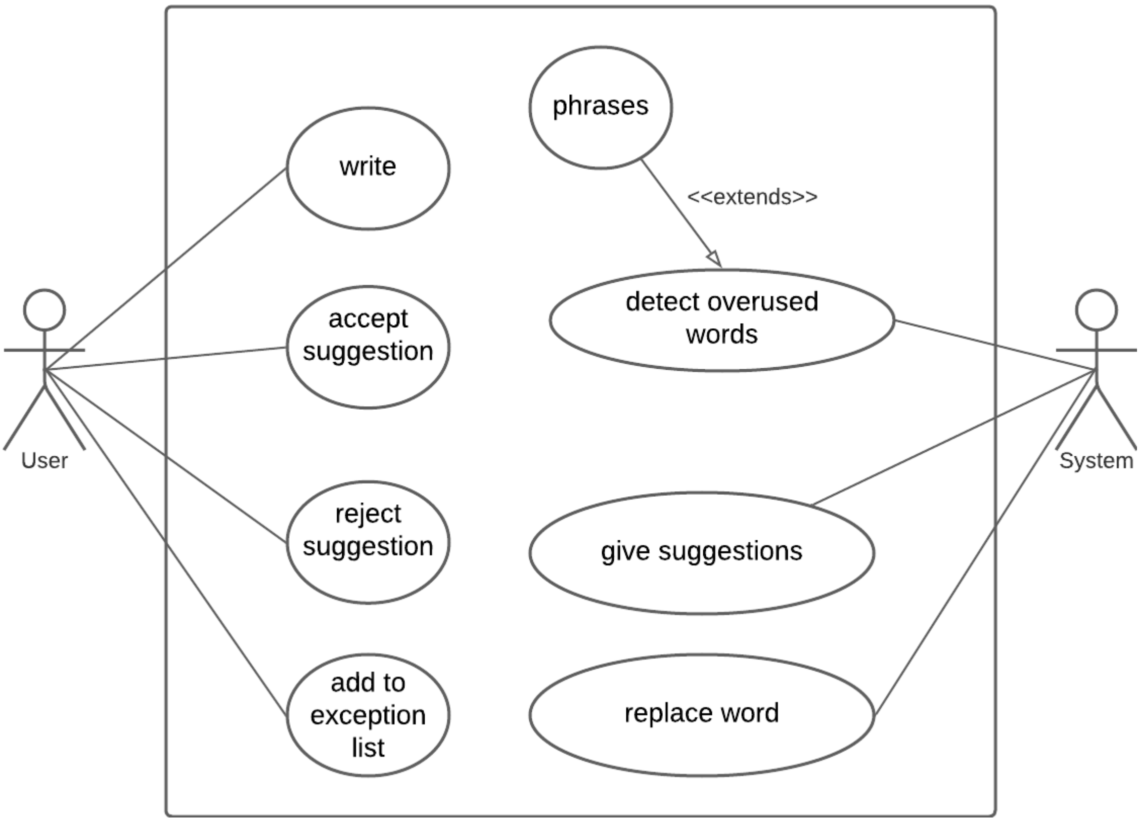


Figure 3.2 Overview Use case Diagram

Figure 3.2 displays all use cases and actors who interact with them. The diagram shows all of the main goals that the program should support in each use case.

1. Writing texts into the program, either by typing in directly or copy and pasting text.
2. Detecting overused words or phrases in the text given.
3. Giving suggestions of a replacement word to the user.
4. Accepting the suggestion, so the suggestion word can be used to replace in the text.
5. Rejecting the suggestion, so that overused words will be ignored.
6. Add overused words to the exception list which will not be suggested later on the document.
7. Replacing the suggestion word to where overused words are.

3.3.2 Sequence diagram

Scenario 1: User input texts into the program

Figure 3.3 displays of a normal scenario. when the user inputs texts through the program's UI. And after they're done typing and clicking the scan button. Those texts will be sent to the detector's part to detect overused words. Then those overused words will be sent to the suggestor's part to find alternatives for replacing words and send those word lists back to the UI to display the result to the user.

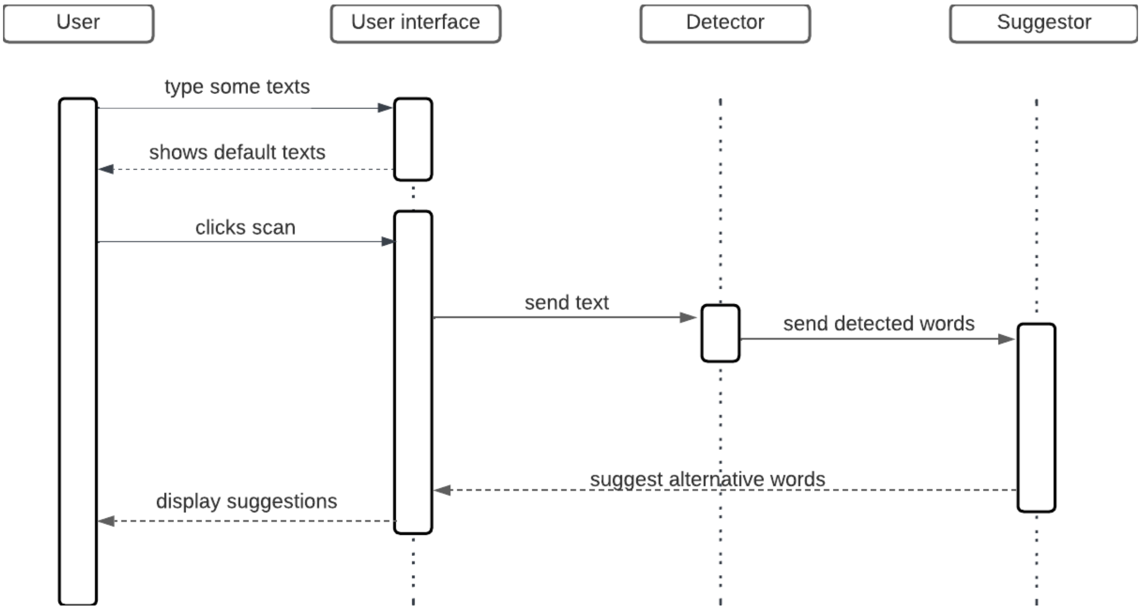


Figure 3.3 User text input sequence diagram

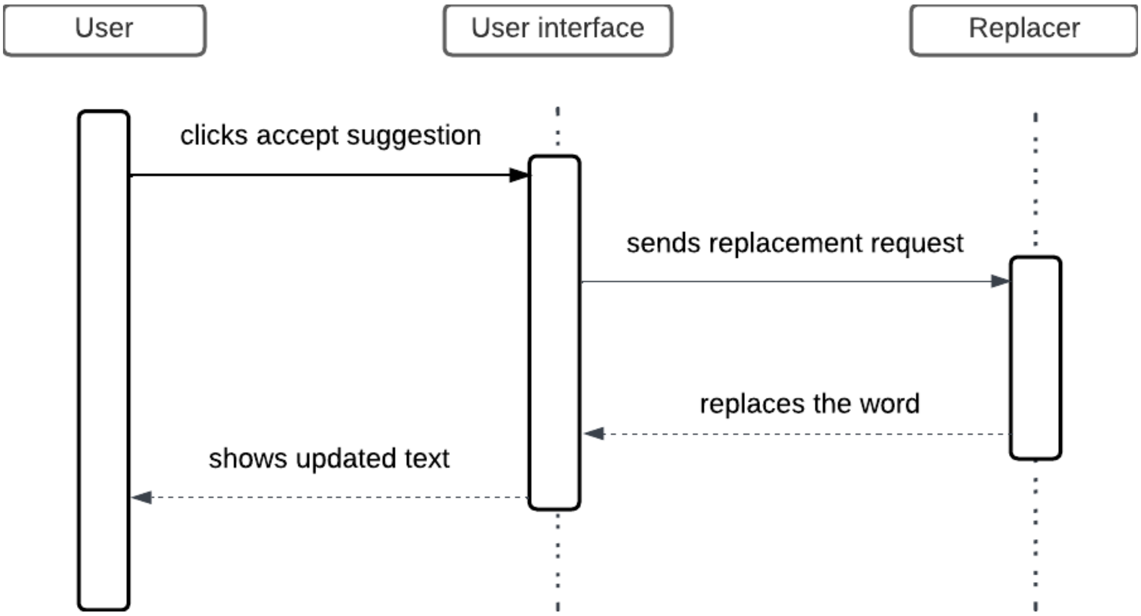


Figure 3.4 Accepting word suggestion sequence diagram

Scenario 2: User accepts the suggestion

Figure 3.4 shows the process in case the user is satisfied with the suggestion and clicks the accept suggestion button. UI will send this request to the replacer's part, and the replacer will replace the suggestion word into those overused parts throughout the UI. And shows the finalized text.

Scenario 3: User rejects the suggestion

Figure 3.5 shows the process in case the user doesn't want to change this overused word. After they clicked the reject suggestion button. The program will simply hide those suggested word lists for users not to be notified.

Scenario 4: User Add the overused word to exception list

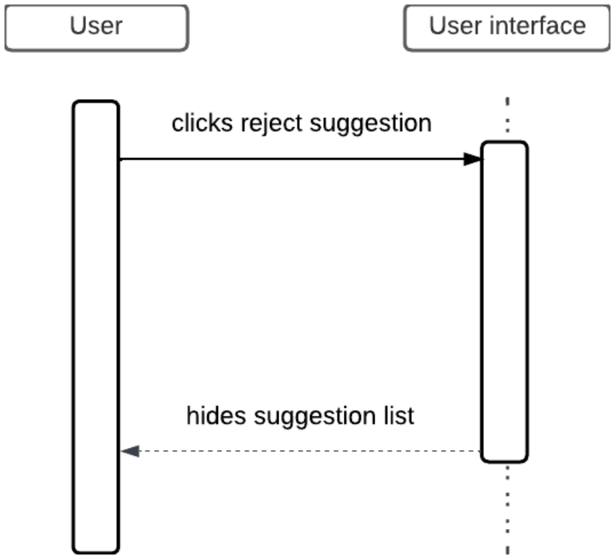


Figure 3.5 Accepting word suggestion sequence diagram

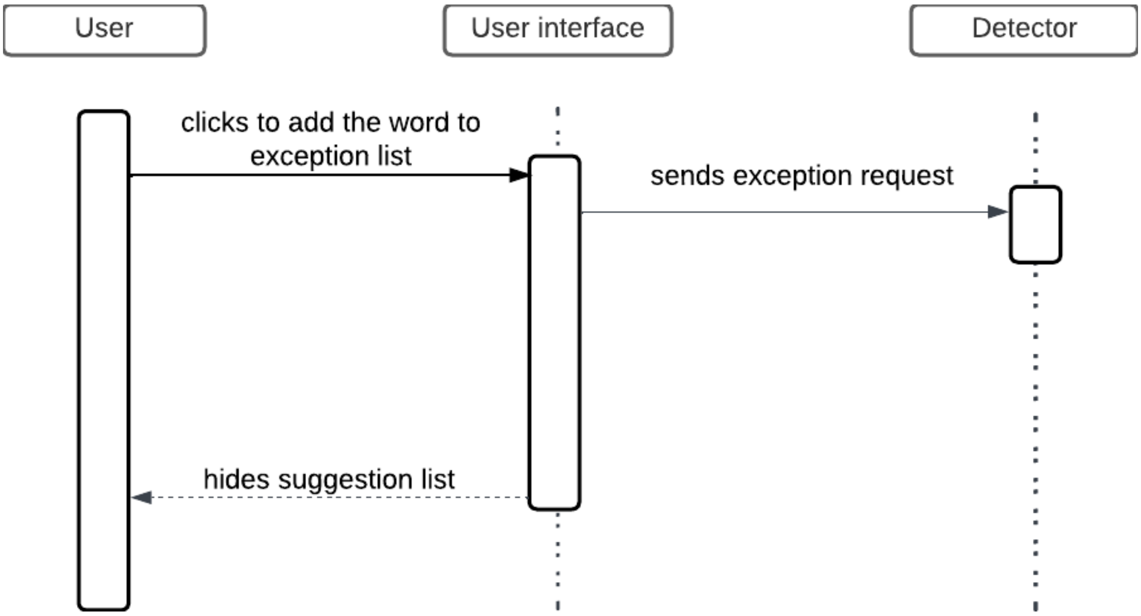


Figure 3.6 Adding word to exception list sequence diagram

Figure 3.6 shows the process when the user doesn't want this overused word to not be scanned at all in the document. So, after they click to add this word to the exception list. The program will send the exception word information to the detector to include it in. And also hide the suggestion list in the UI itself.

3.4 User Interface

3.4.0.1 Input area

Allow user to input the text either by typing out or copy-and-paste the text. The words that are marked as overused will be highlighted.

3.4.0.2 Information area

The side panel will be used to show additional details such as alternative words as well as their definition.

3.4.1 Text normalizer

The writer can either type out the text or copy and paste from somewhere else. The latter can cause problems during tokenization. One example is the apostrophe, the tokenizer expects straight quotes, if the curly variant is used, it will lead to a wrong result. The text normalizer performs token transformation to turn all the input into a compatible form supported by the tokenizer.

3.5 Tokenizer

Tokenizer is a task to break input text into individual components known as tokens. This is a critical step and is done before any further analysis can take place. The token can be in a form of a word or sub-word depending on the desired goal. In this project, we need several tokenizers to achieve the goal.

3.5.0.1 Regex Tokenizer

Using regular expression to match the specified pattern of the input text and then separate them into a token. Currently, the regular expression supports the following cases: Word matching: apple, banana, pencil word with hyphen: plant-based, state-of-the-art, cloud USD currency: $120, 1$ abbreviation: U.S.D., N.D. initials containing up to 3 characters: Mr. Mrs. Ph.D.

Unlike most out-of-the-box tokenizers which are designed to behave in a specific way, although it is easier to use, altering its behavior is nearly impossible. Some tokenizers break apart the contraction (don't → do n't) which is not a desirable result for our project. Instead, all contractions must remain as a single token. This is where the regex tokenizer comes in, it is very flexible and can be extended to cover most patterns the way we intended. However, one downside is the performance, some matcher uses greedy search behavior which can increase the execution speed, especially for a long text. After weighing the pros and cons, we decided to go with a regex-based tokenizer due to its flexibility.

3.5.0.2 Spacy tokenizer

The default tokenizer shipped with the package outputs a specific format of tokens that are different from our tokenizer, for example, don't → do and n't. In Spacy, there are many models out of the box which we can use to perform various NLP tasks namely NER and POS, but all of them expect the token to be in this particular format. Therefore, with our regexp-based tokenizer, the output tokens are not in the right format, thus, performance suffered significantly. This is why we need to use the default Spacy tokenizer instead.

3.5.0.3 BERT tokenizer

This is a special kind of tokenizer used for BERT family tokenizer. Instead of breaking down the text into words level, this breaks down the individual word to a sub-word level. For example, Internationalization → inter #national ##iz##ation. The reason is to overcome the out-of-vocabulary problem. This approach is called workpiece tokenization. Each token will be converted to embedding specific to the transformer architecture. Note that this approach is used only in the suggestor module, not anywhere else.

3.6 Cleaner

This module will be used after the text has been tokenized by the tokenizer module. At this point, the tokenized text still contains stop words, for example, I, You, So, It's, To, As, etc. These words are very common and will appear very frequently in any text regardless of the writer's ability. Thus, counting those into the repetition list will not provide much benefit for the writer in terms of helping them avoid repetitions.

3.7 Detector

The role of this module is to count all the occurrences of words in the text. There are single-word or unigrams, two-word phrases or bigrams, and three-word phrases or trigrams. In NLP, this is known as ngrams.

3.7.0.1 Unigram counter

Detect a single word occurrence in the text by counting, the common words will be filtered to keep only the meaningful words.

3.7.0.2 Bigram counter

Detect a 2-word occurrence in the text using NLTK bigram collocation package, the quality of bigram is evaluated using association score.

3.7.0.3 Trigram counter

Detect a 3-word occurrence in the text using NLTK bigram collocation package, the quality of trigram is evaluated using association score.

3.8 Suggestor

The word suggestor is used to suggest an alternative word that can be substituted in the original sentence without changing the meaning of the text.

3.8.1 Candidate generation

Roberta is used as the main model to generate substitution candidates. The model is similar to BERT but without the next sentence prediction part, leaving with just the masking language modeling (MLM) objective. The primary task is to use MLM to fill in the masked token. For instance, I like playing video game because it is fun. Let's say we want to replace the word "fun", we replace it with <mask> token. The sentence "I like playing video game because it is <mask> will be concatenated with the original unmasked sentence to create a sentence pair. The model attempts to predict the sentence pairs using the context clue from the original sentence with no mask. This approach is used alongside the second approach called embedding dropout which randomly set the target embedding to zero. Without the second approach, the predicted output is very likely to be the original word itself, which is not what we want. We can overcome this issue with the help of embedding dropout.

3.8.2 Candidate validation

Not all generated candidates are a suitable substitution. Although most of them fit in the context well, it changes the meaning of the original sentence entirely. The validation process contains scoring criteria based on semantic similarity between both sentences, and lexical relationships extracted from wordnet. A higher score will advance the rank, the lower score will be filtered out if it falls below the threshold.

CHAPTER 4 WORK PROGRESS

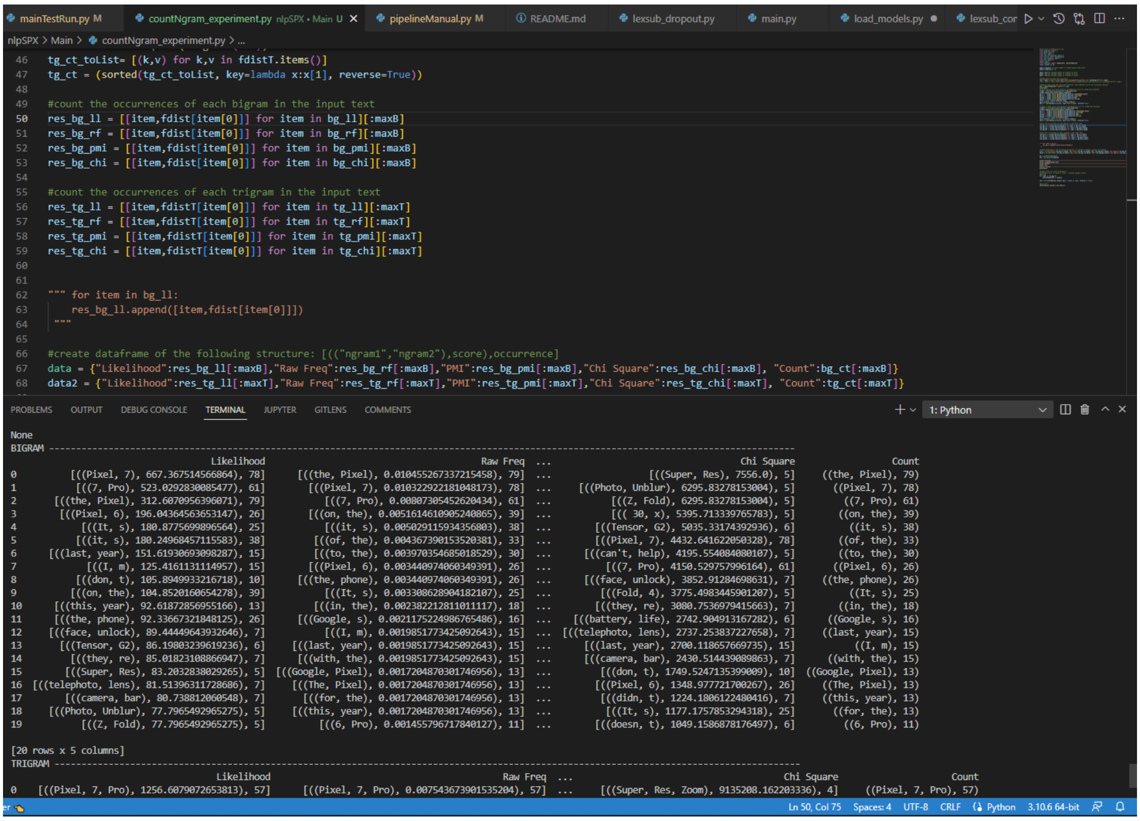
We have been working on the preprocessor including the tokenizer module, ngram detector and counter. We have also done some experiments for the word suggestion which is the second half of our project.

Tokenizer: Experiment with eight different tokenizer methods including whitespace, punctuation, word, Spacy, and regex.

Ngram detection: The current version is able to detect up to trigram. However, it is still case-sensitive which means “The camera” and “the camera” will be counted separately.

Synonym generation: We tried using BERT based model but it is quite slow, so we switched to another BERT family model such as DistilBERT and RoBERTa to see the performance improvement. The result has not been concluded yet.

Embedding dropout: Test the performance of the generated prediction using this technique. Sentence concatenation: Test the performance of the generated prediction using this technique.



```python
tg_ct_toList= [(k,v) for k,v in fdistT.items()]
tg_ct = (sorted(tg_ct_toList, key=lambda x:x[1], reverse=True))

#count the occurrences of each bigram in the input text
res_bg_ll = [[item,fdist[item[0]]] for item in bg_ll][:maxB]
res_bg_rf = [[item,fdist[item[0]]] for item in bg_rf][:maxB]
res_bg_pmi = [[item,fdist[item[0]]] for item in bg_pmi][:maxB]
res_bg_chi = [[item,fdist[item[0]]] for item in bg_chi][:maxB]

#count the occurrences of each trigram in the input text
res_tg_ll = [[item,fdistT[item[0]]] for item in tg_ll][:maxT]
res_tg_rf = [[item,fdistT[item[0]]] for item in tg_rf][:maxT]
res_tg_pmi = [[item,fdistT[item[0]]] for item in tg_pmi][:maxT]
res_tg_chi = [[item,fdistT[item[0]]] for item in tg_chi][:maxT]


""" for item in bg_ll:
    res_bg_ll.append([item,fdist[item[0]]])
 """

#create dataframe of the following structure: [(("ngram1","ngram2"),score),occurrence]
data = {"Likelihood":res_bg_ll[:maxB],"Raw Freq":res_bg_rf[:maxB],"PMI":res_bg_pmi[:maxB],"Chi Square":res_bg_chi[:maxB], "Count":bg_ct[:maxB]}
data2 = {"Likelihood":res_tg_ll[:maxT],"Raw Freq":res_tg_rf[:maxT],"PMI":res_tg_pmi[:maxT],"Chi Square":res_tg_chi[:maxT], "Count":tg_ct[:maxT]}
```

```
None
BIGRAM ----------------------------------------------------------------------------------------------------------------------------
                                 Likelihood                                   Raw Freq  ...                              Chi Square                Count
0          [((Pixel, 7), 667.367514566864), 78]     [((the, Pixel), 0.010455267337215458), 79]  ...               [((Super, Res), 7556.0), 5]     ((the, Pixel), 79)
1           [((7, Pro), 523.0292830085477), 61]      [((Pixel, 7), 0.010322922181048173), 78]  ...    [((Photo, Unblur), 6295.83278153004), 5]      ((Pixel, 7), 78)
2      [((the, Pixel), 312.6070956396071), 79]         [((7, Pro), 0.00807305452620434), 61]  ...         [((Z, Fold), 6295.83278153004), 5]        ((7, Pro), 61)
3       [((Pixel, 6), 196.04364563653147), 26]      [((on, the), 0.0051614610905240865), 39]  ...         [(( 30, x), 5395.713339765783), 5]       ((on, the), 39)
4            [((It, s), 180.8775699896564), 25]        [((it, s), 0.005029115934356803), 38]  ...     [((Tensor, G2), 5035.33174392936), 6]         ((it, s), 38)
5           [((it, s), 180.24968457115583), 38]      [((of, the), 0.004367390153520381), 33]  ...      [((Pixel, 7), 4432.641622050328), 78]       ((of, the), 33)
6     [((last, year), 151.61930693098287), 15]      [((to, the), 0.003970354685018529), 30]  ...    [((can't, help), 4195.554084080107), 5]       ((to, the), 30)
7             [((I, m), 125.4161131114957), 15]     [((Pixel, 6), 0.003440974060349391), 26]  ...         [((7, Pro), 4150.529757996164), 61]      ((Pixel, 6), 26)
8          [((don, t), 105.8949933216718), 10]   [((the, phone), 0.003440974060349391), 26]  ...   [((face, unlock), 3852.91284698631), 7]    ((the, phone), 26)
9          [((on, the), 104.8520160654278), 39]        [((It, s), 0.003308628904182107), 25]  ...        [((Fold, 4), 3775.4983445901207), 5]         ((It, s), 25)
10      [((this, year), 92.61872856955166), 13]      [((in, the), 0.002382212811011117), 18]  ...      [((they, re), 3080.7536979415663), 7]       ((in, the), 18)
11      [((the, phone), 92.33667321848125), 26]   [((Google, s), 0.0021175224986765486), 16]  ...  [((battery, life), 2742.904913167282), 6]    ((Google, s), 16)
12    [((face, unlock), 89.44449643932646), 7]        [((I, m), 0.0019851773425092643), 15]  ... [((telephoto, lens), 2737.253837227658), 7]   ((last, year), 15)
13       [((Tensor, G2), 86.19803239619236), 6]    [((last, year), 0.0019851773425092643), 15]  ...    [((last, year), 2700.118657669735), 15]         ((I, m), 15)
14          [((they, re), 85.01823108866947), 7]   [((with, the), 0.0019851773425092643), 15]  ...    [((camera, bar), 2430.514439089863), 7]      ((with, the), 15)
15         [((Super, Res), 83.2032838029265), 5] [((Google, Pixel), 0.0017204870301746956), 13]  ...        [((don, t), 1749.5247135399009), 10] ((Google, Pixel), 13)
16 [((telephoto, lens), 81.51396311728686), 7]   [((The, Pixel), 0.0017204870301746956), 13]  ...      [((Pixel, 6), 1348.977721700267), 26]    ((The, Pixel), 13)
17         [((camera, bar), 80.738812060548), 7]     [((for, the), 0.0017204870301746956), 13]  ...       [((didn, t), 1224.1806122480416), 7]    ((this, year), 13)
18    [((Photo, Unblur), 77.7965492965275), 5]   [((this, year), 0.0017204870301746956), 13]  ...         [((It, s), 1177.1757853294318), 25]      ((for, the), 13)
19           [((Z, Fold), 77.7965492965275), 5]        [((6, Pro), 0.001455796717840127), 11]  ...      [((doesn, t), 1049.1586878176497), 6]        ((6, Pro), 11)

[20 rows x 5 columns]
TRIGRAM ---------------------------------------------------------------------------------------------------------------------------
                                 Likelihood                                   Raw Freq  ...                              Chi Square                Count
0   [((Pixel, 7, Pro), 1256.6079072653813), 57]   [((Pixel, 7, Pro), 0.007543673901535204), 57]  ...   [((Super, Res, Zoom), 9135208.162203336), 4]    ((Pixel, 7, Pro), 57)
```

REFERENCES

1. Fosu K., 2020, "How Unintentional Repetition Hurts Your Writing," Available at https://medium.com/the-brave-writer/how-repetition-hurts-your-writing-cb68c292525f, [Online; accessed October 2022].

2. IBM, 2020, "Natural Language Processing (NLP)," Available at https://www.ibm.com/cloud/learn/natural-language-processing, [Online; accessed October 2022].

3. D. Jurafsky and J.H Martin, 2021, "Vector Semantics and Embeddings," Available at https://web.stanford.edu/~jurafsky/slp3/6.pdf, [Online; accessed October 2022].

4. Utkarsh Ankit, 2022, "Transformer Neural Networks: A Step-by-Step Breakdown," Available at https://builtin.com/artificial-intelligence/transformer-neural-network, [Online; accessed November 2022].

5. Zhou W, Ge T, Xu K, Faru W, and Ming Zhou, "BERT-based Lexical Substitution," **Aclanthology**.

6. Yi Zhu Jipeng Qiang, Yun Li, "A Simple BERT-Based Approach for Lexical Simplification," **arxiv**.

7. R Horev, 2018, "BERT Explained: State of the art language model for NLP," Available at https://towardsdatascience.com/bert-explained-state-of-the-art-language-model-for-nlp-f8b21a9b6270, [Online; accessed October 2022].