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

..... (Asst. Prof. Nuttanart Muansuwan, Ph.D.)	Project Advisor
..... (Asst.Prof. Santhitham Prom-On, Ph.D.)	Committee Member
..... (Asst.Prof. Rajchawit Sarochawikasit, Ph.D.)	Committee Member

Project Title	Writing Aid to Avoid Repetitions
Credits	3
Member(s)	Mr.Annop Pornsawatdipak
	Mr.Thanapatr Amnuaywit
Project Advisor	Asst. Prof. Nuttanart Muansuwan, 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 a computer. Most of the tools focus on fixing grammatical accuracy and misspelled words, only a 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 fit in the 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. It can detect a single word (unigram) up to three words (trigram) repetitions. Next, the suggestor module will generate multiple substitution candidates by feeding the original sentence along with the masked sentence into the Transformer model. The models such as BERT and Roberta are 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 will be evaluated by four different judges (judges = models), and the final result will be determined using a consensus-based ranking system to find broadly acceptable positions among all judges.

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

การเขียนโดยใช้คอมพิวเตอร์ช่วยมีจุดมุ่งหมายเพื่อช่วยปรับปรุงคุณภาพของข้อความด้วยความช่วยเหลือของคอมพิวเตอร์ เครื่องมือส่วนใหญ่มุ่งเน้นไปที่การแก้ไขความถูกต้องทางไวยากรณ์และคำที่สะกดผิด แต่ก็ยังมีเครื่องมือน้อยที่ใช้ในการตรวจจับคำซ้ำ ในโครงการนี้เราจะสร้างโปรแกรมเพื่อติดตามความถี่ของคำและวลี นอกจากนี้ โปรแกรมของเราจะแนะนำคำทางเลือกที่เหมาะสมกับบริบทเดิม ดังนั้น ผู้เขียนสามารถแทนที่คำซ้ำด้วยคำใหม่ที่แนะนำแทน วิธีแก้ปัญหาที่เรานำเสนอประกอบด้วยสองส่วนหลัก ได้แก่ ส่วนตรวจจับ และส่วนแนะนำ ในส่วนแรก โปรแกรมจะพยายามตรวจหาเฉพาะคำซ้ำที่เกี่ยวข้องและกรองคำเฉพาะหรือคำที่ไม่สำคัญออก โปรแกรมของเราสามารถตรวจจับคำซ้ำได้ตั้งแต่คำเดียว (ยูนิแกรม) จนถึงสามคำ (ไตรแกรม) จากนั้นโมดูลผู้แนะนำจะสร้างรายการคำตัวเลือกโดยป้อนประโยคต้นฉบับพร้อมกับประโยคที่ทำการมาสก์โทเค่นลงในแบบจำลองทรานส์ฟอร์มเมอร์สเช่น เบิร์ทและ โรเบอร์ต้า ได้รับการฝึกฝนด้วยมาสก์แลงเกวจโมเดลลิ่ง ซึ่งเราสามารถใช้เพื่อทำนายตำแหน่งของมาสก์ ในกรณีนี้ คำเป้าหมายที่จะแทนที่ รายการคำที่สร้างขึ้นจะได้รับการประเมินโดยผู้ตัดสินที่แตกต่างกัน 4 คน (ผู้ตัดสินหมายถึงแบบจำลอง) และผลลัพธ์สุดท้ายจะถูกตัดสินโดยใช้ระบบการจัดอันดับตามฉันทามติเพื่อค้นหาการจัดอันดับที่เป็นที่ยอมรับจากผู้ตัดสินทั้งหมด

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. Conventionally, the POS tag uses hard-coded, rule-based approaches. However, with the advancement of technology, nowadays, machine learning techniques have become a standard for assigning part of speech to words in a sentence. This modern approach uses a statistical model trained on large multi-language datasets, it is capable of generalizing across all languages.

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. A well-known example is OpenAI’s Generative Pretrained Transformer (GPT) which has taken the world by storm recently due to its human-like response.

2.2.1.6 Keyword Extraction

A method used to automatically find the most relevant words and phrases from a piece of document[3]. 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)[4]. 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 → th is to ke ni zi ng → this is token in zi ng. The most frequent character pair are merged into one token, which will be added to the vocabulary. The process repeats until it the target vocabulary limit is reached. The BPE-based tokenizer is used in multiple models including BERT and Roberta to combat the out-of-vocabulary problem. Considering the size and amount of datasets required to train the model, it may encounter domain-specific vocabularies that are not in the dictionary. A Wordpiece tokenizer is used in BERT family model. According to Eram M, [A comprehensive guide to subword tokenisers | by Eram Munawwar | Towards Data Science] Instead of relying on the frequency of the pairs, WordPiece chooses the one which maximizes the likelihood of the training data. This means that it trains a language model starting on the base vocabulary and picks the pair with the highest likelihood. For example, Internationalization → inter national ##iz##ation. This is again, to overcome the out-of-vocabulary problem.

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 inputp[5]. 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[6][7]. 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.

2.2.6.10 Masked langauge modelling

A training objective used in Transformer models like BERT and Roberta. This technique masks some portions of the input sentence, the model will try to fill in the blank by looking at the remaining unmasked token. For example: In reality, rich people can bribe the officer to avoid a long prison sentence. In reality, rich <mask> can bribe the officer to avoid a long prison sentence. The model will try to predict what word would fit in the mask position taking into account the surrounding contexts. The model has been trained on huge datasets both online and offline sources such as Wikipedia text and book corpus. Despite enormous data, the model may give biased results toward a certain gender or race. This technique combined with attention mechanism permits the model to learn and capture contextual information. Some of the models that utilize this technique are BERT, Roberta, XLM, and distilBert. There are many use cases namely, text summarization, question answering, translation, token classification, entailment problem, and more.

2.2.6.11 Finetuning

A machine learning technique used to adapt a pre-trained model to perform specialized tasks. For instance, ResNet, an image classification model has already gained some general knowledge about the visual world. When building a new model for a domain-specific problem, we do not need to train it from scratch, instead, we build on top of the pre-trained model. In this project, we experimented with this technique but the result was unsatisfactory.

2.2.6.12 Consensus Ranking

A consensus-based ranking is a type of voting system used in an election. Instead of relying on a majority vote, consensus voting uses a broadly accepted preferential order to determine the winner. There are numerous existing policies, but in this project, the Borda count method will be used. Borda count is a family of positional voting first purposed by Nicholas of Cusa in 1435, the nomenclature "Borda" is named after the French mathematician and engineer Jean-Charles de Borda in 1770. Generally, voters will rank their most wanted candidate first, second, third, and so on. Each position is assigned a point, the first position gets n-1 point, and the last position gets 0 points (*where n is the number of candidates). The one who possessed the most point will be the winner.

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 PyTorch

A machine learning library written in python.

2.3.3 Pycharm Community

Another code editing software from JetBrains, it's an open source version, specialized on edit and testing code in Python language.

2.3.4 Natural Language Toolkit (NLTK)

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

2.3.5 Spacy

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

2.3.6 Tkinter

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

2.3.7 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.

2.3.8 Matplotlib

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

2.3.9 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.

2.3.10 Sentence Transformer

A library leveraging on the original BERT architecture and Huggingface's Transformers library, provides the ability to work with sentence-level as opposed to token-level embedding in the vanilla BERT. The pre-trained models have been trained on the datasets including Standford Natural Language Inference (SNLI) containing 570K sentence pairs and Multi-Genre Natural Language Inference (MNLI) containing 430K pairs. The dataset is labeled with 3 tags namely contradiction, neutral, and entailment. The model will learn to classify each sentence into one of the three categories. The main goal of Sentence Transformer framework is to compute the semantic textual similarity between 2 sentences, semantic search, and paraphrase mining. The closeness of embeddings is assessed with a cosine similarity score. In our project, it will be used to evaluate the quality substitution candidates and determine their appropriate ranking.

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.[8][9], 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 Simplification purposed by Jipeng et al[10][9]. 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. The sentence Concatenation technique guarded inappropriate predictions produced by the model. This behavior is expected because the model has been trained to guess the <mask> with the most probable word based on the context. However, in reality, the highest probability word although grammatically correct and perfectly make sense is not guaranteed to retain the sentence's original meaning.

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)[11]. 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[12]. 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. Roberta stands for A Robustly Optimized BERT Pretraining Approach proposed by Yinhan Liu et.al. This model includes some tweaks mainly in the masking technique, as a result, it performs better than the original BERT. The researcher used dynamic masking to change the masking pattern of the sentence in each epoch. Unlike in BERT where the masking is done only once at the beginning, thus it remains static throughout the procedure. The modified BERT also used larger batch sizes and datasets during the training. Allowing the model to advance even further.

2.4.4 Existing product in the market

2.4.4.1 Grammarly

A writing assistant powered by Artificial Intelligence based in San Francisco, California. Grammarly is a powerful tool that can suggest and correct written text in multiple dimensions, for instance, grammatical accuracy, plagiarism, tone detection, style guide, structural, and brand tone. It supports a variety of platforms including Microsoft Word, Google Docs, Mac, Windows, Chrome, Safari, Firefox, iPhone, iPad, and Grammarly Keyboard. According to its official website, it generates over 350,000 suggestions every minute for its 30 million daily users (Grammarly, n.d.). Despite all the capabilities, besides grammar and spelling checker, most of the advanced features are not available on the free version, thus, the users must pay a monthly subscription to access them. Moreover, Grammarly doesn't have any feature that focuses on the problem of repetition, so there is a gap to improve upon[10].

2.4.4.2 Prowritingaid.com

Another writing assistant tool claims its superiority, saying "good writing is bout more than just grammar"(ProWritingAid, n.d.). The tool highlights a range of writing issues including overused words, sentence structure, punctuation issues, repeated phrases, consistency, pacing, and readability. In terms of pricing, similar to Grammarly, ProWritingAid free offers basic functionality such as a spelling checker and basic grammar correction, other features are limited to paid users only. One interesting feature is the repeated words report, it summarizes all the most frequent words and gives some suggested alternatives to choose from. This feature

detects repeated words and phrases spanning across the specified distance, for example, the distance of 300, any repeats beyond 300 characters apart will not be flagged[11].

2.4.4.3 Writer.com

Another writing assistant that can help polish a piece of writing. But it emphasizes on team editing and content creation for the brand. There are numerous features that are far beyond the scope of the 2 previously mentioned services, such as a customized machine learning model using a custom writing style, and collaborative writing in real time that ensure consistency across all the team members. The basic corrections are available in its free plan, but any advanced features require a subscription[12].

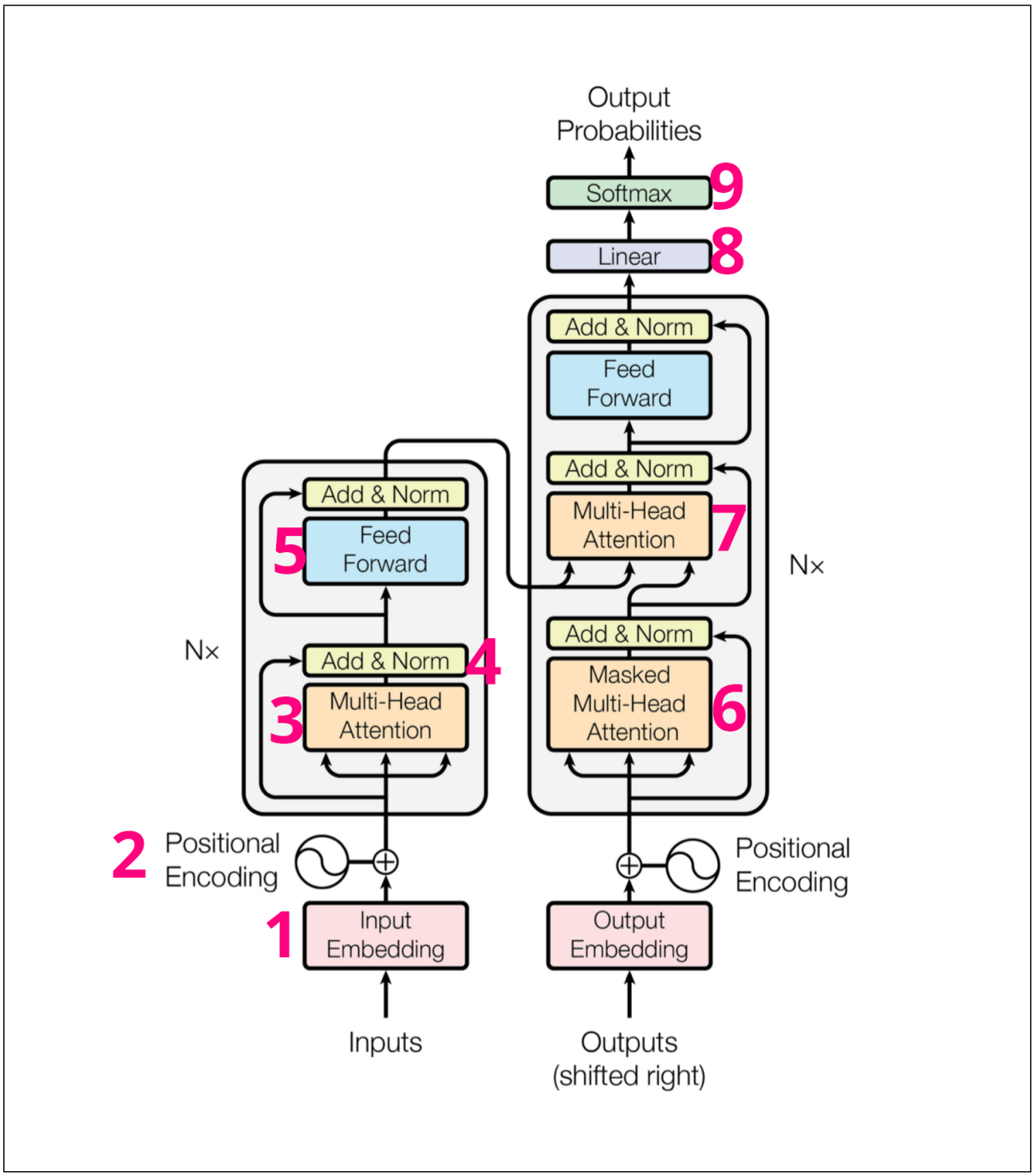

Figure 2.1 The network model

CHAPTER 3 CHAPTER 3 DESIGN AND METHODOLOGY

3.1 System Architecture

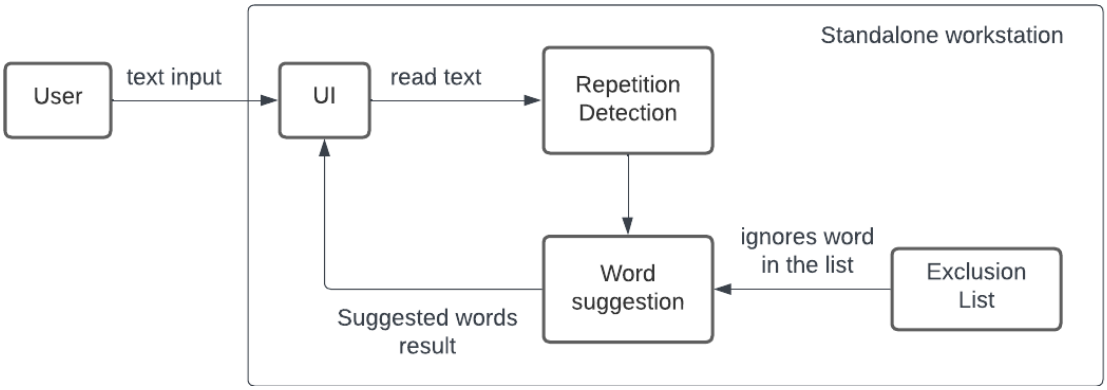

Figure 3.1 System Architecture of an application

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

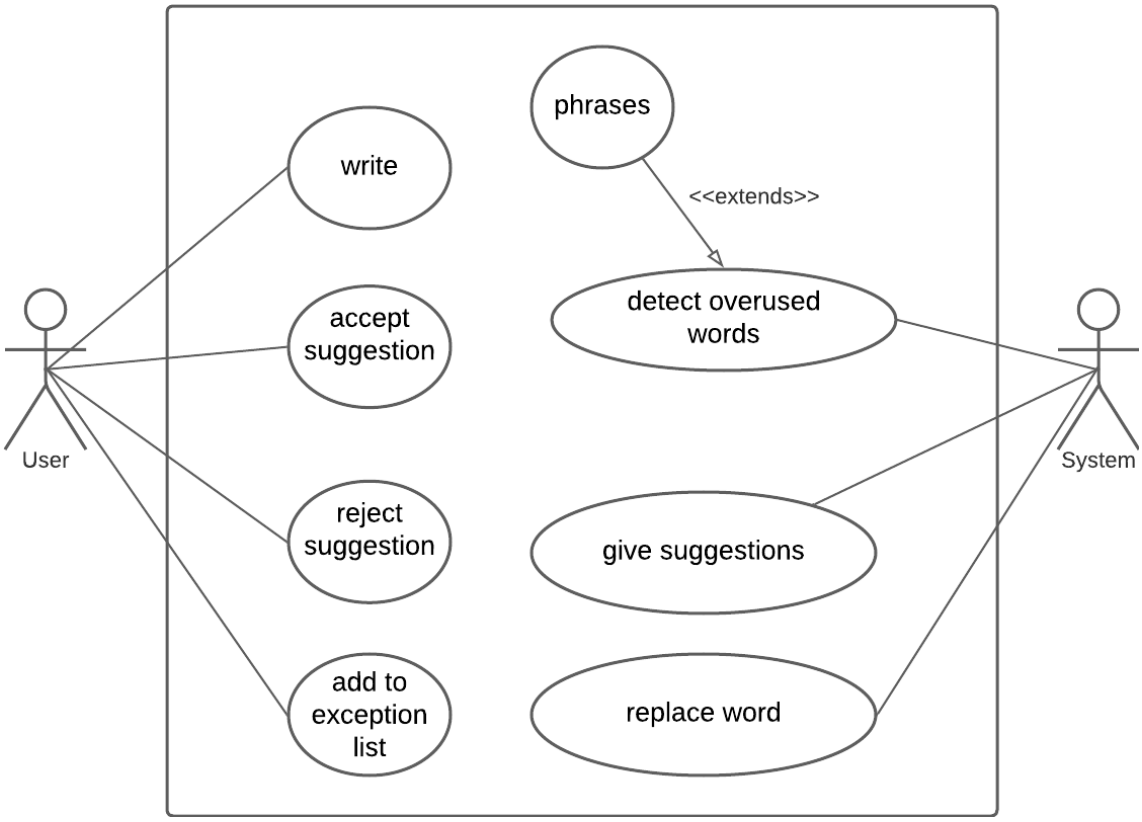

Figure 3.2 Overview Use case Diagram

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.

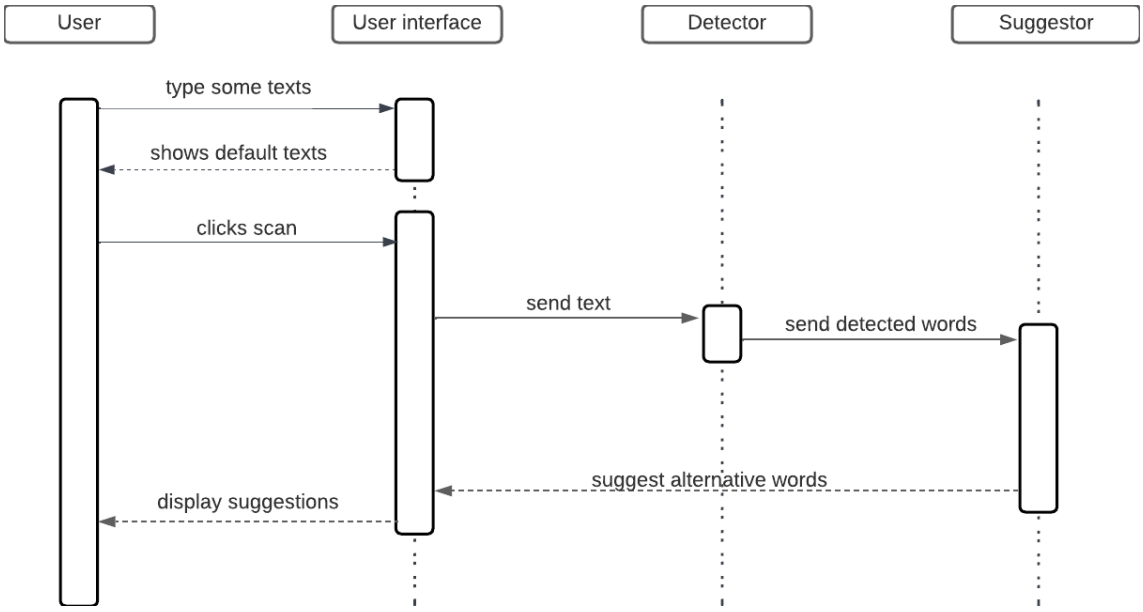

Figure 3.3 User text input sequence diagram

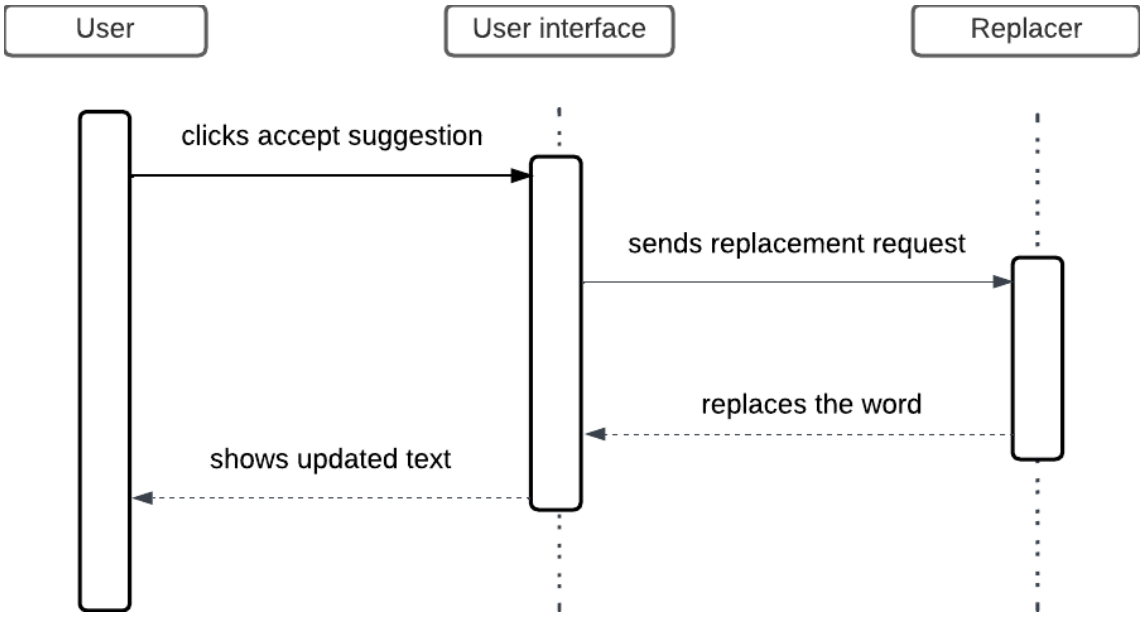

Figure 3.4 Accepting word suggestion sequence diagram

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

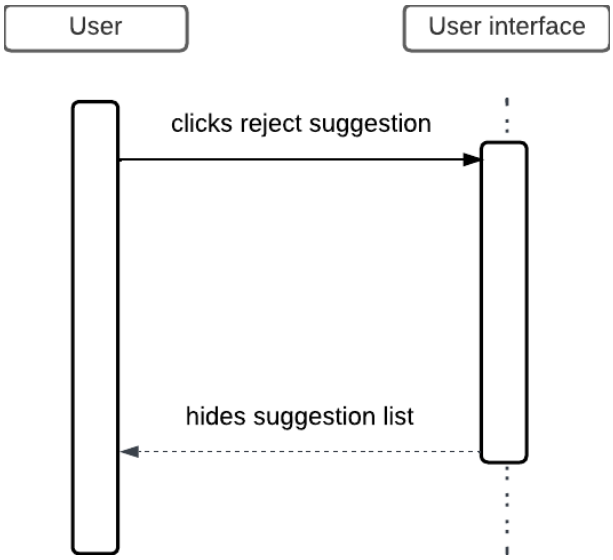

Figure 3.5 Accepting word suggestion sequence diagram

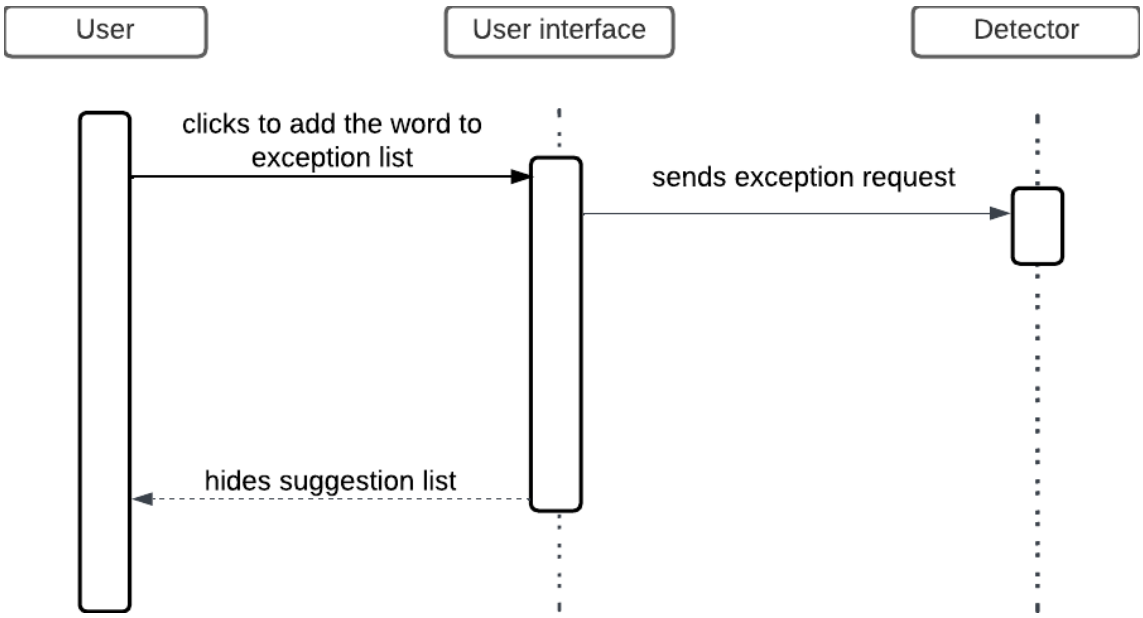

Figure 3.6 Adding word to exception list sequence diagram

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

3.4.1 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.

3.4.2 Information area

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

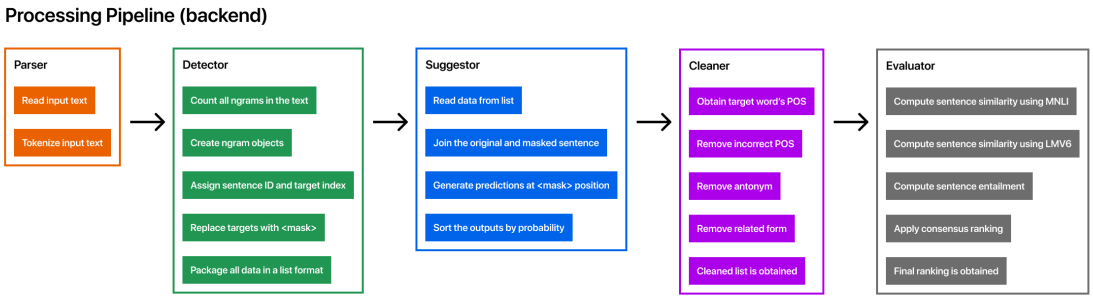

Figure 3.7 Processing pipeline of the core engine

3.5 Parser

3.5.1 Read input text

This program supports both user-typed text and pdf files. In the case of pdf file, the program will extract the text content and tokenize it.

3.5.2 Tokenizer input text

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. For the Parser, we used whitespace tokenizer in conjunction with a Spacy tokenizer.

3.5.2.1 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 such as the one included in Spacy will 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. While working on the project, we noticed that our long and complex regex tokenizer produced a result comparable to a more simple whitespace tokenizer. Our regex matcher is superior to whitespace only in the case of abbreviations for example Mr. and U.S.A., regex keeps all the dots intact. However, retaining the dot doesn't add much value to the prediction accuracy of the model. Moreover, our custom regex expression is more prone to errors when dealing with complex text which may contain untested edge cases. Ultimately, we have made the decision to use a modified whitespace tokenizer instead.

3.5.2.2 Modified Whitespace Tokenizer

All the bigram counters will work on the tokens produced by the new modified whitespace tokenizer. Usually, this type of tokenizer splits words by whitespace, however, this means the last word of each sentence will include a full stop. Any word with an extra dot at the end will be counted as a separate item by the n-gram frequency counter.

I like apple. → apple I like apple. → apple.

Thus, the program trims out the excess dots and symbols at the end of each token. This will not affect the accuracy of the prediction.

3.5.2.3 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. All the hyphens are split as a separate tokens. In Spacy, basic tasks such as tokenization, sentence boundary detection, POS, and NER all require a specific model. Our custom tokenization cannot be used in this context. Instead, the default tokenizer shipped with the spacy language package will be used. In Parser, the Spacy tokenizer is used so that we can perform sentence boundary detection, NER, and POS later on in the process

3.6 Detector

The role of this module is to count all the occurrences of n-gram up to trigram in the text. In the end, it generates a package that will be sent to the sugestor model.

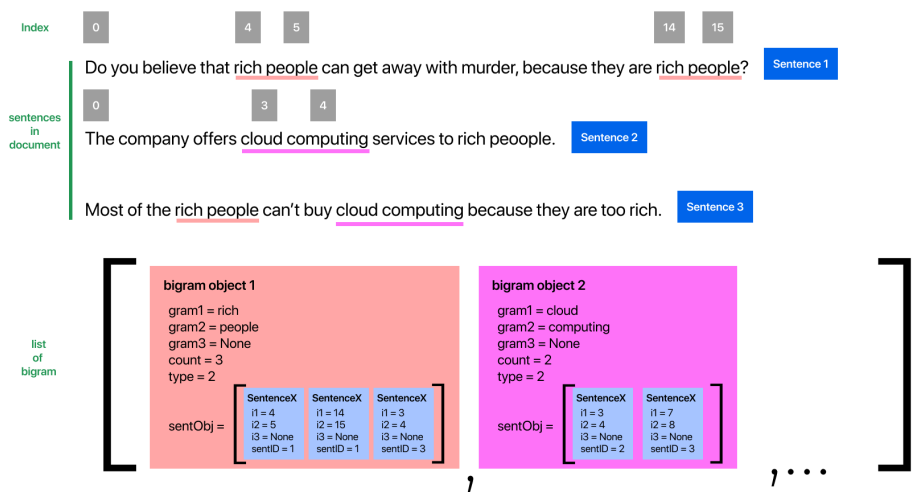

Figure 3.8 Structure of n-gram object

3.6.1 Count all n-gram in the text

Detect unigram, bigram, trigram occurrences in the text by counting, the words in the exclusion list will be filtered out to keep only the meaningful words.

3.6.2 Create n-gram object

The detector will generate a list of n-gram objects for each type of n-gram namely unigram, bigram, and trigram.

3.6.3 Assign sentence ID and target index

The findWord() module will go through the list of n-gram objects. For each of them, it finds the sentence containing the target n-gram and indexes of the target in the sentence. All the positional information of each n-gram target is stored as an index instead of a full sentence as in our previous implementation to reduce memory usage. The information is in a cascaded list of objects inside the n-gram list. By doing this, we can directly access the position of the target word during the <mask> replacement process.

3.6.4 Repalce target with <mask>

After that, it replaces all the target index with <mask> token, if the word is not on the exclusion list. If the word is a part of the list, the replacement will not be performed. In case of multi-words expression like bigram or trigram, each gram is assessed individually, if it is not in an exclusion list, <mask> token is applied, otherwise the word will be skipped. In case of multiple occurrences of the same bigram present in the same sentence, a separate sentence object will be created for all the duplicated instances.

3.6.5 Package all data in a list format

Finally, the FM (Fill Mask) package containing the original sentence, masked sentence, and the target word is created. This format is sent to the predictor module.

3.7 Exclusion List

In a document, not all words are substitutable, some of them have special grammatical features which if changed, may create ungrammatical sentences. This is particularly true for preposition, modal verb, conjunction, contraction, negation, and pronoun. In NLP, we refer them as a stopword. Thus, the detector will ignore all of the stopwords and will not generate alternatives. Additionally, all the special entities such as name, place, country, work of art, and organization will also be excluded. The following words will be discarded by default: Stopwords such as I, You, We, He, They, Can't, Could, Will, To, Not, Am, Are Is, Were, Because, etc. Named entity: Location, Name, Country, Work of Art, Person, Organization, etc
If the users do not want to get suggestions for a particular word, they can manually add them to the list as well.

3.8 Suggestor

Roberta, a Robustly Optimized BERT pretraining Approach 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 only 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.
The candidate generation process begins with the FM package from the Detector module. It contains information of the following structure The first element, unmasked sentence will be concatenated with the second element, masked sentence, and the third element, target word, will be used to facilitate the evaluation process.

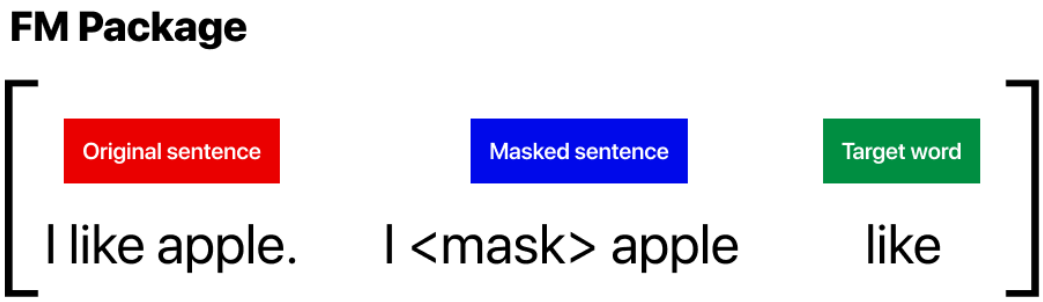

Figure 3.9 FM Package (Fill Mask Pacakage)

In the case of bigram and trigram, each FM (Fill-Mask) package will be created separately for each word in the n-gram. Refer to the following example:

- SENTENCE: Most rich people like trees. → target bigram = "rich", "people"
- PACKAGE 1: [Most rich people like trees., Most <mask> people like tree, rich]
- PACKAGE 2: [Most rich people like trees., Most rich <mask> like tree, people]

The suggestor module will perform the prediction according to the list being sent by the Detector module. In this case, 2 predictions are applied to the same sentence.

3.9 Cleaner

The output candidates still contain words with incorrect part of speech, verb tense, and non-words. As a result, further cleaning is performed to remove all the wrong items. The following operations will be applied to the candidate list;

3.9.0.1 POS filtering

Firstly, part of speech of the original word is obtained, next, the part of speech of the all the candidates is obtained. Any word with a mismatched POS with respect to the original will be removed, leaving only the candidates with the correct POS. Nevertheless, this method is not perfect because sometimes, candidates with wrong verb tense may be tagged incorrectly. Consider this example:

– I drive Tesla Model X to London. drive is a verb

– I driver Tesla Model X to London driver is a noun, but it will be tagged as a verb

This is due to the tagger being trained to recognize grammatically valid sentences, this sentence is ungrammatical, therefore, unexpected behaviors may occur.

3.9.0.2 Antonym removal

Secondly, the antonym removal, during this stage, an antonym list is extracted from Wordnet synset database. Words within the antonym list will be removed from the candidate list.

3.9.0.3 Derivationally related form removal

A derivationally related form is a derived form of the root word. This function removes all variant forms of the word occur in the output list.

3.9.0.4 Inflection removal

Lastly, inflection removal removes any inflected form of the target word. All of the aforementioned procedures grealtly improve the correctness of the final result. Despite all that, some incorrect word may still be present in the result.

3.10 Evaluator

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.

During the implementation, we found out that Wordnet lexical relationship only includes verbs and nouns, but not adjectives. So we cannot rely solely on Wordnet. Our revised strategy employs consensus-based ranking. The technique involves 4 judges, in this project, 4 models are used. The first model is the original ranking based on top-k output from the fill-mask pipeline. The second model is called Roberta-base-v2 from Sentence Transformer library. In this project, we use this model to compute sentence similarity against the original sentence. The third model is named Roberta-base-MNLI. It is suitable for computing the similarity between sentences. Lastly, the cross-encoder NLI Roberta-base. This model is used to predict whether 2 sentences entail or contradict each other. NLI is useful for filtering out words that carry the opposite meaning to the original word but are not present in the Wordnet antonym list. This can happen because some words may not explicitly be an antonym according to the dictionary, but in certain contexts, they could be.
All the rankings from each model will be aggregated into the ultimate ranking by considering common positions among all 4 judges. The candidate that is ranked consistently across all 4 rankings will hold its position. In contrast, if there are some disagreements, the final position of a particular word will be readjusted accordingly. To elaborate, the word A ranked 3,2,3,4 which is fairly consistent. On the other hand, word B ranked 5,12,9,16, which indicates inconsistency and will need to be reordered using the Borda count algorithm. This method improves average ranking accuracy compared to our previous approach where we relied on a single judge.

3.10.1 Consensus Ranking

All the rankings from each model will be aggregated into the ultimate ranking by considering common positions among all 4 judges. The candidate that is ranked consistently across all 4 rankings will hold its position. In contrast, if there are some disagreements, the final position of a particular word will be readjusted accordingly. To elaborate, the word A ranked 3,2,3,4 which is fairly consistent. On the other hand, word B ranked 5,12,9,16, which indicates inconsistency and will need to be reordered using the Borda count algorithm. This method improves average ranking accuracy compared to our previous approach where we relied on a single judge.

CHAPTER 4 IMPLEMENTATION

4.1 Current progress

This section contains the current progress of the actual program and its resources to show. Separate into two parts which are the application's UI and the model. Thus, some features we declared in (Feature lists) can not be shown here.

4.1.1 User Interface

1. Initialize

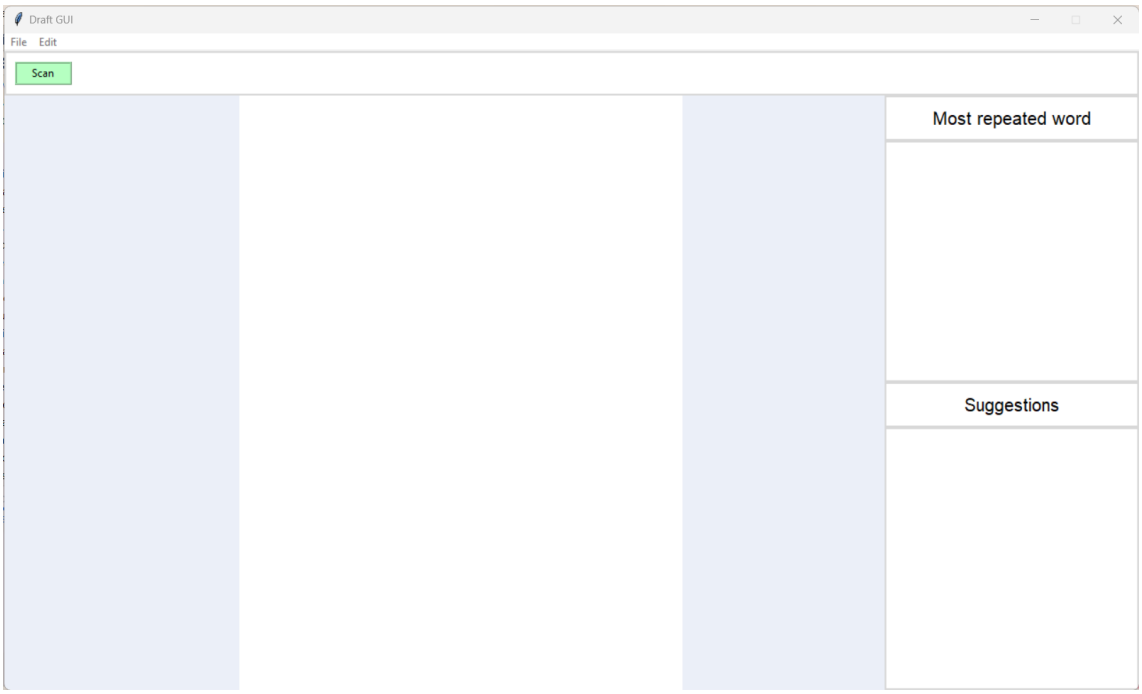

Figure 4.1 Application's starting screen

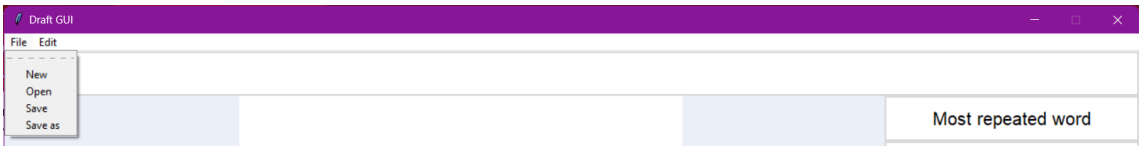

Figure 4.2 'File' button dropdown in the menu bar

Figure 4.1 shows the program's blank screen after it's launched. Showing some basic labels and spaces including.

- Menubar: At the top of the window, there's a bar that contains basic functions such as opening documents or saving.
- Text workspace: A white blank textbox in the middle of the window, consume most of the window's area for editing text.

– Scan button: This button is located below the menubar. Which will scan the text in the text workspace when there's any text in the space.

– Sidebar: Showing key functions to look for most repeated words and its suggestion to replace them or else.

2. Open file

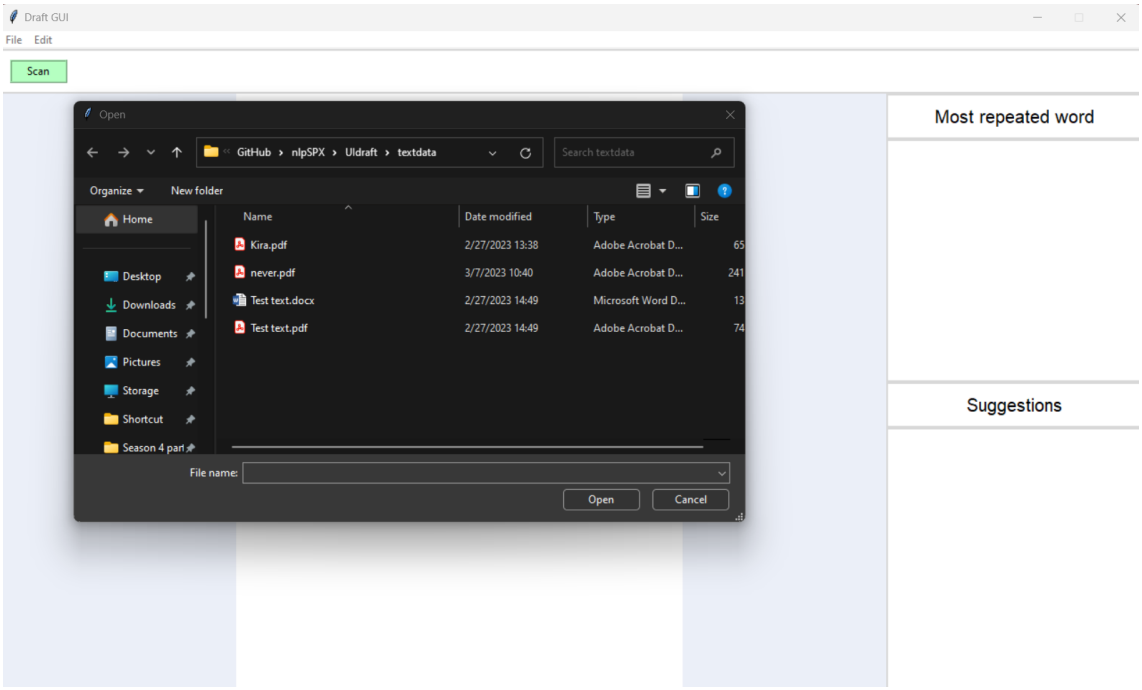

Figure 4.3 Choose file window

After the user choose to import the document into the text workspace by going to 'File' –> 'Open' from the menubar shown in figure 4.3. A File Explorer window will appeared to choose .pdf folder from. And the application will read the text in the file, then show it in the text workspace.

3. Scan function

After the user has imported the text or there's some amount of text in the workspace. If they click the 'scan' button below the menu bar. The application will scan the text and show most repeated word in the upper side of the sidebar as figure 4.4 shows.

4. Text highlighting

The method of text highlighting has been changed from what it should be in section 3.2.2. As figure 4.5 and 4.6 shows, after the user clicks on any most repeated word shown in the sidebar. That word in the text will be highlighted by using Regex method, captures the exact word as the selected repeated word. Also, clicking a different word will remove the highlight from the old word and moves to another word in the text.

4.1.2 Application's backend

Parser The text reader module inside a Parser is used to read pdf files from disk. This module read the input, ignoring images and graphics, and removes all the extra spaces in the text. The textual content of the file will be shown on the text GUI as seen in section 4.1.1.

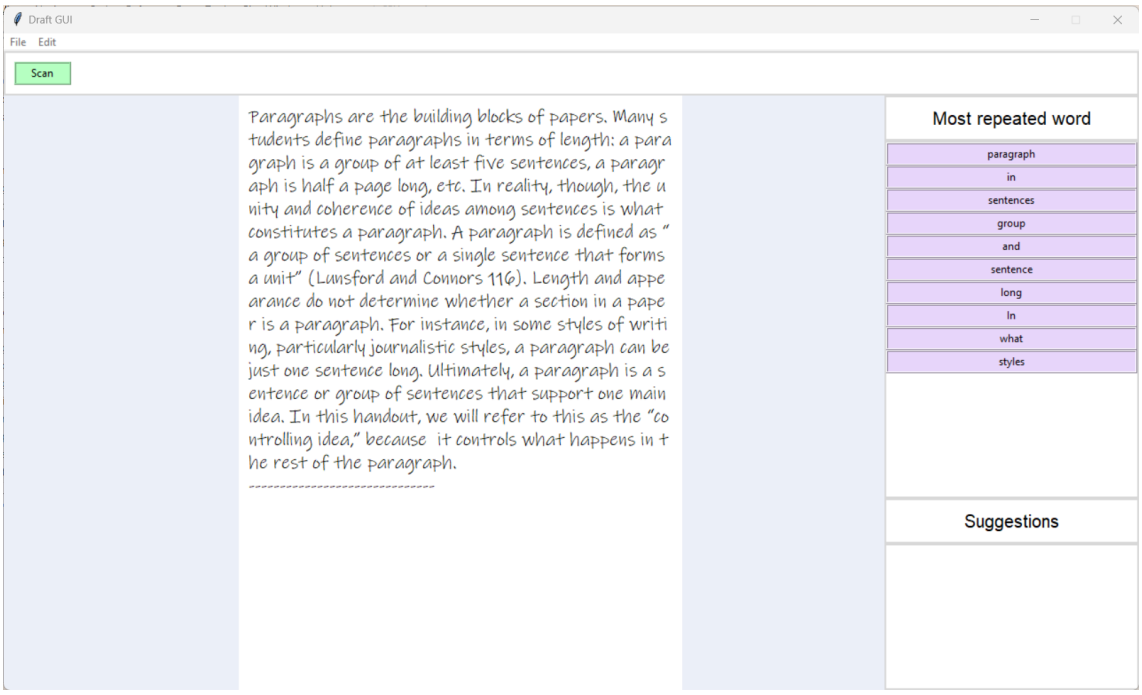

Figure 4.4 Most repeated words shown after clicking 'scan' button

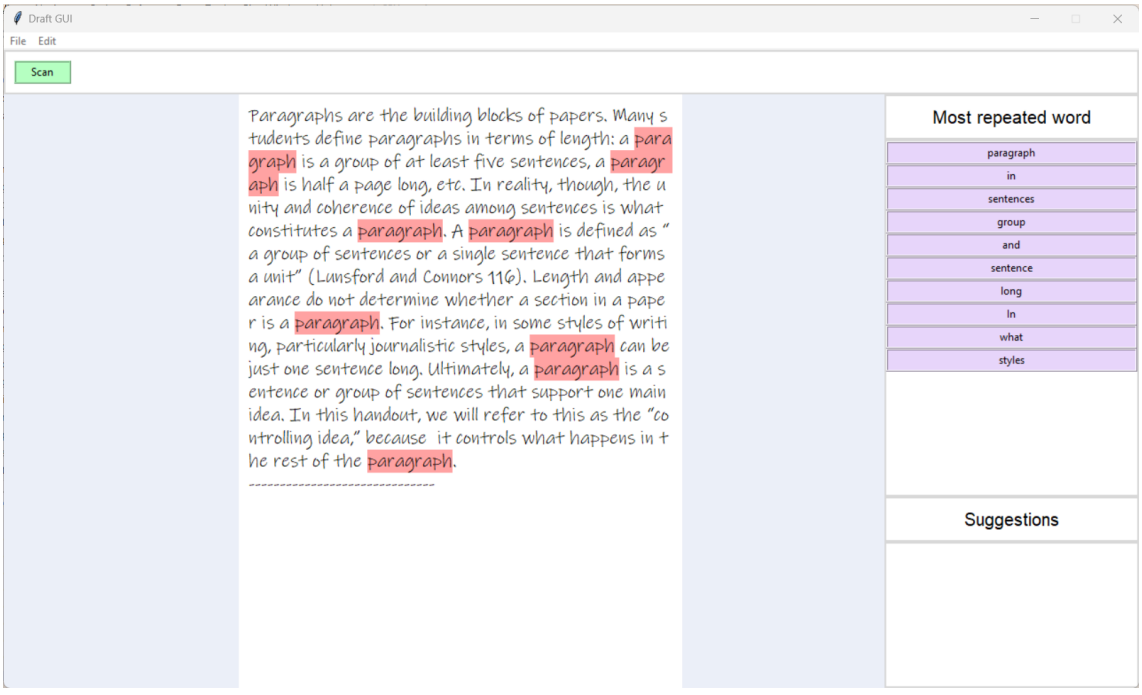

Figure 4.5 Most repeated words shown after clicking 'scan' button

After the user click scan, Parser module will analyze all the text and create n-gram objects for the most frequent words. It can find words from unigram up to trigram. (only unigram is displayed on the UI in the current implementation). The n-gram objects are stored by the program and are ready to be accessed when needed. In our current version, although all the repeated entities are shown and highlighted on the right panel, not all words will have an alternative. (stopwords cannot be substituted)

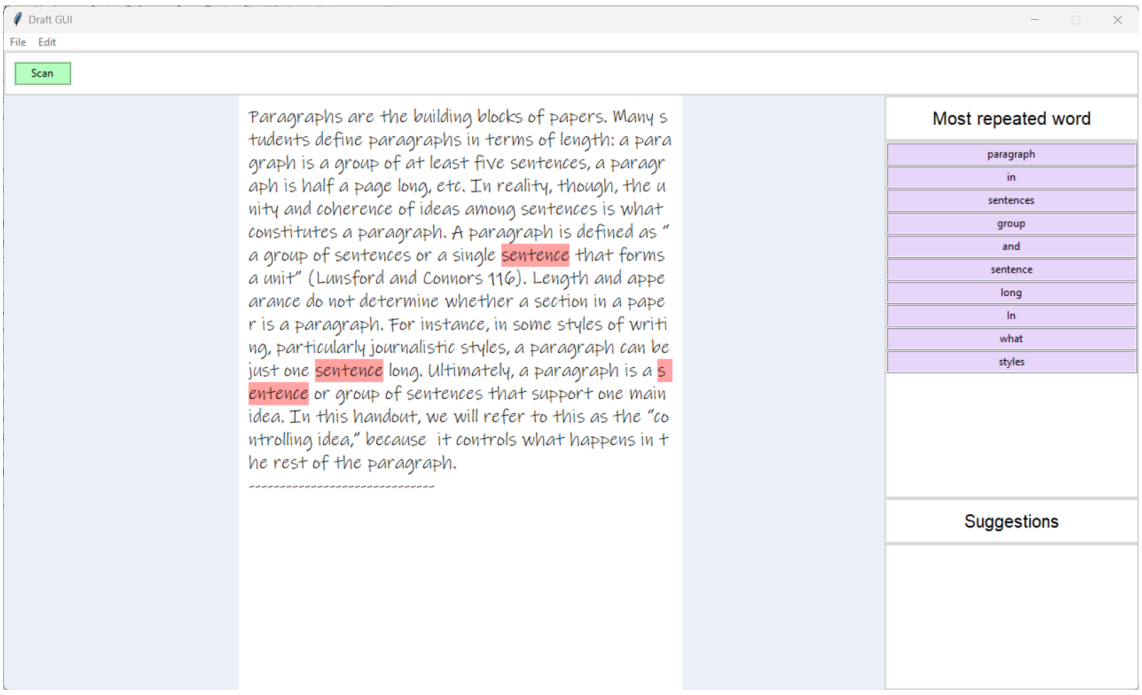

Figure 4.6 Most repeated words shown after clicking 'scan' button

4.2 Problems and Plan

4.2.1 Current issues

Right now we are facing a progress problem, which goes slower than we planned. The actual program still can not be tested since not all of the functions are completed. Because some coding problems such as initializing, or combining code take too much time to perform. The problem affects the project to be slower, and some features' operations are more or less differ from what we declared in the feature list (section 3.2). But surely we are speeding up the process in order to make the program meets the requirement, before adding secondary functions or improving UI's style.

For the data modeling side. We have tried Model finetuning on a custom dataset, but it doesn't work as intended. As a part of our initial plan, we need to retrain the model on a custom dataset focusing on the academic document. The dataset we collected is from the open-source arXiv research paper abstract containing 1.7 million articles with a size of 3GB. Due to a large amount of data and task complexity, the finetuning process was executed on Google Colab Pro+ with the help of GPU acceleration. We finetuned Roberta-large and Roberta-base using Masked Language Modelling. This technique required only the actual text data, so all the unnecessary columns were discarded. During our first attempt, we used an entire 1.7 million sample to train, unfortunately, it quickly consumed a huge amount of compute units and took almost 6 hours to complete. Despite a Pro+ subscription, there was not much room for experimentation due to the resource limit. As a consequence, we needed to shrink the size from 1.7 million to 440K, which reduced resource utilization but accuracy sacrificed accuracy. We were able to achieve 5 attempts. We tried adjusting the parameters namely learning rate, batch size, decay, and the number of epochs. The lowest loss we achieve was 1.29 after 8 epochs. Regardless, the performance of the finetuned model was not satisfactory. It performed no better than the original out-of-the-box model for both Roberta-base and Roberta-large.

After experimenting with Embedding Dropout method, we could not reproduce the result achieved by the original publisher. During the experiment, the produced results were inconsistent and most of them were not usable as a substitution candidate. Thus, we have decided not to use this approach in the final deliverable. Nevertheless, the Sentence Concatenation technique yields decent results in our experimentation. It produces candidates that mostly fit in the original context.
Another behavior we noticed is when the model tried to predict the last word, for example, I like this game because it is <mask>. This task may produce unwanted results because it resembles the next word prediction problem which is not what MLM is good at. The output could be <s>, </s>, .. The workaround may involve swapping the sentence order by putting a masked sentence at the beginning followed by the origins sentence.

4.2.2 Plans

- To complete the application
  First of all, we have to complete coding the application. By continue developing core functions to work properly. For example, currently the scan function returns all 'words' as most repeated words, which is not exactly what we need and requires fixing. After that is to finish creating all other missing feature lists we declared such as the word suggestion function, export or saving a file, importing .txt format, etc. We will prioritize these core features to be completed, to make sure that at least the program is likely done.

- Testing strategy
  We are planning to test the application after all important features are ready to work. The scope of testing is the actual application including all supporting module that combines with the main file which runs the UI and makes the application run as intended. And the test doesn't include the model training files which we use before creating the application but doesn't use in it.

  We will do the functional test on each unit, test each possible input and verify that it's running as expected. Such as the process of opening the file, if we cancel halfway or input other types of files, we will check the result and fix if there's an error occurred. The same goes for component test and system tests.

REFERENCES

1. K. Fosu, 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. T. Pradeep, 2022, "Keyword Extraction Methods from Document in NLP," Available at https://www.analyticsvidhya.com/blog/2022/03/keyword-extraction-methods-from-documents-in-nlp/, [Online; accessed October 2022].

4. 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].

5. A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, and L. KKaiser, "Attention Is All You Need," **arXiv**.

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

7. Maxime, 2019, "What is a Transformer?," Available at https://medium.com/inside-machine-learning/what-is-a-transformer-d07dd1fbec04, [Online; accessed November 2022].

8. W. Zhou, T. Ge, W. Xu, W. Faru, and Z. Ming, "BERT-based Lexical Substitution," **ACL Anthology**.

9. N. Arefyev, B. Sheludko, A. Podolskiy, and A. Panchenko, "Always Keep your Target in Mind: Studying Semantics and Improving Performance of Neural Lexical Substitution," **ACL Anthology**.

10. Grammarly, "About Grammarly," Available at https://www.grammarly.com/about, [Online; accessed October 2022].

11. Prowritingaid, "What makes ProWritingAid different?," Available at https://prowritingaid.com/, [Online; accessed October 2022].

12. Writer.com, "Product Overview," Available at https://writer.com/product/overview/, [Online; accessed October 2022].