TRIBHUVAN UNIVERSITY
INSTITUTE OF ENGINEERING
PULCHOWK CAMPUS

A
PROJECT REPORT
ON

TEXT SUMMARIZATION USING LSA WITH TRANSFORMERS

SUBMITTED BY:
ABHAY NEPAL (075BEI003)
DIPESH TRIPATHI (075BEI013)
GOKARNA ADHIKARI (075BEI014)
KSHITIZ DHAKAL (075BEI015)

SUBMITTED TO:
DEPARTMENT OF ELECTRONICS & COMPUTER ENGINEERING

March, 2023

Page of Approval

TRIBHUVAN UNIVERSIY
INSTITUTE OF ENGINEERING
PULCHOWK CAMPUS
DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING

The undersigned certifies that they have read and recommended to the Institute of Engineering for acceptance of a project report entitled **"Text summarization using LSA with Transformers"** submitted by **Abhay Nepal**, **Dipesh Triapthi**, **Gokarna Adhikari**, **Kshitiz Dhakal** in partial fulfillment of the requirements for the Bachelor's degree in Electronics & Computer Engineering.

....
Supervisor
Prof. Dr. Subarna Shakya
Professor
Department of Electronics and Computer Engineering,
Pulchowk Campus, IOE, TU.

....
Internal examiner
Department of Electronics and Computer Engineering,
Pulchowk Campus, IOE, TU.

....
External examiner
Department of Electronics and Computer Engineering,
Pulchowk Campus, IOE, TU.

Date of approval:

Copyright

Acknowledgments

We would like to express our humble and sincere gratitude to the Department of Electronic and Computer Engineering, Central Campus, Pulchowk for providing us the opportunity to choose the project work which is enlisted in the syllabus of Fourth year as per the Major project IOE, Tribhuvan University. We would like to extend our heartfelt thanks and gratitude to the college for the arrangement and support for working in the academic project. We are particularly indebted to our revered supervisor **Prof. Dr. Subarna Shakya** sir for his constructive and encouraging suggestions regarding this project, explaining the concepts of **Text Summarization**, providing us the guidelines for using software and analysis and motivating us to progress through the project work.
We are thankful to and fortunate enough to get constant encouragement, support and guidance from all respected teachers namely **DR .Nanda Bikram Adhikari, DR. Dibakar Raj Pantha, Dr.Surendra Shrestha, Prof. DR. Ram Krishna Maharjan,Dr.Jyoti Tandukar, Assist. Prof.Lok Nath Regmi,Assist. Prof. Nischal Acharya, Assist.Prof Basanta Joshi, Assist. Prof. Sanjivan Satyal** and all who, directly and indirectly have lent their helping hand and suggestions in this venture.

Abstract

This project endeavors to present an implementation of a text summarization method employing the amalgamation of Latent Semantic Analysis (LSA) with Transformers. The primary objective of the proposed approach is to create a brief summary of an input text while retaining its fundamental meaning. The summarization model is assessed through two metrics, namely BLEU scores and ROUGE scores, which are utilized to gauge the model's efficacy in generating a succinct and accurate summary.

The project comprises several steps, including text preprocessing, feature extraction using LSA, and summary generation using Transformers. The resulting summary is evaluated by comparing it against a reference summary, and the quality of the summary is measured by the BLEU metric and ROUGE scores.

The evaluation results reveal that the proposed approach yields high scores on both metrics, indicating its effectiveness in generating precise and concise summaries. Moreover, the project incorporates an analysis of the impact of various parameters on the performance of the summarization model, thereby providing valuable insights into the optimal parameter settings for the proposed method.

In conclusion, this project exemplifies the potential of leveraging LSA with Transformers for text summarization and furnishes a practical implementation for producing high-quality summaries.

Keywords: *Latent Semantic Analysis (LSA), Transformers, Text summarization, BLEU scores, ROUGE scores*

Contents

List of Figures

List of Tables

List of Abbreviations

NLP	Natural Language Processing
ML	Machine Learning
AI	Artificial Intelligence
LSA	Latent Semantic Analysis
RNN	Recurrent Neural Network
ROUGE	Recall-Oriented Understudy for Gisting Evaluation
BLEU	Bilingual Evaluation Understudy

1. Introduction

1.1 Background

The main objective of this project is to develop and train a machine learning model that is capable of taking in a block of text and generating a summary along with significant keywords in a suitable format.

To achieve our goal, we will be utilizing various natural language processing (NLP) techniques and machine learning algorithms to train our model. Our focus will be on generating a relevant summary of text , particularly in fields such as journalism, fact-checking, news generation, and lecture summarization.

Our model will also be able to summarize formal articles that are free of grammatical errors and are precise. The ability to generate accurate and concise summaries from formal texts is particularly useful for academic research and other domains where precision and accuracy are paramount.

While there are some existing text summarization models that can produce decent output, the scope of our project is unique, and we are confident that with hard work and guidance from our mentors, we will produce a model that stands out among the current models. We believe that our approach, which combines NLP techniques and machine learning algorithms, will yield a model that can provide high-quality summaries of a variety of texts.

In summary, our project aims to develop a cutting-edge machine learning model that can accurately summarize a wide range of texts, including unstructured and formal content. With our focus on utilizing NLP techniques and machine learning algorithms, we are confident that we can achieve this goal and create a model that stands out among the existing models.

Automated summary generation is a crucial tool for extracting information from lengthy texts or other forms of text data. With the sheer amount of time it takes to go through hours of text, generating summaries automatically can be incredibly time-effective. In today's fast-paced world, time is a valuable commodity, and the ability to quickly generate accurate summaries is a highly desirable feature.

1.2 Problem Statement

Develop an automated system that can accurately and efficiently generate a concise summary of a given text while preserving its most important information. This includes addressing challenges such as identifying the key topics and relevant information, detecting sentence boundaries, and evaluating the quality of the generated summary. The goal is to create a system that can extract the most important content from a lengthy text, making it easier and faster for users to understand the key points without having to read the entire document. Currently, most practical text summarization techniques are extractive, meaning they extract the most relevant sentences or phrases from the original text and combine them into a summary. In contrast, abstraction-based summarization, which involves creating new sentences that capture the essence of the original text, is a more challenging task and an active area of research. Despite these challenges, automated text summarization has the potential to greatly improve efficiency and productivity, particularly in fields such as journalism, research, and news reporting.

1.3 Objectives

- To summarize the text.
- To identify the important phrases and words that occurs in the text.
- To generate recommendation for heading.

1.4 Application

Our project has versatile scope and serves numerous applications. The project can be used as various useful tools.

- Title generation
- Redundancy elimination
- Information retrieval from articles
- News generation and summarization

2. Literature review

2.1 Related work

The summarization approach can be broadly classified into extractive and abstractive. The extractive approach focuses on choosing how paragraphs, and important sentences can be chosen in such a way that it produces the set of sentences that it closely approaches the meaning of the original text [2].Text summarization is the process of creating a shorter version of a longer text while retaining its key information. It has become an important tool for information retrieval and management, as well as for natural language processing applications. Text summarization techniques can be broadly categorized as extractive or abstractive summarization.

Extractive summarization involves selecting a subset of sentences from the source text to create a summary. This approach is often used for news articles or scientific papers, where the sentences are usually well-structured and contain important information. One popular approach to extractive summarization is the use of graph-based models. Carbonell and Goldstein [3] introduced a diversity-based re-ranking method called Maximal Marginal Relevance (MMR) to produce summaries, while Erkan and Radev [4] proposed a graph-based model called LexRank to identify salient sentences for summary generation. Abstractive summarization, on the other hand, involves generating new sentences that capture the essence of the source text. This approach is more challenging than extractive summarization, as it requires natural language generation and understanding. More recently, deep learning models such as sequence-to-sequence (Seq2Seq) models have shown promising results in abstractive text summarization. Nallapati et al. [5] proposed a Seq2Seq model with attention mechanism for abstractive summarization, while See et al. [6] introduced a pointer-generator network that can copy words from the source text to produce more accurate summaries.

To guide the generation of summaries, researchers have proposed various approaches. Zhang et al. [7] proposed a Key Information Guide Network that identifies key information in the source text and uses it to guide the summarization process. Other approaches include using reinforcement learning [8] and incorporating discourse structure [9].

Overall, text summarization is an active area of research, and several approaches have been proposed over the years. Radev et al. [10] provided an overview of the state-of-the-art in the field, while Nenkova and McKeown [11] presented a comprehensive survey of automatic summarization techniques. Wan et al. [12] provided a survey of automatic text summarization, covering both extractive and abstractive approaches. However, the problem of text summarization remains a challenging one, and there is still much work to be done to improve the effectiveness and efficiency of summarization techniques. The summarization of the text can be described as a two-step process: building from source text a source representation and summary generation forming representation from the source representation build in the first step synthesizing the output summary text [13]. LSA is converting the metrics matrix to a single value decomposition(SVD) matrix and comparing the correlation for extraction of sentences [14]. Extractive Summarization is a method for determining salient text units (typically sentences) by looking at the text unit's lexical and statistical relevance or by matching phrasal patterns [15]. In the extractive method, the sentences and words are dragged and combined them to produce arguably a comprehensive summary [16]. Abstractive Summarization is a method for novel phrasing describing the content of the text which requires heavy machinery from natural language processing, including grammar and lexicons for parsing and generation [15]. A good sentence reduction system can improve the conciseness of generated summaries significantly (typically by 44.2 %) [17].

2.2 Related Theory

2.2.1 Text to summary

There are different features on which we can classify a text summarization system which is illustrated on the chart given below[18]:

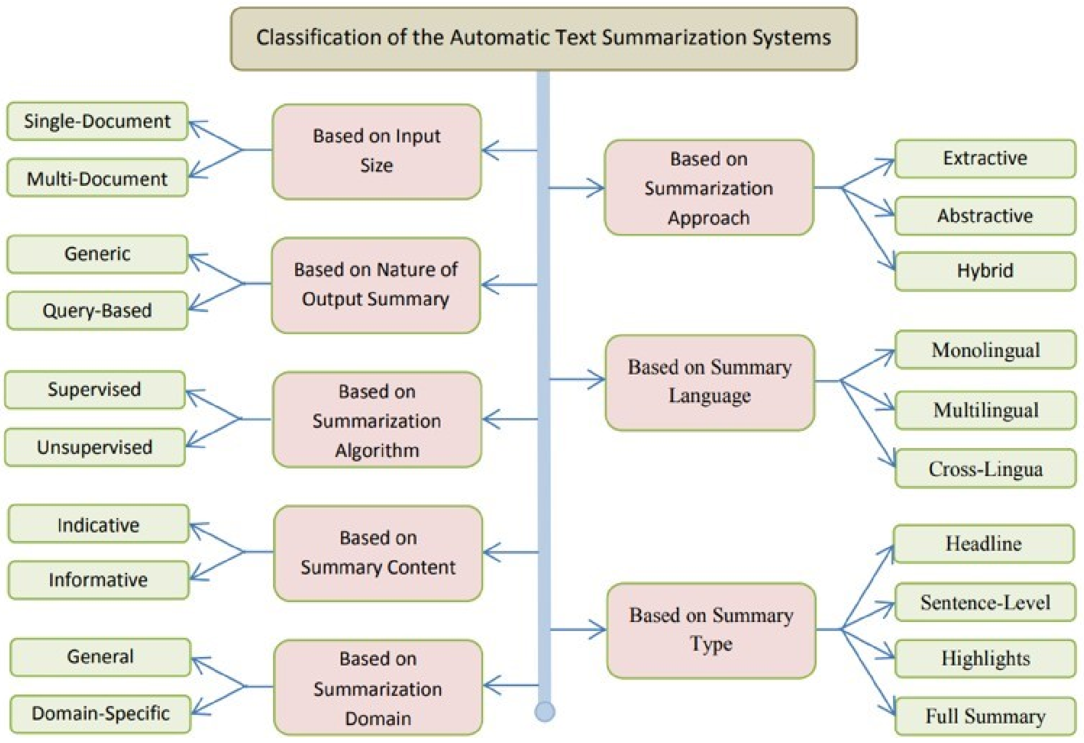


Figure 2.1: Classification of text summarization techniques

Extractive Text summary

In Extractive Summarization, we identify important phrases or sentences from the original text and extract only these phrases from the text. These extracted sentences would be the summary.

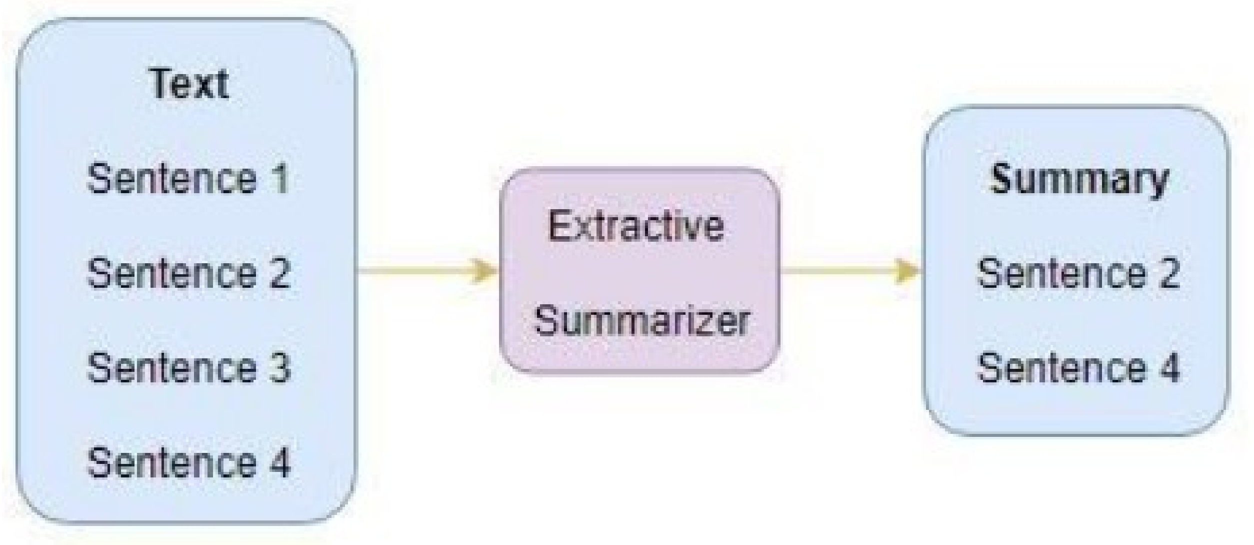


Figure 2.2: Extractive Text Summarization

Abstractive Text summary

In the Abstractive Summarization approach, we work on generating new sentences from the original text. The abstractive method contrasts with the approach that was described above. The sentences generated through this approach might not even be present in the original text.

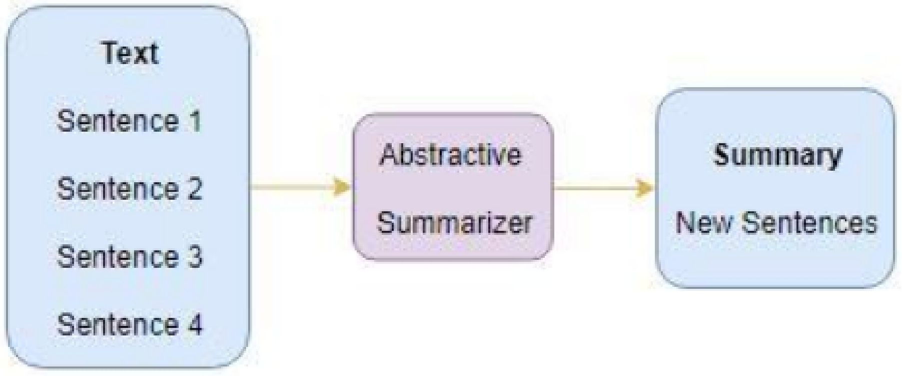


Figure 2.3: Abstractive Text Summarization

Hybrid

Summarizer that combines the advantages of the abstractive and extractive approaches to summarization and implements a solution to the opinion holder attribution problem.

3. Methodology

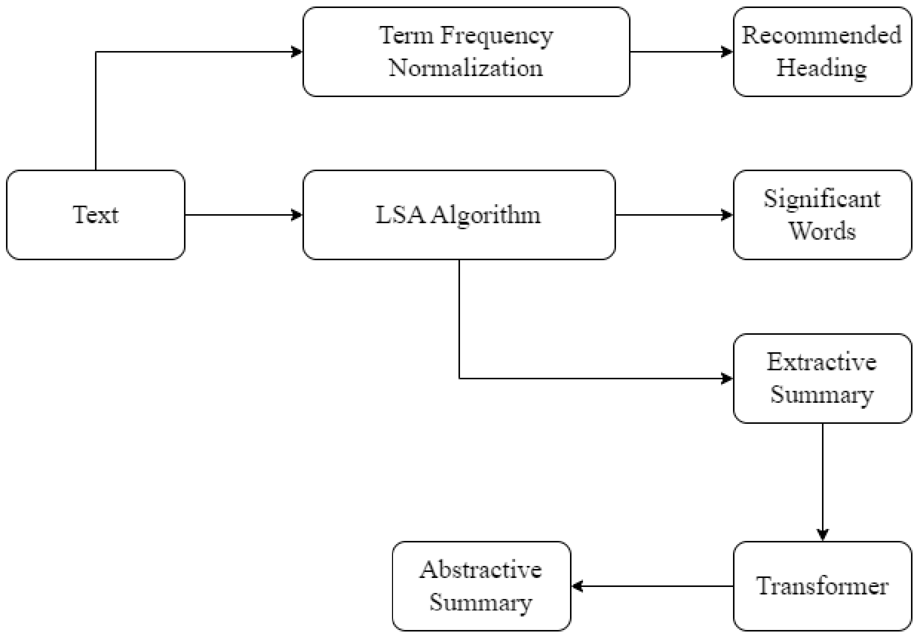


Figure 3.1: Methodology Block Diagram

The text is obtained either directly from the already stored database. The obtained text is then fed into LSA model after preforming necessary pre-processing to obtain an extractive summary of the text. The extractive text summary obtained is fed into the abstractive summary generation model and then the abstractive text summary is obtained from which the title generation and data mining are performed.
Each process in detail is further explained in this chapter.

3.1 Text To Summary

We intend to use the *hybrid model* initially for the summarization, concluded from the literature review. An overview of the text summarization approaches that will be implemented can be understood from the block diagram below:

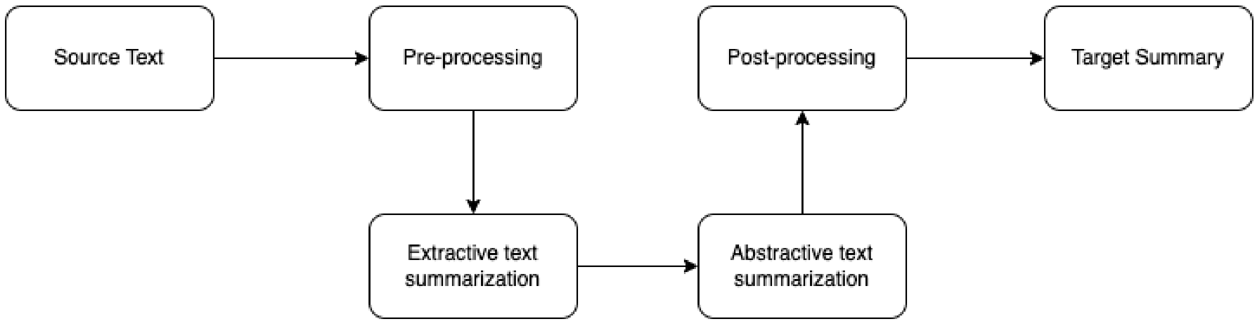


Figure 3.2: Hybrid Text Summarization Approach

3.1.1 Extractive Summary

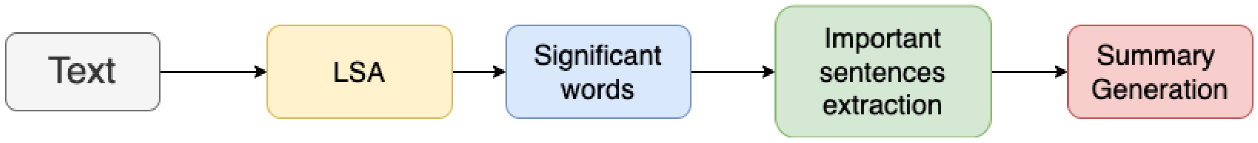


Figure 3.3: Extractive Summarization approach

Text Input: The process begins with a text input, which is a document or a piece of text that needs to be summarized.
LSA: LSA is a statistical method used to identify the underlying structure of the text and to reduce the dimensionality of the text representation. The LSA algorithm creates a matrix of word occurrences and maps the words into a lower-dimensional space that captures the semantic relationships between them.
Significant Word: In LSA, each word is assigned a weight that reflects its importance in the text. The weight of a word is determined by its frequency in the text and its similarity to other words in the text. The most important words in the text are referred to as significant words.
Significant Sentence Mapping with Significant Word: After identifying significant words, the next step is to map each significant word to the sentences in which it occurs. This is done by assigning a score to each sentence based on the occurrence and weight of the significant words in the sentence. This mapping helps in identifying the most important sentences in the text.
Generate the summary with required compression ratio: Finally, the summary is generated by selecting the most important sentences based on the scores assigned in step 4. The compression ratio is used to determine the length of the summary.

For an extractive summary, we intended to use Latent semantic analysis(LSA) with Singular

Value Decomposition(SVD).

Latent semantic analysis

Latent semantic analysis (LSA) is a mathematical method for computer modeling and simulation of the meaning of words and passages by analysis of representative corpora of natural text. LSA closely approximates many aspects of human language learning and understanding. It supports a variety of applications in information retrieval, educational technology, and other pattern recognition problems where complex wholes can be treated as additive functions of component parts. LSA assumes that words that are close in meaning will occur in similar pieces of text.
The underlying idea is that the aggregate of all the word contexts in which a given word does and does not appear provides a set of mutual constraints that largely determines the similarity of meaning of words and sets of words to each other[19].

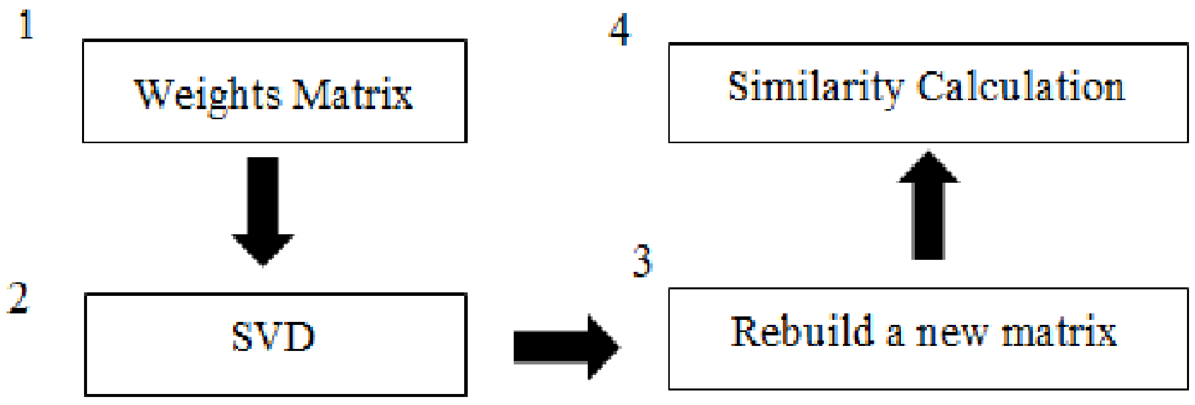


Figure 3.4: Block diagram of LSA

The LSA uses a long-known matrix-algebra method, Singular Value Decomposition (SVD).

Singular Value Decomposition (SVD)

The Singular Value Decomposition (SVD) of a matrix is a factorization of that matrix into three matrices. It has some interesting algebraic properties and conveys important geometrical and theoretical insights about linear transformations. It also has some important applications in data science.

The SVD of mxn matrix A is given by the formula :

$$A = UWV^T \tag{3.1}$$

where:
$U = m * n$ matirix of the orthogonal eigenvectors of AA^T
V^T =transpose of an nxn matrix containing the orthonormal eigenvectors of $A^T A$
W = a n*n diagonal matrix of the singular values which are the square roots of the eigenvalues of $A^T A$

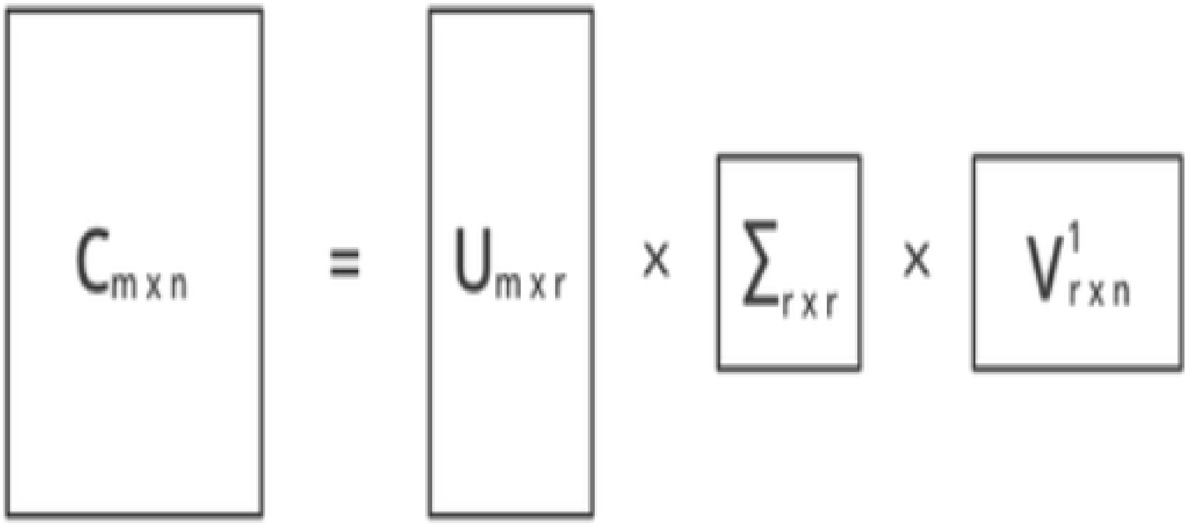


Figure 3.5: SVD Process

3.1.2 Abstractive summary

An abstract is a brief summary of a research article, thesis, review, conference proceeding, or any in-depth analysis of a particular subject and is often used to help the reader quickly ascertain the text's purpose[20]. Unlike the extractive summary, it includes new words and phrases while summarizing a text. The abstract should be able to convey the main results and conclusions of an article.

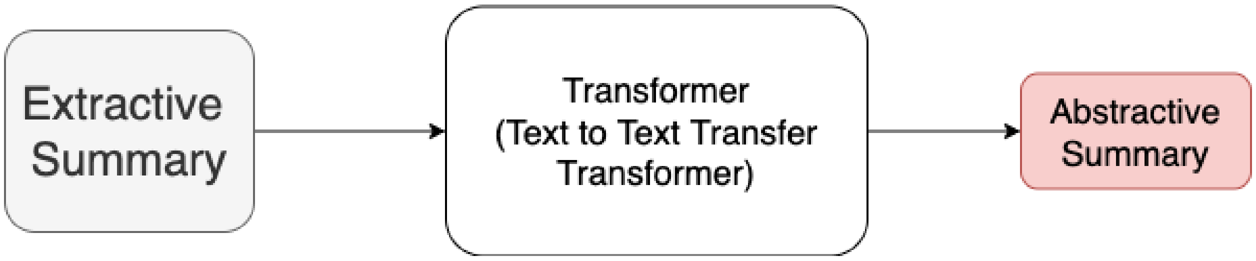


Figure 3.6: Abstractive Summarization approach

Abstractive Text Summarization is the task of generating a short and concise summary that captures the salient ideas of the source text. The generated summaries potentially contain new phrases and sentences that may not appear in the source text.
RNN-based encoder-decoder models for abstractive summarization have achieved good performance on short input and output sequences[21]. Neural network-based methods for abstractive summarization produce outputs that are more fluent than other techniques but can be poor at the content selection. The content selector can be used for bottom-up attention that restricts the ability of abstractive summarizers to copy words from the source. The combined bottom-up summarization system leads to improvements in ROUGE scores[22].

Transformers

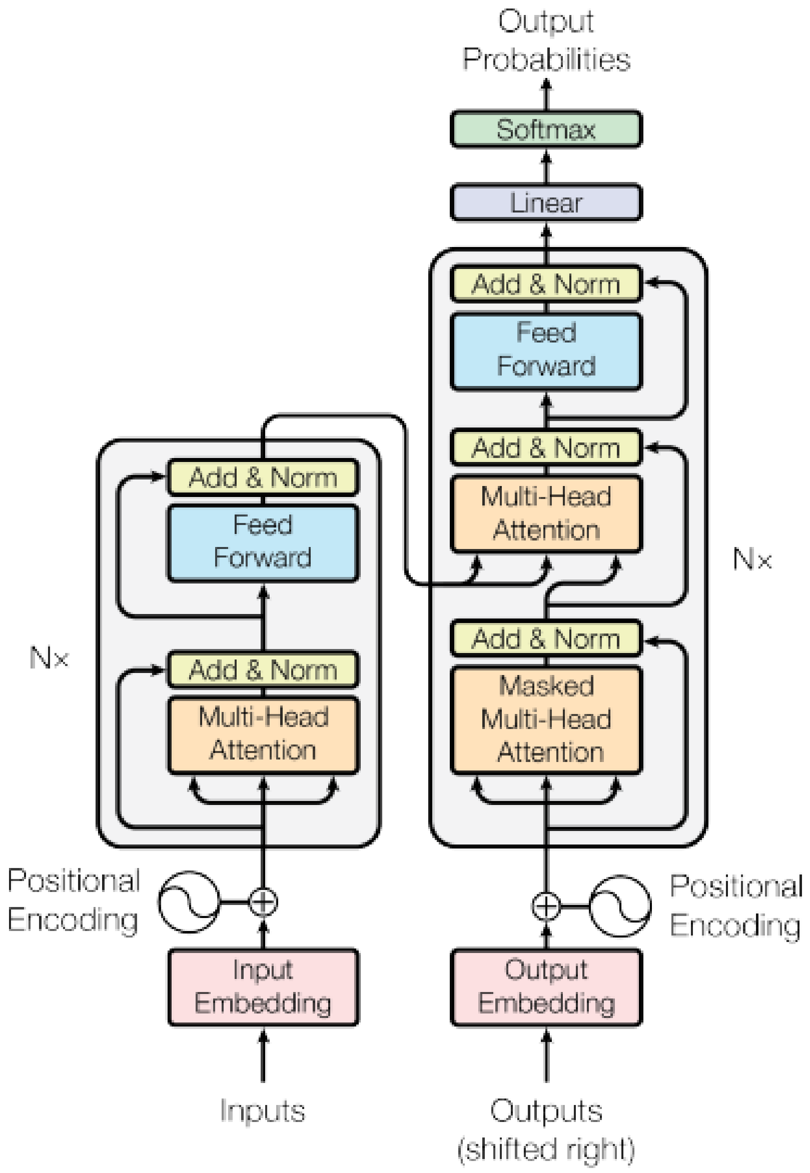


Figure 3.7: Transformer Block Diagram [1]

Transformers are a type of deep learning model widely used in Natural Language Processing (NLP) tasks. They have first introduced in the paper "Attention is All You Need" by Vaswani et al. in 2017 [1]. The key idea behind Transformers is the self-attention mechanism, which allows the model to weigh the importance of different input tokens in computing the output. Transformers are a type of deep learning model used in Natural Language Processing (NLP)

tasks. They are designed to handle sequences of varying lengths and are well-suited for tasks such as machine translation, text classification, and question-answering.
A Transformer model consists of two main components: an encoder and a decoder. The encoder takes in a sequence of input tokens (such as words in a sentence) and generates a continuous representation of the input sequence, called the context vector. The decoder then takes the context vector and generates the final output, such as a translation or a classification label.
The key innovation in Transformers is the self-attention mechanism, which allows the model to weigh the importance of different input tokens in computing the output. In traditional recurrent neural networks (RNNs), the hidden state is updated based on the previous hidden state and the current input. In Transformers, however, the hidden state is generated based on a weighted sum of all the input tokens, where the weights are determined by the self-attention mechanism. This allows the model to attend to relevant information in the input sequence and capture long-range dependencies.
The encoder and decoder in Transformers are both composed of several layers of multi-head self-attention and fully connected layers. The multi-head self-attention mechanism allows the model to attend to different parts of the input sequence simultaneously, improving its ability to capture complex relationships.
Transformers have proven to be highly effective in NLP tasks and have achieved state-of-the-art results on a variety of benchmarks. They are also easily parallelizable and can be trained on large amounts of data, making them well-suited for large-scale NLP tasks.
We used the CNN dailymail dataset with the data which was collected and processed by our group and used it to train the transformer model. The dataset is also used to make the dictionary using the vocabs and also for the tokenizer and detokenizer.
In summary, Transformers are a type of deep learning model that have revolutionized NLP by introducing the self-attention mechanism and a flexible architecture composed of encoders and decoders. They have achieved state-of-the-art results in many NLP tasks and have shown promise in other domains as well.

3.1.3 Loss Functions

Categorical Cross-Entropy Loss

This loss function is used in multi-class classification problems. It measures the difference between the predicted probability and the actual probability of the target class.

Sparse Categorical Crossentropy Loss

Sparse Categorical Crossentropy is a loss function used in machine learning for multi-class classification problems. It is commonly used when the labels are integers instead of one-hot encoded vectors.

In traditional Categorical Crossentropy, the model outputs a probability distribution over all possible classes for each input sample, and the target labels are one-hot encoded vectors that indicate the correct class for each sample. The loss function then calculates the difference between the predicted probability distribution and the true probability distribution.
In contrast, with Sparse Categorical Crossentropy, the target labels are represented as integers that correspond to the correct class for each input sample. The model still outputs a probability distribution over all possible classes, but the target labels are not one-hot encoded.

3.2 Model Development

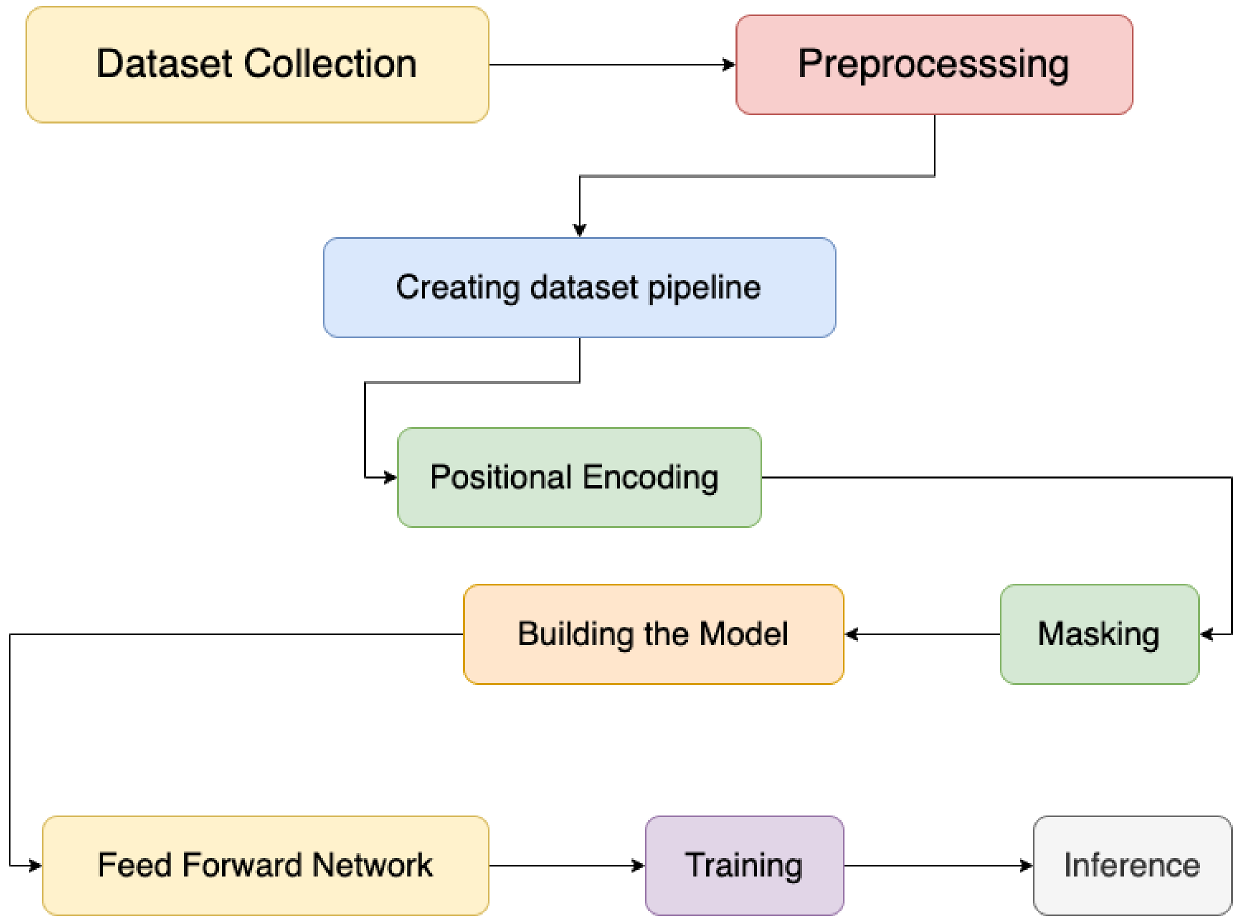


Figure 3.8: Overall Model Development Process

3.2.1 Dataset Collection

The process of collecting data for our project involved extracting information from various open source dataset collections, such as Kaggle and CNN/Daily Mail. These collections provided a wealth of data, covering a wide range of topics that were relevant to the project. In addition to using these pre-existing datasets, we also manually added many text paragraphs and summaries to the dataset. This process involved carefully selecting relevant information and creating new text to supplement the existing data.

By combining these various sources of data, we were able to create a robust dataset that would be useful for training our machine learning models. The dataset contained a diverse set of data, representing a variety of perspectives and viewpoints on the topics of interest.

To ensure the quality of the data, we carefully curated and reviewed each text paragraph and summary before adding it to the dataset. This involved checking for accuracy, relevance,

and consistency with the existing data.
Throughout the data collection process, we remained mindful of the ethical implications of using data that was collected from open sources. We took care to respect the privacy of individuals and organizations that were represented in the data, and we adhered to all relevant legal and ethical guidelines.
Overall, the process of collecting data for our machine learning project was a challenging but rewarding endeavor. By using a combination of existing datasets and manually adding new data, we were able to create a high-quality dataset that would be useful for training and evaluating our machine learning models.

3.2.2 Pre-processing

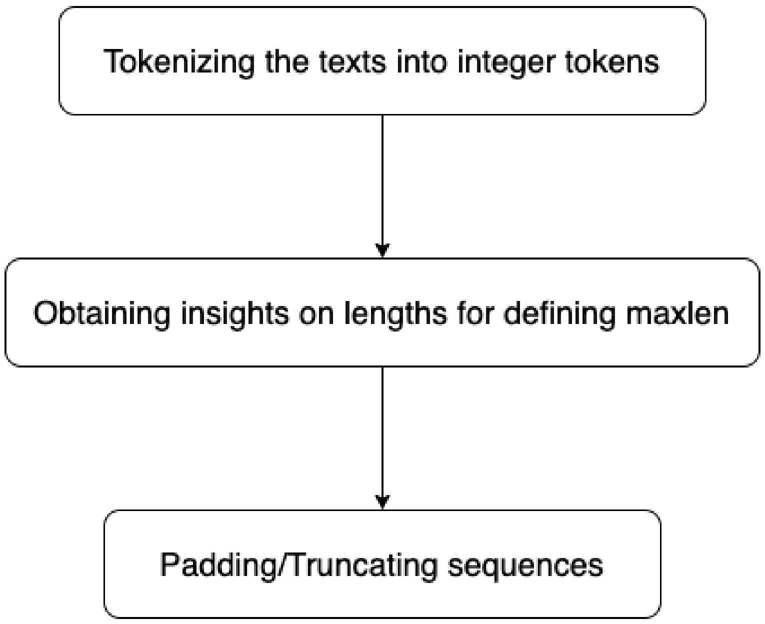

Tokenizing the texts into integer tokens
Obtaining insights on lengths for defining maxlen
Padding/Truncating sequences

Figure 3.9: Pre-processing for model development

Tokenizing the texts into integer tokens

Tokenizing text is the process of converting raw text into a list of smaller units, or tokens, which can then be processed by machine learning models. In the case of converting text into integer tokens, each unique token is assigned a unique integer value.
We use 'tokenizer' module of the nltk library for tokenizing text into integer tokens.

Obtaining insights on lengths for defining maxlen

We calculate the average and standard deviation of sequence lengths: Computing the mean and standard deviation of the sequence lengths can provide a rough estimate of the central tendency and the spread of the sequence lengths. And the maxlen value was set to slightly higher than third quartile to avoid outliers.

Padding/Truncating sequences for identical sequence lengths

Padding

When padding a sequence to a desired length, you add special padding tokens (usually zeros) to the end of the sequence until it reaches the desired length. This is commonly used when training models that require sequences of equal length, such as recurrent neural networks (RNNs) or convolutional neural networks (CNNs).

Truncation

When truncating a sequence to a desired length, you remove tokens from the end of the sequence until it reaches the desired length. This is commonly used when dealing with sequences that are too long to be processed efficiently or when comparing sequences of different lengths.
Note that when padding or truncating sequences, it's important to keep the overall sequence length within a reasonable range for the task at hand. Padding or truncating sequences too much can lead to loss of important information and negatively impact model performance, while not padding or truncating enough can lead to memory errors or inconsistent input sizes for the model.

3.2.3 Creating Dataset Pipeline

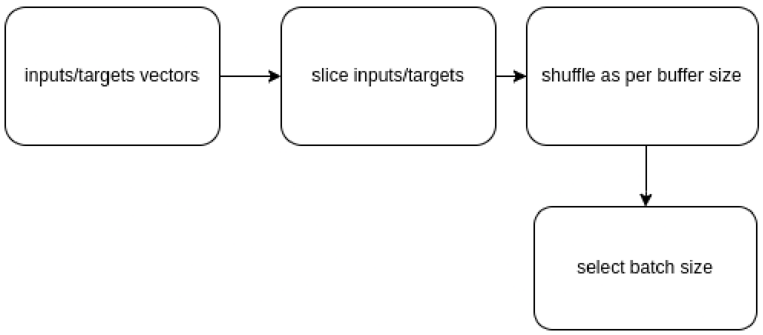


Figure 3.10: Creating Dataset Pipeline Block Diagram

Input/Target Vectors: In some cases, the input and target sequences can be very long. In such cases, it can be helpful to split them into smaller, more manageable blocks. This is done by creating a list of input/target vectors, where each vector represents a block of the input/target sequence.
Slicing Input or Target: Slicing the input or target data can be useful in cases where only a portion of the data is required for training. This can help reduce the memory requirements of the model.
Shuffling as per buffer size: Shuffling the data can help prevent the model from overfitting to the training data. In the case of large datasets, shuffling the entire dataset may not be practical due to memory constraints. In such cases, shuffling can be done on a smaller buffer size.
Selecting Batch Size: The batch size is an important hyperparameter that determines the number of input sequences processed in each training iteration. The optimal batch size may vary depending on the size of the dataset and the available memory. Therefore, it is important to experiment with different batch sizes to determine the optimal value.

3.2.4 Positional Encoding

Positional encoding is a technique used in transformers to inject positional information into the input sequence of tokens. Since transformers do not have any inherent notion of position or order in the input sequence, positional encoding is necessary to help the model understand the order in which the tokens appear.
The positional encoding is added to the input embeddings of each token, and it consists of a vector that encodes the position of the token in the sequence. The vector is added to the embedding vector of the token, so that the model has access to both the token's meaning and its position in the sequence.
There are different ways to compute the positional encoding vector, but one common approach is to use trigonometric functions. Specifically, for a token at position i in the sequence, the positional encoding vector is defined as:

$$PE_{i,j} = \begin{cases} \sin\left(\frac{i}{10000^{\frac{2j}{d}}}\right) & \text{if } j \text{ is even} \\ \cos\left(\frac{i}{10000^{\frac{2j}{d}}}\right) & ifj \text{ is odd} \end{cases} \tag{3.2}$$

Here, d is the dimensionality of the input embeddings (i.e., the number of features in each embedding vector), and j is the index of the feature in the embedding vector. The positional encoding vector has the same dimensionality as the input embeddings.
By adding the positional encoding vectors to the input embeddings, the transformer is able to distinguish between tokens that appear in different positions in the sequence, even if the

tokens themselves have the same meaning. This allows the model to capture more complex patterns in the input sequence, and it is a key component of the transformer architecture.

3.2.5 Masking

Masking refers to the process of selectively hiding certain tokens or positions in the input sequence during training or inference. This is typically done to ensure that the model cannot cheat by peeking ahead at tokens that it should not be able to see.

There are two types of masking commonly used in transformers: padding masking and sequence masking.

Padding masking is used to hide padding tokens that are added to the input sequence to make it uniform in length. Since the input sequence can have variable length, it is often padded with special tokens (e.g., zeros) to ensure that all sequences have the same length. However, these padding tokens should not be processed by the model, since they do not carry any meaningful information. Therefore, padding masking is used to set the attention scores for the padding tokens to zero, so that the model ignores them during training.

Sequence masking is used to hide tokens in the input sequence that should not be visible to the model. For example, during language modeling, the model should not be able to see tokens from the future when predicting the next token in the sequence. Therefore, sequence masking is used to set the attention scores for future tokens to zero, so that the model cannot attend to them during training or inference.

Both padding masking and sequence masking are implemented as binary masks that are applied to the attention scores before softmax normalization. By setting certain entries in the mask to zero, the model is prevented from attending to certain tokens or positions in the input sequence. This helps to ensure that the model learns to make predictions based only on the information that is available at each step, and not on information that it should not have access to.

3.2.6 Building the model

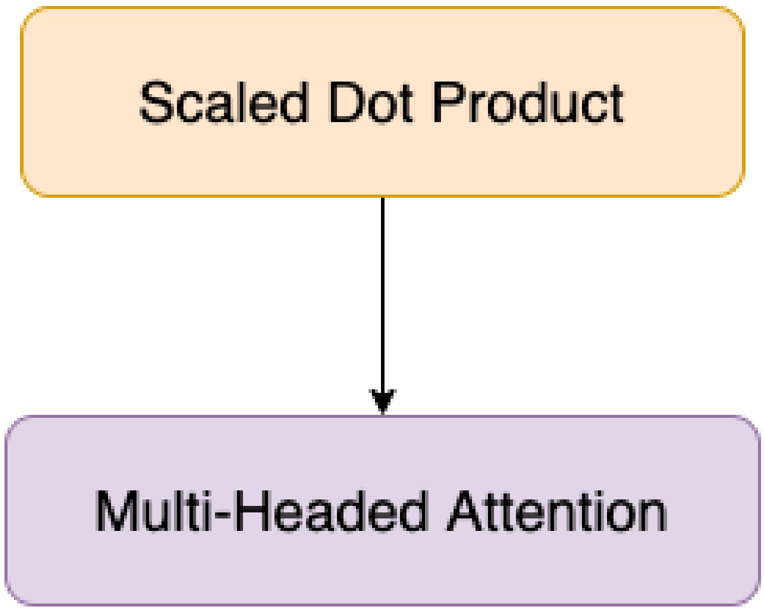


Figure 3.11: Building the model

Scaled Dot Product

Scaled dot product attention is a type of self-attention mechanism used in the transformer architecture. It is used to compute the attention scores between each query and key pair in a sequence of input data.
The attention score between a query and a key is computed as the dot product of the query and the key, which measures how similar they are. However, in order to prevent the dot product from becoming too large, the scores are scaled by the square root of the dimension of the key vectors. This scaling ensures that the dot product is not too large or too small, which can help to stabilize the training process. Formally, given a set of queries Q and a set of keys K, the scaled dot product attention is computed as follows:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) V \tag{3.3}$$

where Q, K, and V are the query, key, and value vectors, respectively, $\hat{\text{T}}$ denotes matrix transpose, and d_k is the dimension of the key vectors. The output of the attention operation is a weighted sum of the value vectors, where the weights are given by the attention scores. Scaled dot product attention is a computationally efficient and effective mechanism for modeling dependencies between the elements in a sequence, and it has been widely used in many NLP applications.

Multi-head Attention

Multi-head attention is a type of attention mechanism used in the transformer architecture. It extends the scaled dot product attention mechanism to enable the model to attend to different aspects of the input data in parallel.
In multi-head attention, the input queries, keys, and values are transformed into multiple smaller dimensional spaces using linear projections, resulting in h different "heads". Each head computes the attention scores separately, which allows the model to capture different patterns in the input data. The outputs of the different heads are then concatenated and transformed again to produce the final output.
Formally, given a set of input queries Q, keys K, and values V, the multi-head attention is computed as follows:

$$MultiHead(Q, K, V) = Concat(head_1, ..., heada_h)W^O \tag{3.4}$$

where:

$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V). \tag{3.5}$$

Here, W_i^Q, W_i^K, and W_i^V are learnable projection matrices for the i-th head, and W^O is a learnable output projection matrix.
The Concat operation concatenates the outputs of the different heads, and the final output is obtained by applying a linear transformation to the concatenated output.
Multi-head attention allows the transformer to model different relationships between the input elements and to attend to different aspects of the input in parallel. This leads to improved performance on a wide range of natural language processing tasks, including language modeling, machine translation, and text classification.
For our transformer blocks, We used multihead attention witn num_heads = 8 that is attention is calculated for a word in 8 different contexts.

3.2.7 Feed Forward Network

Fundamental Unit of Transformer encoder

The encoder block is one of the main components of the transformer architecture, used for processing input data in a self-attention mechanism. The encoder block consists of several layers of self-attention and feed-forward neural networks.
The input to the encoder block is a sequence of embeddings, which could represent words, tokens, or any other kind of input. The embeddings are first passed through a self-attention layer, where each token in the sequence is compared to every other token to compute a set of attention weights. The attention weights are used to compute a weighted sum of the input

embeddings, which is then passed through a feed-forward neural network to obtain a new set of embeddings. This process is repeated for each layer in the encoder block.
More formally, the computation in an encoder block can be described as follows:

1. The input embeddings are first processed through a multi-head self-attention mechanism. This allows the model to attend to different parts of the input sequence and capture long-range dependencies. The output of this layer is a set of attention heads, which are then concatenated and passed through a linear layer to obtain a set of new embeddings.

2. The new embeddings are then passed through a feed-forward neural network with a ReLU activation function. This provides a non-linear transformation of the input embeddings and enables the model to learn complex representations of the input data.

3. The output of the feed-forward network is then passed through another linear layer to obtain a final set of embeddings.

The encoder block is typically stacked multiple times to process the input data with multiple layers of self-attention and non-linear transformations. The output of the final encoder block can be used as input to a decoder block or as a representation of the input data for downstream tasks. The encoder block is a key component of the transformer architecture and has been used for a wide range of natural language processing tasks, including language modeling, machine translation, and text classification.

Fundamental Unit of Transformer decoder

The decoder block is another key component of the transformer architecture, used for generating output data based on the input data and previously generated output. The decoder block is similar to the encoder block, but it has some important differences in its design and operation.
The input to the decoder block is a sequence of target embeddings, which are typically derived from the same vocabulary as the input embeddings. In addition to the target embeddings, the decoder block also receives the output of the final encoder block as input. The encoder output is used to compute attention weights for the target embeddings, allowing the model to attend to different parts of the input sequence and capture relevant information for the generation of the target sequence.
Like the encoder block, the decoder block is typically composed of multiple layers of self-attention and feed-forward neural networks. The computation in a decoder block can be described as follows:

1. The input target embeddings are first processed through a multi-head self-attention mechanism, where each target embedding is compared to every other target embedding in the sequence. The attention weights are computed based on the encoder output and used to compute a weighted sum of the encoder output embeddings. This allows the model to attend to different parts of the input sequence and capture relevant information for the generation of the target sequence.

2. The output of the self-attention layer is passed through a second multi-head self-attention mechanism, where the model attends to different parts of the target sequence to capture dependencies between different target embeddings.

3. The output of the second self-attention layer is then passed through a feed-forward neural network with a ReLU activation function. This provides a non-linear transformation of the decoder input embeddings and enables the model to learn complex representations of the target data.

4. The output of the feed-forward network is then passed through another linear layer to obtain a final set of target embeddings, which are used to generate the final output sequence.

The decoder block is typically used in conjunction with the encoder block to generate a sequence of target outputs based on an input sequence. The output of the final decoder block can be used as the final output sequence, or as a representation of the generated output for downstream tasks. The decoder block is a key component of the transformer architecture and has been used for a wide range of natural language processing tasks, including machine translation, summarization, and text generation.

Encoder consisting of multiple EncoderLayer(s)

In natural language processing (NLP), an encoder is a neural network component that converts a sequence of input tokens (such as words or characters) into a fixed-length vector representation. The fixed-length representation is often used as an input to other downstream models or tasks.
The Transformer architecture, which is widely used in NLP, consists of an encoder and a decoder. The encoder component typically consists of multiple encoder layers, with each layer consisting of two sub-layers:

1. Self-attention layer: This layer computes the attention scores between every pair of input tokens in the sequence and generates a weighted sum of the input tokens based on

their importance scores. This enables the model to capture the contextual relationships between different tokens in the input sequence.

2. Feedforward layer: This layer applies a simple feedforward neural network to the output of the self-attention layer, which introduces non-linearity and helps the model to learn more complex representations of the input sequence.

Each encoder layer in the Transformer architecture also includes residual connections and layer normalization, which help with the training stability and performance.
By stacking multiple encoder layers on top of each other, the Transformer model is able to capture increasingly complex relationships between the input tokens, leading to more effective representations and improved performance.
We used num_layers value of 4 for the encoder.

Decoder consisting of multiple DecoderLayer(s)

In natural language processing (NLP), a decoder is a neural network component that takes as input a fixed-length vector representation generated by the encoder and produces a sequence of output tokens (such as words or characters).
The Transformer architecture, which is widely used in NLP, consists of an encoder and a decoder. The decoder component typically consists of multiple decoder layers, with each layer consisting of three sub-layers:

1. Self-attention layer: This layer computes the attention scores between every pair of output tokens generated by the decoder and generates a weighted sum of the output tokens based on their importance scores. This enables the model to capture the dependencies between different output tokens in the sequence.

2. Encoder-decoder attention layer: This layer computes the attention scores between the output tokens generated by the decoder and the input tokens generated by the encoder. This allows the decoder to focus on different parts of the input sequence based on the context of the current output token being generated.

3. Feedforward layer: This layer applies a simple feedforward neural network to the output of the previous two sub-layers, which introduces non-linearity and helps the model to learn more complex representations of the output sequence.

Each decoder layer in the Transformer architecture also includes residual connections and layer normalization, which help with the training stability and performance.

By stacking multiple decoder layers on top of each other, the Transformer model is able to generate increasingly accurate and fluent output sequences. During training, the decoder is usually fed with the ground-truth output sequence as input, but during inference, it generates the output sequence one token at a time using the previously generated tokens as context. We used num_layers value of 4 for the Decoder.

3.2.8 Training

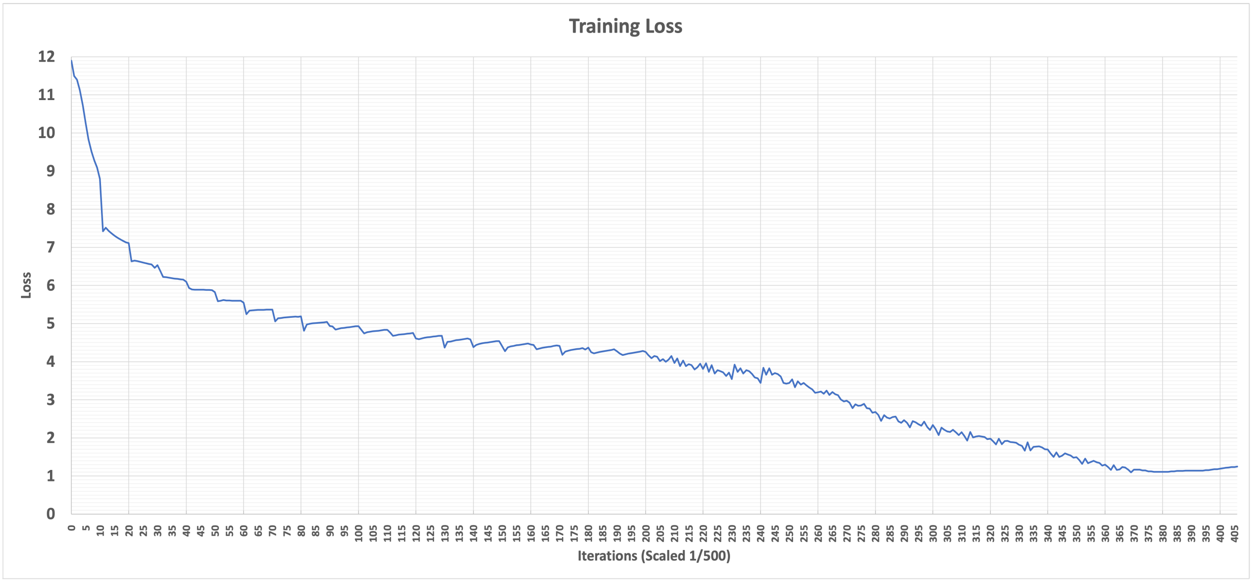


Figure 3.12: Training Loss for adaptive learning rate

Various learning rates were used at the beginning of training process, But with fixed training rates of 0.1, 0.01 and 0.001 the Sparse Categorical Cross-entropy loss was fluctuating at constant value after few epochs. Then, We used the Scheduled Adaptive learning rate which adjust as the training process goes on which yeilds us a better model loss value.Each epoch took us around 500 seconds.

Adam optimizer

Adam (Adaptive Moment Estimation) is an optimization algorithm commonly used for training deep learning models. It is a gradient-based optimization algorithm that calculates an adaptive learning rate for each parameter based on the estimation of the first and second moments of the gradients.

In simpler terms, Adam optimizer adjusts the learning rate for each weight in the neural network during training, based on the average of the magnitude of the gradients for that

weight and the rate of change of those gradients. The adaptive learning rate ensures that the parameters are updated in a way that avoids overshooting or undershooting the optimal values.
We initialized our optimizer with beta1 and beta2 values of 0.9 and 0.98 and epsilon value for the stability of $\mathbf{10^{-9}}$.
We tried different learning rates at the beginning but the loss was not decreasing after certain epoches so we used the costum schedule apporach to change learning rate as the step increases which made the loss to decrease steadily.

Defining Losses

Sparse Categorical Crossentropy is a loss function that is commonly used in multiclass classification problems where the classes are mutually exclusive (i.e., an input can belong to only one class). It measures the dissimilarity between the true labels and the predicted probabilities for each class, with the goal of minimizing this dissimilarity during training.
The "Categorical" part of the name refers to the fact that the loss is calculated using a categorical distribution, which is a probability distribution over mutually exclusive outcomes. The "Sparse" part of the name refers to the way that the labels are encoded. In contrast to one-hot encoding, where each label is represented as a binary vector of length equal to the number of classes, in sparse categorical encoding, each label is represented as an integer value corresponding to the index of the true class.
In other words, if you have n classes, then the true label for each example will be an integer between 0 and n-1. The predicted probabilities for each class are also represented as a probability distribution over the n classes, and the loss is calculated by comparing the true labels to the predicted probabilities using the cross-entropy formula.
For the loss calculation we utilized the Sparse Categorical Crossentropy loss with parameters from_logits = True which says y_Pred is expected to be a logits tensor and reduction applied to loss to 'None'.

Masks, Checkpoints and Training Steps

The Transformer architecture uses a technique called masking to handle variable-length input sequences. During training, certain tokens in the input sequence are masked to indicate that they should not be used as input to the model. For example, in a machine translation task, the target sequence is shifted by one position relative to the input sequence, and the first token of the target sequence is masked to prevent it from being used as input. In TensorFlow, masking is usually implemented using a boolean mask that is multiplied by the

input sequence to zero out certain tokens.
Checkpoints are saved snapshots of the model during training that can be used to resume training or to evaluate the model on new data. Checkpoints typically include the weights and biases of the model, as well as the optimizer state and other training parameters. In TensorFlow, checkpoints are saved to disk in a binary format that can be restored later using the ”tf.train.Checkpoint” and ”tf.train.CheckpointManager” classes.
Training a Transformer model typically involves many iterations of feeding input sequences through the model, computing the loss, and using backpropagation to update the model parameters. Each of these iterations is called a training step. In TensorFlow, training steps are typically organized into epochs, where each epoch consists of one pass through the training data. During training, the model is optimized using an optimizer such as Adam or Stochastic Gradient Descent (SGD), and the training progress is typically monitored using metrics such as the loss and accuracy on a validation set.
During the training process we saved the checkpoints using the tensorflow checkpoint manager after 5 epochs and the overall training is done in total of 40 epochs.

3.2.9 Inference

Inference in transformers refers to the process of using a trained transformer model to make predictions or generate outputs for new input sequences that the model has not seen during training.
During inference, the input sequence is fed into the transformer model, which processes it through its encoder and decoder components to generate an output sequence. In the case of language modeling, for example, the output sequence could be a sequence of words or characters that correspond to a translated version of the input sequence.
One important aspect of inference in transformers is that the decoder generates the output sequence one token at a time, using the previously generated tokens as context. This allows the model to generate sequences that are coherent and fluent, since each token is generated based on the context provided by the preceding tokens.
Another important consideration during inference is the use of beam search, which is a heuristic algorithm for exploring the space of possible output sequences. Beam search considers multiple candidate sequences in parallel, and selects the most likely sequence based on a combination of the probability scores assigned by the transformer model and a beam width parameter that controls the number of candidate sequences that are considered at each step.
Overall, inference in transformers is a critical component of natural language processing applications.
We implemented a evaluate() function to get the next output word from our model and we

supply the input text to summarize() function which calls the evaluate() function to get the final output text from our transformers.

3.3 Evaluation

Evaluating the effectiveness of summaries is a prominent topic in the field of natural language processing. A number of methods can be used to perform the evaluation, including having human evaluators and using precision/recall measurements.

3.3.1 BLEU Score

In natural language processing, the BLEU (Bilingual Evaluation Understudy) score is also used as an evaluation metric for summarization tasks. It measures the similarity between the machine-generated summary and one or more human-generated summaries of the same text.

To calculate the BLEU score for a summary, the summary is first segmented into n-grams (contiguous sequences of n words). Then, the n-grams of the machine-generated summary are compared to the n-grams of the human-generated summaries to determine the degree of overlap.

The BLEU score ranges from 0 to 1, with higher scores indicating greater similarity between the machine-generated summary and the human-generated summaries. Typically, the BLEU score is calculated for n-grams of different lengths (such as unigrams, bigrams, trigrams, and so on) and then a weighted average is taken to obtain the final score.

$$\text{BLEU} = \text{BP} \cdot \exp\left(\sum_{n=1}^{N} w_n \log p_n\right) \tag{3.6}$$

where:

BP is the brevity penalty factor N is the maximum n-gram order w_n is the weight assigned to the n-gram order p_n is the precision of the n-gram matches, defined as the number of n-grams in the generated text that match at least one of the reference texts, divided by the total number of n-grams in the generated text. The brevity penalty factor is defined as:

$$\text{BP} = \begin{cases} 1 & \text{if } c \geq r \ \ \exp\left(1 - \frac{r}{c}\right) \\ \text{if } c < r \end{cases} \tag{3.7}$$

where c is the length of the generated text and r is the effective reference length, which is the closest reference length to the generated text length.

Note that the weights assigned to each n-gram order can be set to any desired value, but are typically set to $1/N$.

The steps we followed to calculate the BLEU score between a text and its summary

1. Collect a set of human-generated summaries of the text.

2. Preprocess the text and summary, which may include steps such as tokenization, stemming, and stop word removal.

3. Choose the n-gram length to use for the calculation (usually 1-4).

4. Generate n-grams for both the machine-generated summary and the human-generated summaries.

5. Calculate the precision of each n-gram in the machine-generated summary by counting how many times it appears in the human-generated summaries.

6. Calculate the brevity penalty, which penalizes overly short machine-generated summaries, by comparing the length of the machine-generated summary to the length of the shortest human-generated summary.

7. Combine the precision scores with the brevity penalty to obtain a final score for the machine-generated summary.

8. Repeat the process for each machine-generated summary and compute the average BLEU score across all of them.

Python script used to calculate BLEU score

```
import nltk
from nltk.translate.bleu_score import sentence_bleu

# Preprocess the document and summary (tokenization and lowercasing)
document_tokens = nltk.word_tokenize(document.lower())
summary_tokens = nltk.word_tokenize(summary.lower())

# Calculate BLEU score for 1-gram and 2-gram precision
bleu_1gram = sentence_bleu([document_tokens], summary_tokens, weights=(1, 0, 0, 0))
bleu_2gram = sentence_bleu([document_tokens], summary_tokens, weights=(0.5, 0.5, 0, 0))
```

3.3.2 ROUGE Evaluation

ROUGE (Recall-Oriented Understudy for Gisting Evaluation) is a widely used evaluation metric for automatic summarization in natural language processing. ROUGE measures the overlap between the automatic summary and a reference or gold-standard summary, typically generated by a human annotator. The overlap is computed as the number of common n-grams (word sequences) between the two summaries, divided by the total number of n-grams in the reference summary. The n-gram length is specified by the user, but the most common values are unigrams (single words) and bigrams (word pairs). The ROUGE score ranges from 0 to 1, with a score of 1 indicating a perfect match between the automatic and reference summaries. ROUGE is widely used because it is simple to compute and provides a quick and intuitive evaluation of the quality of the automatic summary.

The formula for the ROUGE (Recall-Oriented Understudy for Gisting Evaluation) score between an automatic summary S and a reference summary R can be expressed as:

$$\text{ROUGE} = \frac{\sum_{n=1}^{N} \text{count}(n\text{-gram}(S) \cap n\text{-gram}(R))}{\sum_{n=1}^{N} \text{count}(n\text{-gram}(R))} \tag{3.8}$$

where n-gram(S) and n-gram(R) are the sets of n-grams (word sequences of length n) in the automatic and reference summaries, respectively, and N is the maximum length of the n-grams.

The numerator of the formula computes the count of common n-grams between the two summaries, while the denominator computes the total number of n-grams in the reference summary. The resulting ROUGE score is a value between 0 and 1, with a score of 1 indicating a perfect match between the automatic and reference summaries. The most common values for n are 1 for unigrams (single words) and 2 for bigrams (word pairs).

In ROUGE, "r" (recall), "p" (precision), and "f" (f-score) are the three evaluation measures that are commonly reported.

- **Recall (r)** measures the proportion of relevant information in the reference summary that is present in the generated summary.
- **Precision (p)** measures the proportion of relevant information in the generated summary that is present in the reference summary.
- **F-score (f)** is the harmonic mean of precision and recall and provides a single score that balances both measures.

The steps that can be used to compare a document and its summary using ROUGE are:

1. **Pre-processing**: Clean and pre-process the document and its summary to remove any unwanted characters, symbols, or stop words. This step is important for ensuring that the ROUGE scores are meaningful and accurately reflect the quality of the summary.
2. **Tokenization**: Tokenize the document and its summary into individual words or phrases, depending on the desired level of granularity.
3. **Calculate n-grams**: Calculate the n-grams (sequences of n words) in both the document and its summary. The most commonly used n-grams are unigrams (single words) and bigrams (pairs of words).
4. **Compute recall**: Compute the recall score, which is the number of n-grams in the summary that appear in the document, divided by the total number of n-grams in the document.
5. **Compute precision**: Compute the precision score, which is the number of n-grams in the summary that appear in the document, divided by the total number of n-grams in the summary.
6. **Compute ROUGE-n score**: Compute the ROUGE-n score, which is a weighted harmonic mean of recall and precision scores. The weighting factor determines the relative importance of recall and precision in the ROUGE-n score.
7. **Repeat for different n-grams**: Repeat the above steps for different values of n to calculate the ROUGE-n scores for different levels of granularity.

8. **Interpret the results**: The ROUGE scores can be interpreted to determine the quality of the summary. A high ROUGE score indicates that the summary is a good representation of the document, while a low ROUGE score indicates that the summary is not a good representation of the document.

Python script used to calculate ROUGE Score

```
from rouge import Rouge

# Define the document and summary text
document = ‘‘This is a long document with some important information.‘‘
summary = ‘‘This document contains important information.‘‘

# Create a Rouge object
rouge = Rouge()

# Calculate the ROUGE scores
scores = rouge.get_scores(summary, document)
```

Rouge-1 Score

ROUGE-1 score measures the unigram (word-level) recall between a candidate summary and a reference summary. In other words, it calculates the number of overlapping words between the candidate summary and the reference summary, divided by the number of words in the reference summary. The score ranges from 0 to 1, with a higher score indicating a higher level of overlap and a more accurate summary.
ROUGE-1 is commonly used for evaluating the quality of automatic summarization models, as it provides a simple and intuitive measure of the amount of overlap between the candidate summary and the reference summary. Additionally, ROUGE-1 can be used to compare the performance of different summarization models and to evaluate the improvement of models over time.

Rouge 2 Score

ROUGE (Recall-Oriented Understudy for Gisting Evaluation) is a suite of evaluation metrics for automatic summarization and machine translation. ROUGE-2 score is one of the metrics in the ROUGE suite.
ROUGE-2 score measures the bigram recall between a candidate summary and a reference summary. In other words, it calculates the number of overlapping bigrams (pairs of consecutive words) between the candidate summary and the reference summary, divided by the number of bigrams in the reference summary. The score ranges from 0 to 1, with a higher score indicating a higher level of overlap and a more accurate summary.

ROUGE-2 is used in conjunction with ROUGE-1 to provide a more comprehensive evaluation of the quality of automatic summarization models. While ROUGE-1 measures the word-level overlap between the candidate summary and the reference summary, ROUGE-2 provides a measure of the phrasing and coherence of the candidate summary.
ROUGE-2 is commonly used in natural language processing (NLP) and information retrieval (IR) to evaluate the quality of automatic summarization models, as well as machine translation models, dialogue systems, and other text generation models.

3.4 Significant word detection

Significant word detection, also known as keyword extraction, is the process of automatically identifying important or meaningful words and phrases from a given text. This task is essential in natural language processing and text analysis, as it allows researchers, businesses, and other organizations to quickly understand the main topics, themes, and sentiments present in large amounts of textual data.

There are various methods for detecting significant words from text, including statistical techniques such as TF-IDF (Term Frequency-Inverse Document Frequency), which measures the importance of a word in a document by considering its frequency in that document and inversely proportional to its frequency in the whole corpus of documents. Other methods include using machine learning algorithms like Naive Bayes or Neural Networks.

Significant word detection has numerous applications, including search engine optimization, social media analysis, sentiment analysis, and automated text summarization. By identifying the most important keywords and phrases in a piece of text, businesses and researchers can gain valuable insights into consumer behavior, public opinion, and emerging trends.

3.4.1 Identifying the key words and phrases

We use LSA (Latent Semantic Analysis) for identifying the most relevant terms in a given document or corpus. It achieves this by analyzing the relationships between the terms and their co-occurrence patterns, allowing it to identify the underlying semantic structure in the text. The process of identifying key words with LSA involves preprocessing the text, converting it into a term-document matrix, applying SVD to reduce its dimensionality, and using the resulting matrix to identify the most important terms based on their association with the underlying semantic structure. The key words identified by LSA can be used to summarize the main topics or themes in the text, making it useful for information retrieval and document categorization tasks. By leveraging LSA's ability to identify latent semantic structure, it can help identify the most relevant and important terms in a document or corpus, allowing for more effective analysis and understanding of the text.

3.4.2 Heading Generation

We use various modules from the Natural Language Toolkit (NLTK) library such as stop words, wordtokenize , WordNetLemmatizer, and heapq to perform text processing tasks like tokenization, lemmatization, stopword removal, and word frequency counting. The function first preprocesses the article text by converting it to lowercase, tokenizing it into individual words, and removing stop words and punctuations. Then it calculates the frequency of each word in the preprocessed article text,then each word is frequency normalized and selects the two most frequent words as the basis for the header. The header is returned as a list of the two most frequent words joined together.

3.5 Software Development Approach

We intend to develop the project as a web application GUI (Graphical User Interface) based on REACT that incorporates different NLP techniques, written in PYTHON to achieve the entire desired tasks. Basically, the web application will run the python script in the local host in order to trigger the ML model.

The overall software development process can be stated as:

1. **Data Collection:** Data collection will be done through a survey and some data could be extracted from open source dataset collection like Kaggle, Watson studio, IBM dataset, etc.

2. **Training and testing:** The best suitable ML algorithm will be chosen through diffident experiments/trials and the obtained data is trained and tested for model development

3. **Model Development:** Developing ML model based on train/test result

4. **GUI Development:** Graphical user interface development as a desktop application

5. **Integrating models and the GUI:** Integration of the developed model with the web application to provide an attractive interface

4. System Design

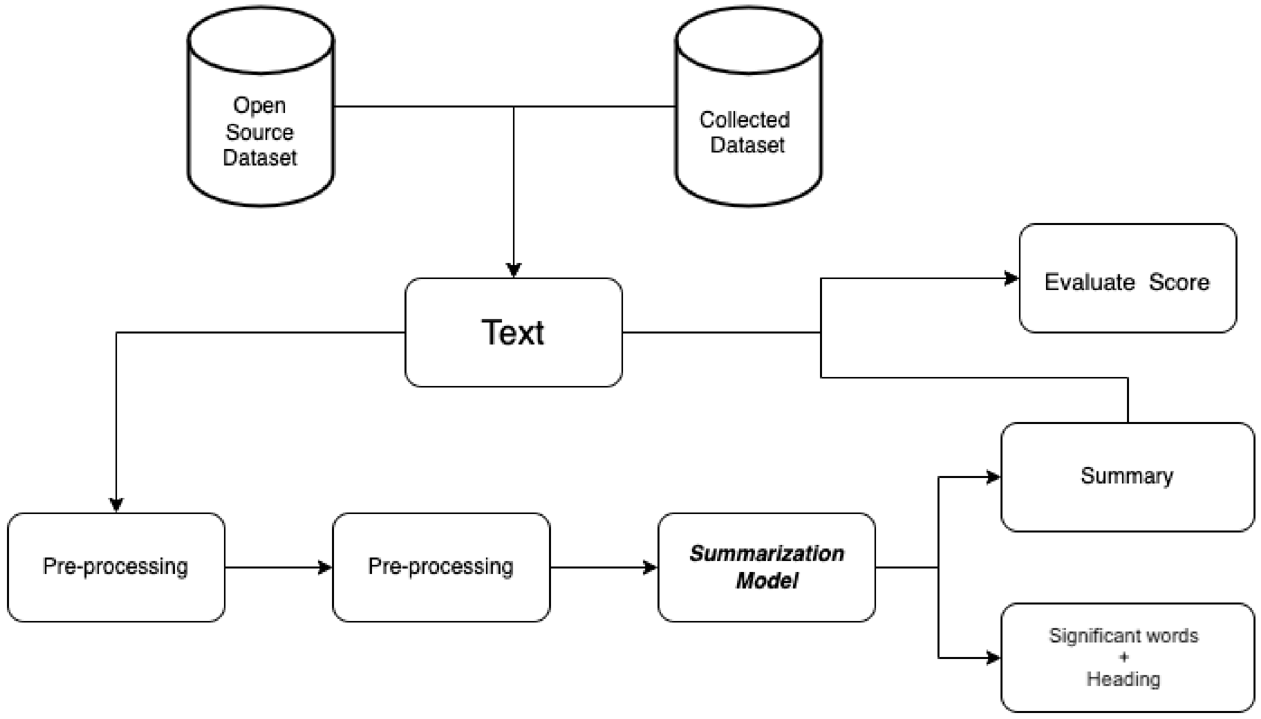


Figure 4.1: Overall System Design

At first, we collect text and preprocess it for the removal of punctuations and stop words. We tokenize the text and extract features from the text such as frequency of words, Entropy, length of sentences, the position of sentences, proper nouns, sentence-to-sentence similarities, etc. Then our model summarizes the text and generates the output with a title and summary. For numeric data, our model outputs it in a tabular format. Then the output is fed into the Evaluation Model and Feedback is observed which is provided to our model and enhances our model.

4.1 System Elements

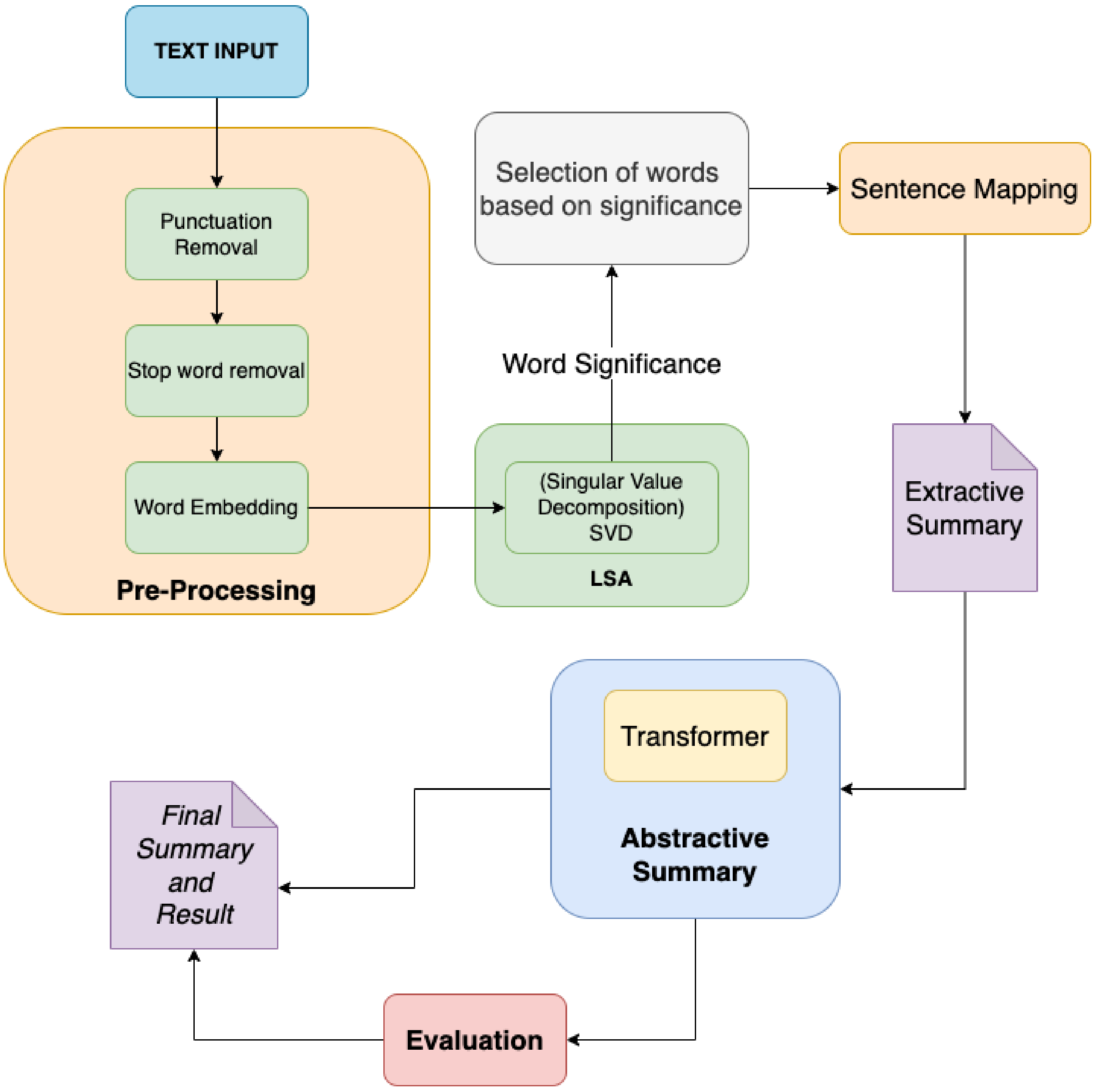


Figure 4.2: Full Block Diagram

4.1.1 Pre-processing

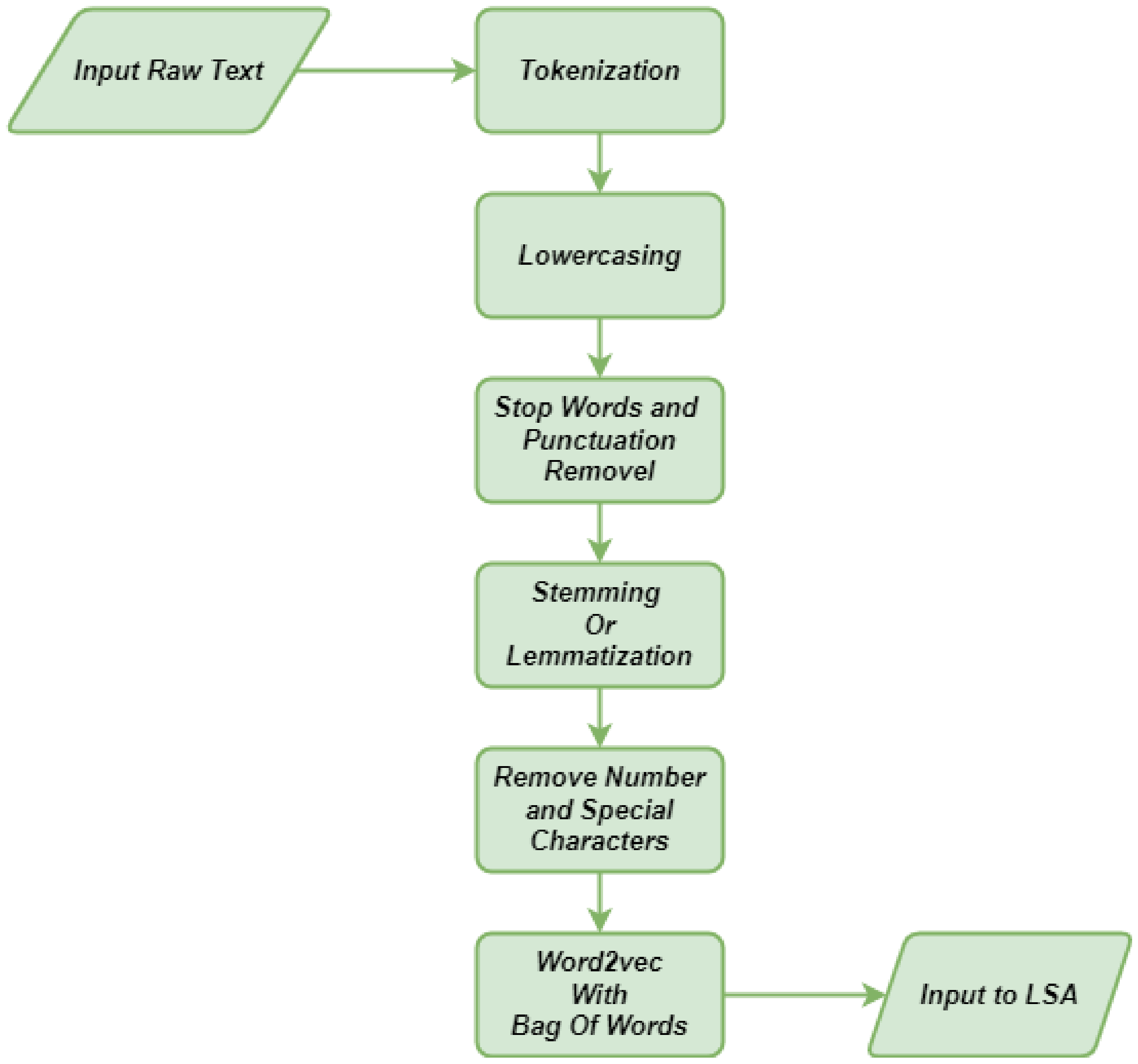

Input Raw Text
Tokenization
Lowercasing
Stop Words and Punctuation Removel
Stemming Or Lemmatization
Remove Number and Special Characters
Word2vec With Bag Of Words
Input to LSA

Figure 4.3: Preprocessing Block Diagram

Before passing the data to the LSA Block we perform the following data processing on the raw input data:

- At first the input data is tokenized by splitting the raw text into individual words.
- Each token is converted into lowercase to standardize the data and reduce the size of vocabulary.
- Stop words and punctuation which does not carry much significance are removed

- Each token is converted into its corresponding lemma ie. root form which reduce the size of vocabulary.
- Numbers and special characters are removed.
- Each token is vectorize using the bag of words.
- Finally the data is subjected to LSA Processing.

4.1.2 Latent Semantic analysis

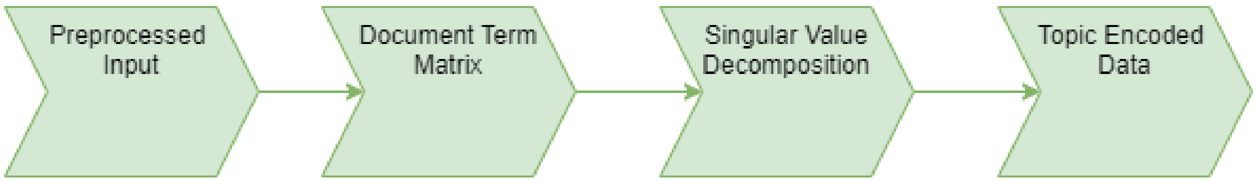


Figure 4.4: LSA Block Diagram

After pre-processessing, a term-document matrix is created, which represent the frequency of therms in each documents. LSA uses Singular Value Decomposition(SVD) to reduce the dimensionality of term document matrix and capture the latent semantic structure of the text. The Latent semantic analysis (LSA) gives the weighted of document belonging to different topics. Summarization is done by selecting the top N documents from each topic depending on weighted.
This gives summarization where we get the higher weighted document form each of the topics for the summary.

4.1.3 Sentence Mapping

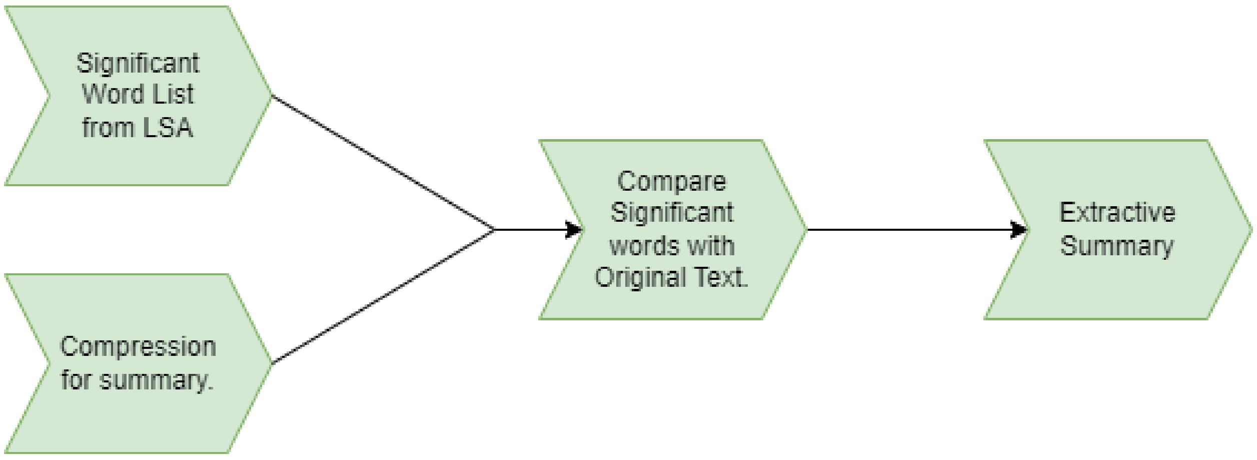

Significant Word List from LSA
Compression for summary.
Compare Significant words with Original Text.
Extractive Summary

Figure 4.5: Sentence Mapping Block Diagram

After LSA , We get a list of significant words according to the correlation of those words. We ask for the compression ratio for the summary and take those two and compare the words with the original text which yeilds us the final extractive summary. Required words are selected according to the compression ratio and the final extraction is done after mapping those words with the original article.

4.1.4 Transformer

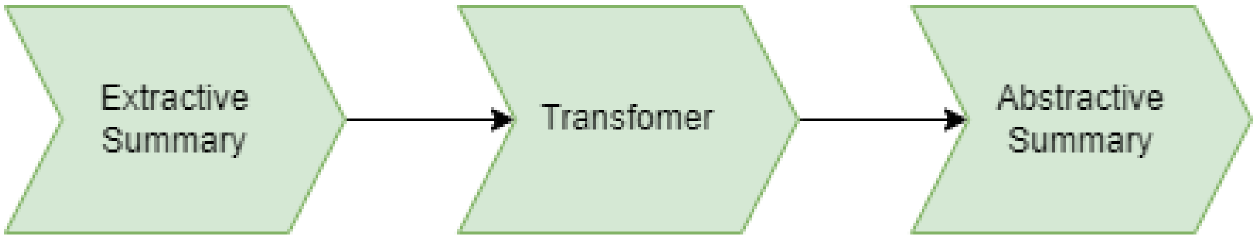


Figure 4.6: Transformers Block Diagram

After we obtain extractive summary, We supply it to the transformer and the greedy decoding algorithm with the model and the extractive summary implements tokenizer and detokenizers to give us the final abstractive summary.

5. Results and Discussion

For the evaluation of our model, we have used 50 texts on random topics taken from different internet sources and the reference summary is generated using ChatGPT.

5.1 Model Evaluation

The individual BLEU and ROUGE scores are mentioned below.

5.1.1 BLEU

We have calculated the BLEU score using 1-gram and 2-gram, comparing the summary generated by our model with the original text and with the reference summary and the obtained results are tabulated below.

With the original text

Summary Type	Average BLEU-1 Score
Small	0.1257292
Medium	0.1352912
Large	0.1558831
Abstractive	0.1032881

Summary Type	Average BLEU-2 Score
Small	0.06140072
Medium	0.08256357
Large	0.09292331
Abstractive	0.07532681

Table 5.1: Average BLEU Score With Original text

With the reference text

Summary Type	Average BLEU-1 Score
Small	0.1032424
Medium	0.1164532
Large	0.12413142
Abstractive	0.1412331

Summary Type	Average BLEU-2 Score
Small	0.1140072
Medium	0.0956359
Large	0.1097331
Abstractive	0.1053241

Table 5.2: Average BLEU Score With Reference Summary

This table above presents the results of evaluating machine-generated summaries using the BLEU score metric.The first subtable shows the average BLEU-1 and BLEU-2 scores obtained when comparing machine-generated summaries with the original text. Overall, theresults indicate that the machine-generated summaries have higher BLEU scores when compared to the original text than when compared to human-generated reference summaries. Additionally, the BLEU-1 scores are generally higher than the BLEU-2 scores for all types of summaries.

5.1.2 ROUGE Evaluation

For small length summary

Rouge-1 Type	Average Score
r	0.547341347
p	0.413546623
f	0.684176683

Table 5.3: Average Rouge-1 score of small length summary with reference summary

Rouge-1 Score Based on the values, it appears that the summarization algorithm has a higher recall than precision, meaning that it tends to include more unigrams from the reference summaries than necessary, but may miss some important content. The F-measure score of 0.684 suggests that overall, the summarization algorithm is performing reasonably well in terms of capturing the important content of the reference summaries.

Rouge-2 Type	Average Score
r	0.219019805
p	0.1527248
f	0.168539058

Table 5.4: Average Rouge-2 score of small length summary with reference summary

Rouge-2 Score These scores indicate the performance of the summarization algorithm in terms of its ability to capture the important bigram content of the reference summaries. The scores are relatively low, suggesting that the summarization algorithm may not be performing well in terms of capturing the important bigram content.

Rouge-l score These scores indicate the level of agreement between the generated summary and the reference text, with higher scores indicating better agreement.
The recall score indicates how much of the reference text was captured in the summary, while

Rouge-l Type	Average Score
r	0.40906756
p	0.3076767
f	0.334966801

Table 5.5: Average Rouge-l score of small length summary with reference summary

the precision score indicates how much of the summary is relevant to the reference text. The F1-score is a harmonic mean of the recall and precision scores, giving equal weight to both metrics.
Overall, these scores suggest that the generated summary has captured around 40% of the information in the reference text, while including around 30% of relevant information. The F1-score, which takes both recall and precision into account, is around 33%, indicating that there is room for improvement in terms of the quality of the generated summary.

For medium length summary

Rouge-1 Type	Average Score
r	0.465829322
p	0.413546623
f	0.356628296

Table 5.6: Average Rouge-1 score of medium length summary with reference summary

Rouge-1 Score The table shows the average Rouge-1 score for a medium length summary compared to a reference summary. Rouge-1 is a measure of the overlap between the summary and the reference summary based on unigrams. The average Rouge-1 recall score is 0.4658, meaning that the summary contains 46.58% of the unigrams that are present in the reference summary. The average Rouge-1 precision score is 0.4135, indicating that 41.35% of the unigrams in the summary are also present in the reference summary. The average Rouge-1 F1 score, which is the harmonic mean of recall and precision, is 0.3566.

Rouge-2 Score The table shows the average scores for Rouge-2 evaluation metric for a medium-length summary compared to a reference summary. The Rouge-2 metric measures the overlap of bigrams (two consecutive words) between the two summaries. The "r" score of 0.2348 indicates that 23.48% of the bigrams in the reference summary are also present in the generated summary. The "p" score of 0.1411 indicates that 14.11% of the bigrams in the generated summary are also present in the reference summary. The "f" score of 0.1615 is the

Rouge-2 Type	Average Score
r	0.234789521
p	0.141130966
f	0.1614725

Table 5.7: Average Rouge-2 score of medium length summary with reference summary

harmonic mean of the ”r” and ”p” scores, indicating the overall similarity between the two summaries. Overall, these scores indicate that there is some overlap between the generated summary and the reference summary

Rouge-l Type	Average Score
r	0.437873077
p	0.330837299
f	0.359466944

Table 5.8: Average Rouge-l score of medium length summary with reference summary

Rouge-l score This table shows the average Rouge-l score of a medium-length summary compared to a reference summary. The Rouge-l score is a metric used to evaluate the quality of a summary by comparing it to a reference summary. The average Rouge-l score is 0.437, which means that the summary has captured 43.7% of the important information from the reference summary. The precision score (p) is 0.330, which means that 33.0% of the summary is relevant to the reference summary. The F1 score (f) is 0.359, which is the harmonic mean of precision and recall, and indicates an overall balance between precision and recall.

For large length summary

Rouge-1 Type	Average Score
r	0.583919428
p	0.291044855
f	0.373552029

Table 5.9: Average Rouge-1 score of large length summary with reference summary

Rouge-1 Score The table shows the average Rouge-1 scores for a large length summary with a reference summary. Rouge-1 evaluates the overlap between unigram (single word) sequences in the summary and reference summary. The results indicate that the recall score

(r) is higher than the precision score (p), suggesting that the summary contains more of the important information in the reference summary, but also more extraneous information. The F1 score (f) is the harmonic mean of recall and precision, and provides a balanced measure of performance. In this case, the F1 score is 0.3735, which indicates comprehensive performance.

Rouge-2 Type	Average Score
r	0.325176236
p	0.134177558
f	0.18023235

Table 5.10: Average Rouge-2 score of large length summary with reference summary

Rouge-2 Score These scores indicate the performance of the summarization system in generating a summary that captures important information from the reference summary. The Rouge-2 recall score suggests that the system is able to capture about 32.5% of the important information from the reference summary. The precision score suggests that the system generates some irrelevant content as well, as it captures only about 13.4% of the important information from the reference summary. Finally, the F1 score provides a balanced measure of both precision and recall, and it is 0.18023235 in this case.

Rouge-l Type	Average Score
r	0.553862214
p	0.273217775
f	0.352078276

Table 5.11: Average Rouge-l score of large length summary with reference summary

Rouge-l score The table reports the average scores for recall (r), precision (p), and F1-score (f). The average Rouge-l recall score is 0.553, which means that on average, the generated summary contains 55.3% of the information present in the reference summary. The average precision score is 0.273, indicating that on average, 27.3% of the information in the generated summary is also present in the reference summary. The average F1-score is 0.352, which is the harmonic mean of the recall and precision scores.

For abstractive summary

Rouge-1 Type	Average Score
r	0.455062454
p	0.352343853
f	0.38316489

Table 5.12: Average Rouge-1 score of abstractive summary with reference summary

Rouge-1 Score This table shows the average Rouge-1 score for abstractive summary with reference summary. The Rouge-1 recall score is 0.455062454, which means that the abstractive summary captures 45.5% of the important information from the reference summary. The Rouge-1 precision score is 0.352343853, which means that 35.2% of the information in the abstractive summary is relevant to the reference summary. The F1-score, which is the harmonic mean of recall and precision, is 0.38316489. This indicates the overall effectiveness of the abstractive summary in capturing important information while maintaining relevance to the reference summary.

Rouge-2 Type	Average Score
r	0.227004122
p	0.154357375
f	0.174448925

Table 5.13: Average Rouge-2 score of abstractive summary with reference summary

Rouge-2 Score The table shows the average Rouge-2 score for the abstractive summary with the reference summary. The Rouge-2 score for this summary is r=0.227004122, p=0.154357375, and f=0.174448925. These scores indicate the quality of the abstractive summary, with higher scores indicating better quality.

Rouge-l Type	Average Score
r	0.422394129
p	0.324339566
f	0.353952306

Table 5.14: Average Rouge-l score of abstractive summary with reference summary

Rouge-l score The average Rouge-l score of the abstractive summary with the reference summary is 0.422, indicating that the model's performance is moderate in terms of capturing the related information in the source text. The recall score (r) is 0.42, which means that the model has identified 42% of the important information in the source text. The precision score (p) is 0.32, indicating that the model has generated only 32% of the important information present in the reference summary. The F1 score is 0.35, which is a harmonic mean of precision and recall.

6. Conclusion

The evaluation of the text summarization model in this project revealed some interesting findings regarding the performance of the model. The Rouge scores obtained for different summary lengths indicated that the model performed best for reference summaries of medium length. This suggests that the model was most effective in summarizing texts that were not too short or too long, which is an important consideration for real-world applications where texts of varying lengths need to be summarized.

Additionally, the Rouge scores were consistently higher for reference summaries than for abstractive summaries, indicating that the model had more difficulty generating its own summaries from the original text than simply extracting important sentences. This is a common challenge in abstractive summarization, where the model needs to capture the essence of the text and convey it in its own words. The finding highlights the importance of carefully selecting the appropriate summarization technique for a given task, as well as continuing to explore ways to improve the performance of abstractive summarization techniques.

However, it is important to note that the Rouge scores provide only a limited evaluation of the summarization model, as they do not take into account the semantic coherence and readability of the generated summaries. Therefore, it is necessary to supplement the Rouge scores with other evaluation metrics that provide a more comprehensive assessment of the quality of the summaries.

Despite these limitations, the Rouge scores and BLEU scores tables provided valuable insights into the performance of the extractive and abstractive summarization techniques on the given text dataset. These metrics can help in selecting the most suitable approach for specific summarization tasks, as well as identifying areas for improvement and future research. Overall, the evaluation of the text summarization model in this project provides a solid foundation for further exploration and development of text summarization techniques.

7. Web Application

The selection of a tech stack is a crucial decision in any web application development project, as it can significantly impact the performance, scalability, and overall success of the application. In the text summarization project, Flask and React JS were chosen as the tech stack for building the web application.

Flask is a lightweight and highly customizable Python web framework that allows for rapid development of web applications. It is known for its simplicity, flexibility, and ease of use. Flask provides tools and libraries for handling web requests, managing data, and creating APIs, making it a popular choice for building RESTful web services.

React JS, on the other hand, is a widely used JavaScript library for building user interfaces. It enables the creation of highly scalable and interactive web applications with a component-based approach. React is known for its high performance and reusability, making it a popular choice for building modern web applications.

Together, Flask and React JS provide a powerful combination for building complex and modern web applications. Flask provides the backend logic for the application, while React allows for the creation of dynamic and responsive user interfaces. The two communicate with each other through APIs, allowing for a seamless user experience.

In the project, Flask was used to handle the server-side logic, including data storage and retrieval, while React was used to create a dynamic and responsive user interface for interacting with the summarization model. The Flask backend provided a RESTful API for the React frontend to consume, using the Axios library to make API calls. The React components were designed and styled using Bootstrap, while state management was handled using Redux.

The use of Flask and React JS allowed for the creation of a single-page application, where the user interface was updated dynamically without requiring a page refresh. This provided a seamless user experience, allowing users to quickly and easily input their text and view the generated summaries.

Overall, the selection of Flask and React JS as the tech stack for the text summarization project proved to be an effective choice, allowing for the creation of a high-quality web application that could handle complex functionality while providing an engaging user experience. The Flask and React JS combination provides developers with a powerful and flexible tech stack that can be used to create a wide range of modern web applications.

7.1 Application Screen-snaps

Home Screen

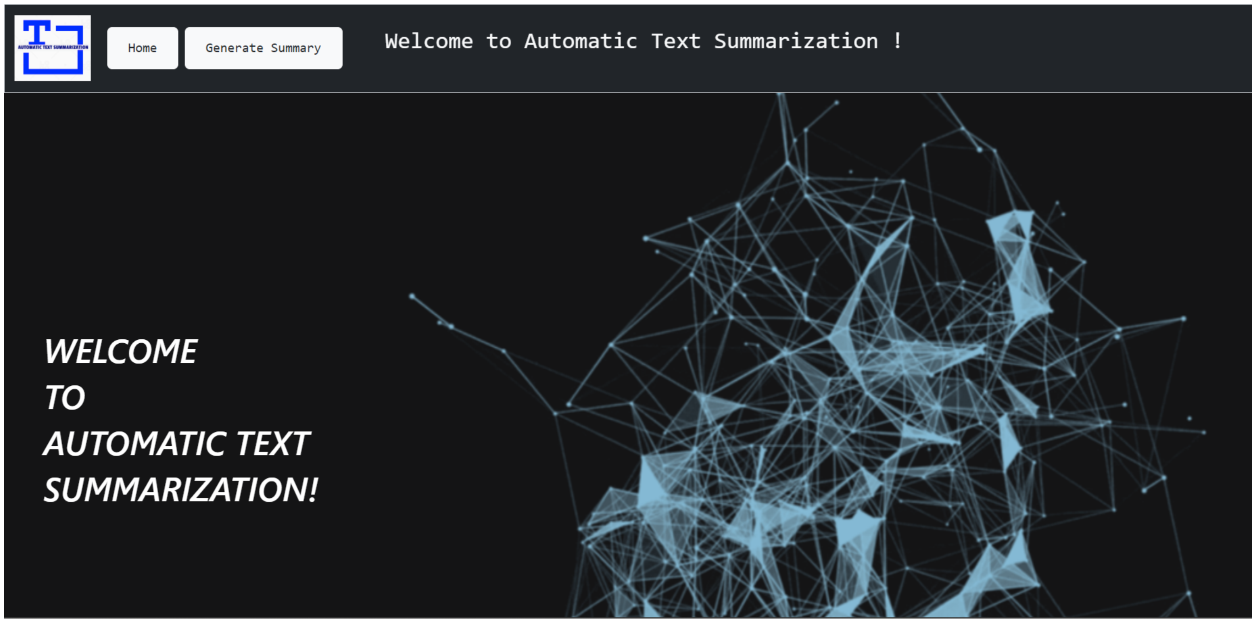
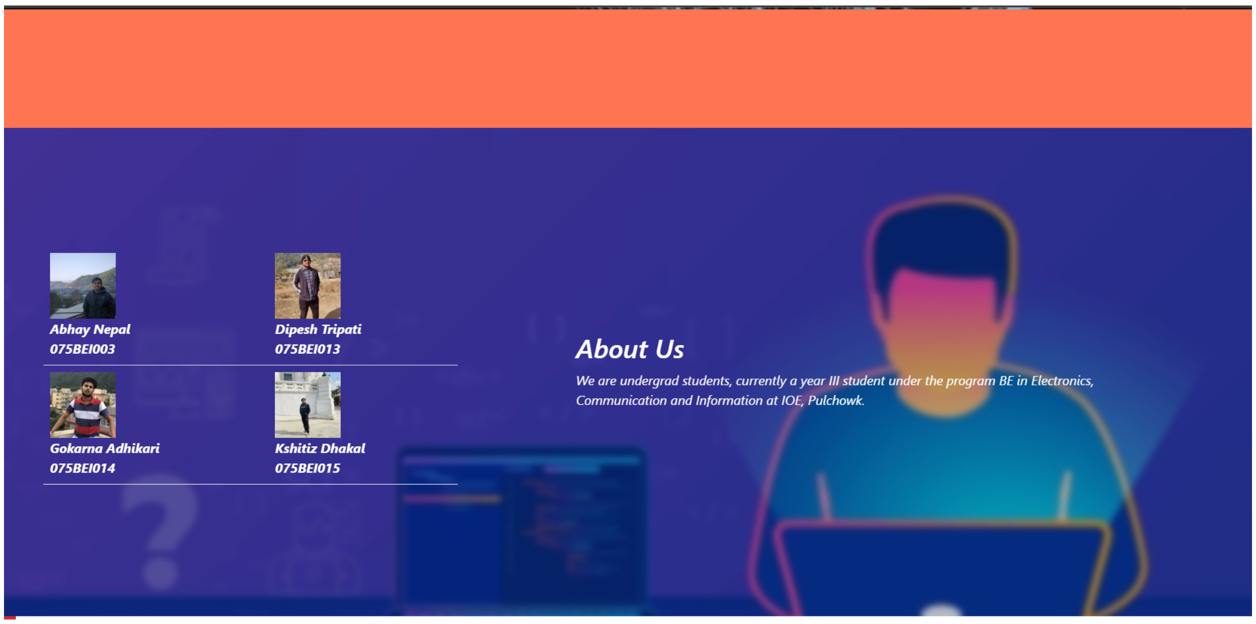
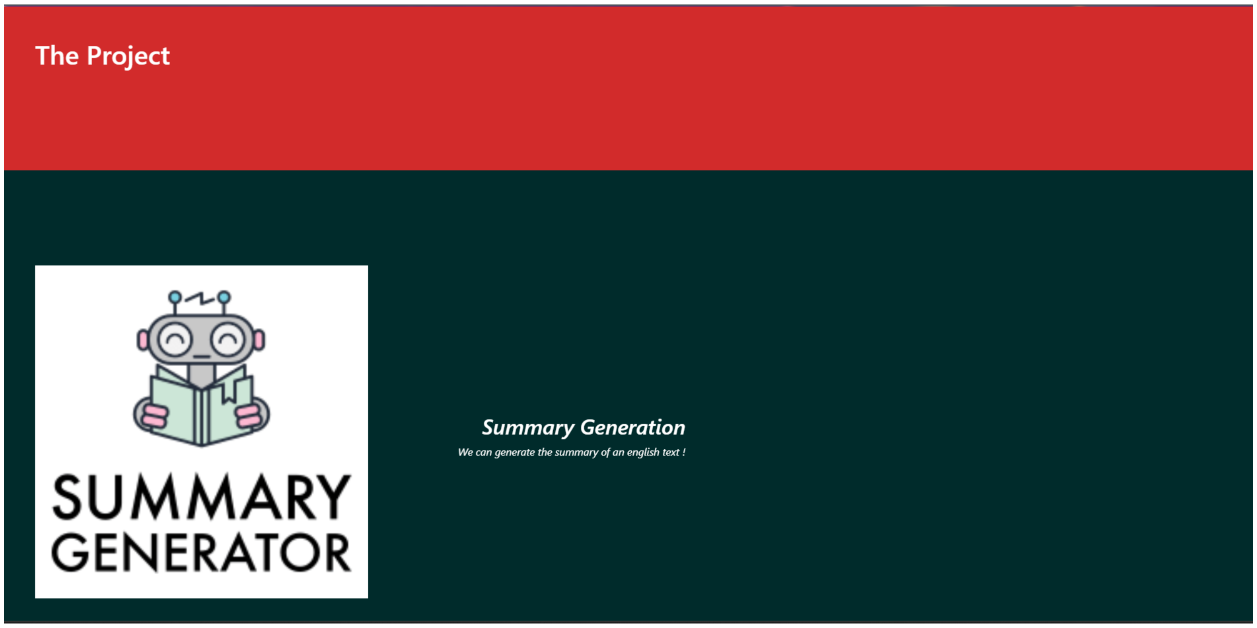


Figure 7.1: First Home Screen



We are undergrad students, currently a year III student under the program BE in Electronics, Communication and Information at IOE, Pulchowk.

Figure 7.2: Second Home Screen

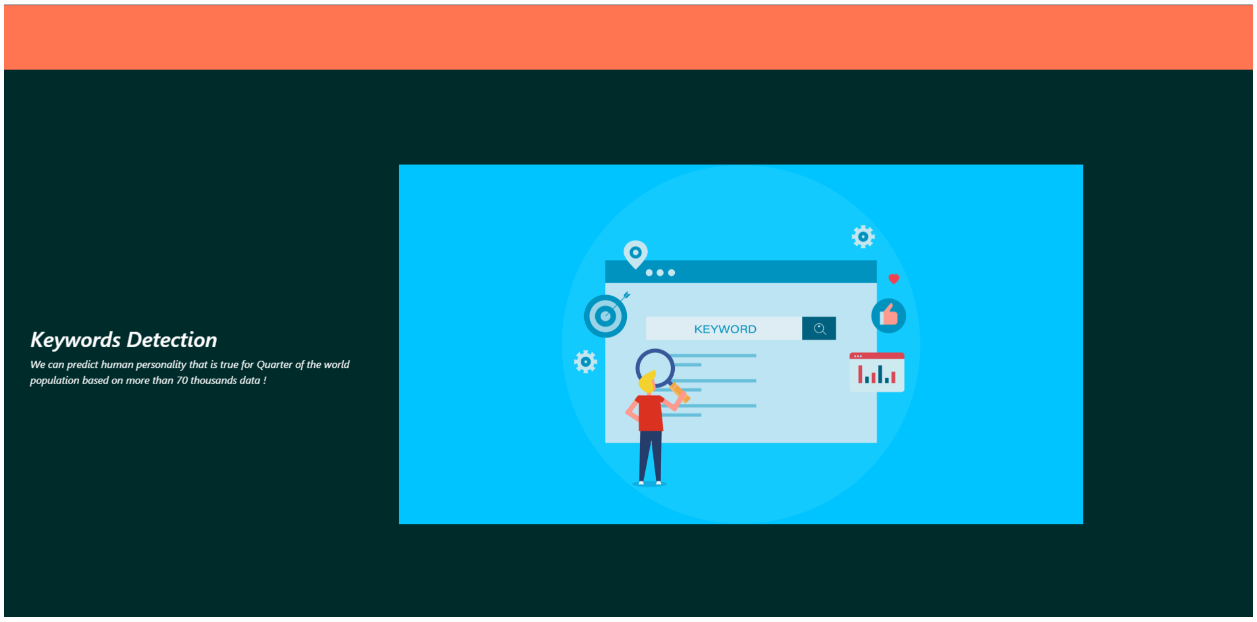


The Project

Summary Generation

We can generate the summary of an english text !

Figure 7.3: Third Home Screen



Keywords Detection

We can predict human personality that is true for Quarter of the world population based on more than 70 thousands data !

Figure 7.4: Fourth Home Screen

Summary Genertion GUI

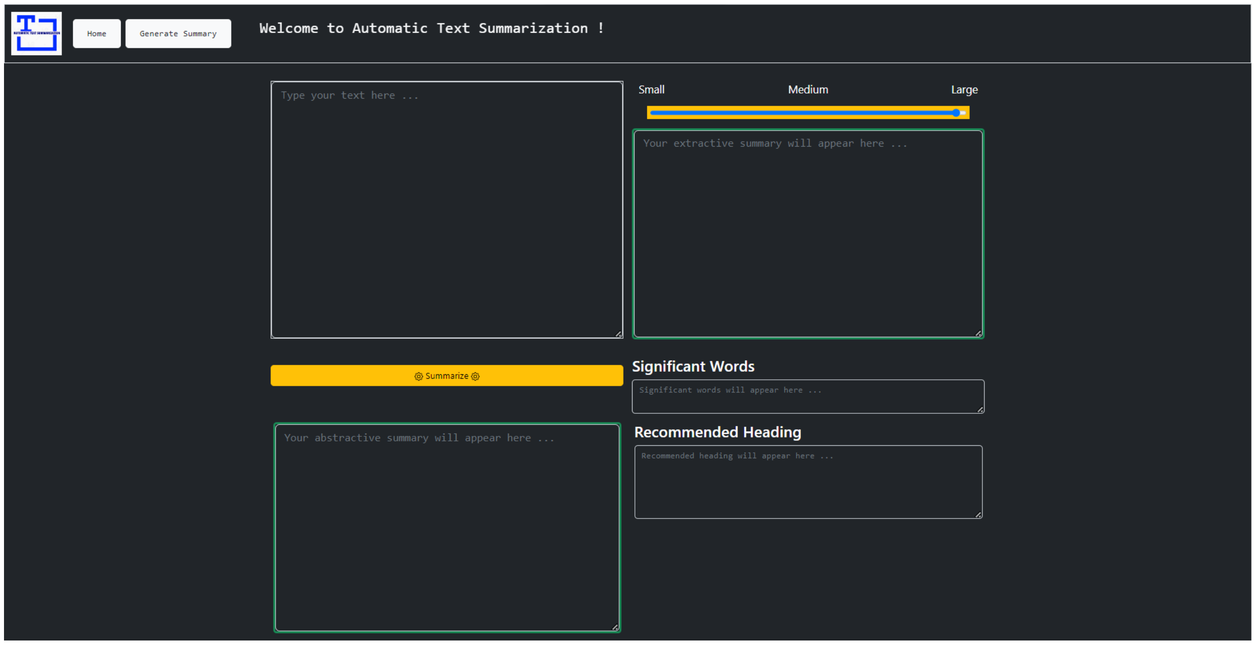
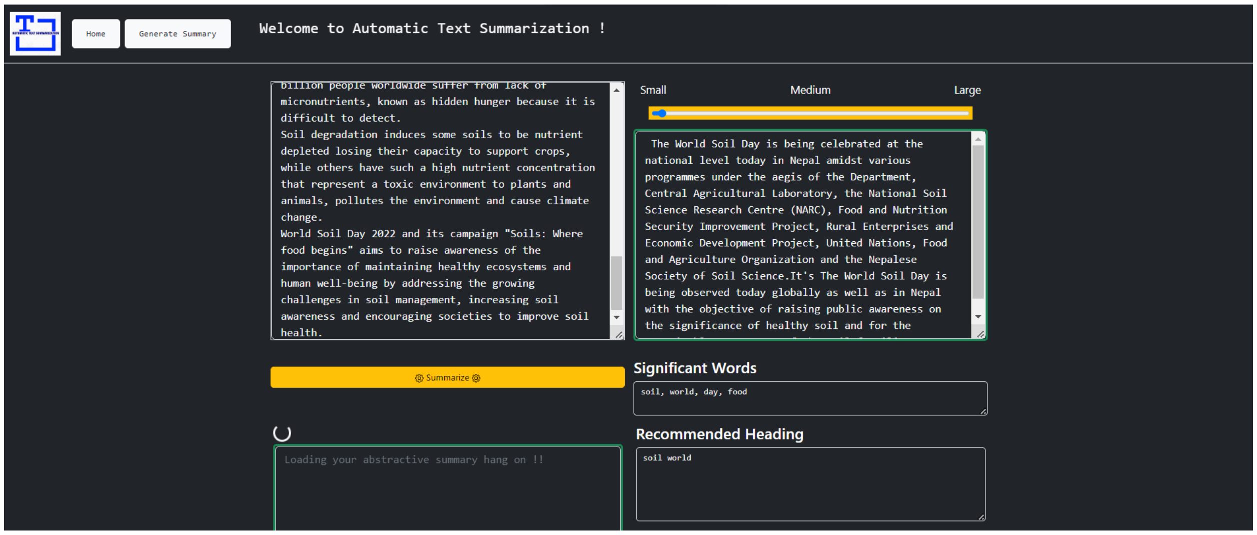

Home
Generate Summary
Welcome to Automatic Text Summarization !
Type your text here ...
Small
Medium
Large
Your extractive summary will appear here ...
Summarize
Significant Words
Significant words will appear here ...
Your abstractive summary will appear here ...
Recommended Heading
Recommended heading will appear here ...

Figure 7.5: Summary Generation GUI


Home
Generate Summary
Welcome to Automatic Text Summarization !
Small
Medium
Large
Summarize
Significant Words
Recommended Heading
Loading your abstractive summary hang on !!

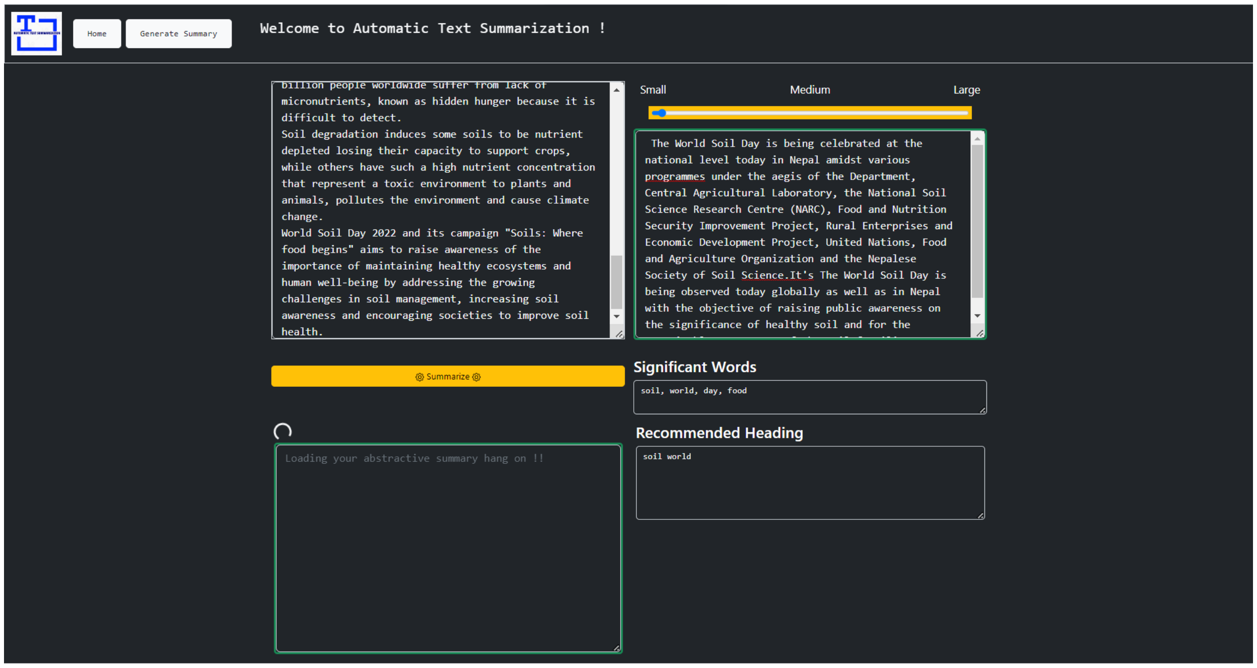
billion people worldwide suffer from lack of micronutrients, known as hidden hunger because it is difficult to detect.
Soil degradation induces some soils to be nutrient depleted losing their capacity to support crops, while others have such a high nutrient concentration that represent a toxic environment to plants and animals, pollutes the environment and cause climate change.
World Soil Day 2022 and its campaign "Soils: Where food begins" aims to raise awareness of the importance of maintaining healthy ecosystems and human well-being by addressing the growing challenges in soil management, increasing soil awareness and encouraging societies to improve soil health.

The World Soil Day is being celebrated at the national level today in Nepal amidst various programmes under the aegis of the Department, Central Agricultural Laboratory, the National Soil Science Research Centre (NARC), Food and Nutrition Security Improvement Project, Rural Enterprises and Economic Development Project, United Nations, Food and Agriculture Organization and the Nepalese Society of Soil Science.It's The World Soil Day is being observed today globally as well as in Nepal with the objective of raising public awareness on the significance of healthy soil and for the

soil, world, day, food

soil world

Figure 7.6: LSA Summary Generated in GUI



billion people worldwide suffer from lack of micronutrients, known as hidden hunger because it is difficult to detect.
Soil degradation induces some soils to be nutrient depleted losing their capacity to support crops, while others have such a high nutrient concentration that represent a toxic environment to plants and animals, pollutes the environment and cause climate change.
World Soil Day 2022 and its campaign "Soils: Where food begins" aims to raise awareness of the importance of maintaining healthy ecosystems and human well-being by addressing the growing challenges in soil management, increasing soil awareness and encouraging societies to improve soil health.

The World Soil Day is being celebrated at the national level today in Nepal amidst various programmes under the aegis of the Department, Central Agricultural Laboratory, the National Soil Science Research Centre (NARC), Food and Nutrition Security Improvement Project, Rural Enterprises and Economic Development Project, United Nations, Food and Agriculture Organization and the Nepalese Society of Soil Science.It's The World Soil Day is being observed today globally as well as in Nepal with the objective of raising public awareness on the significance of healthy soil and for the

Significant Words

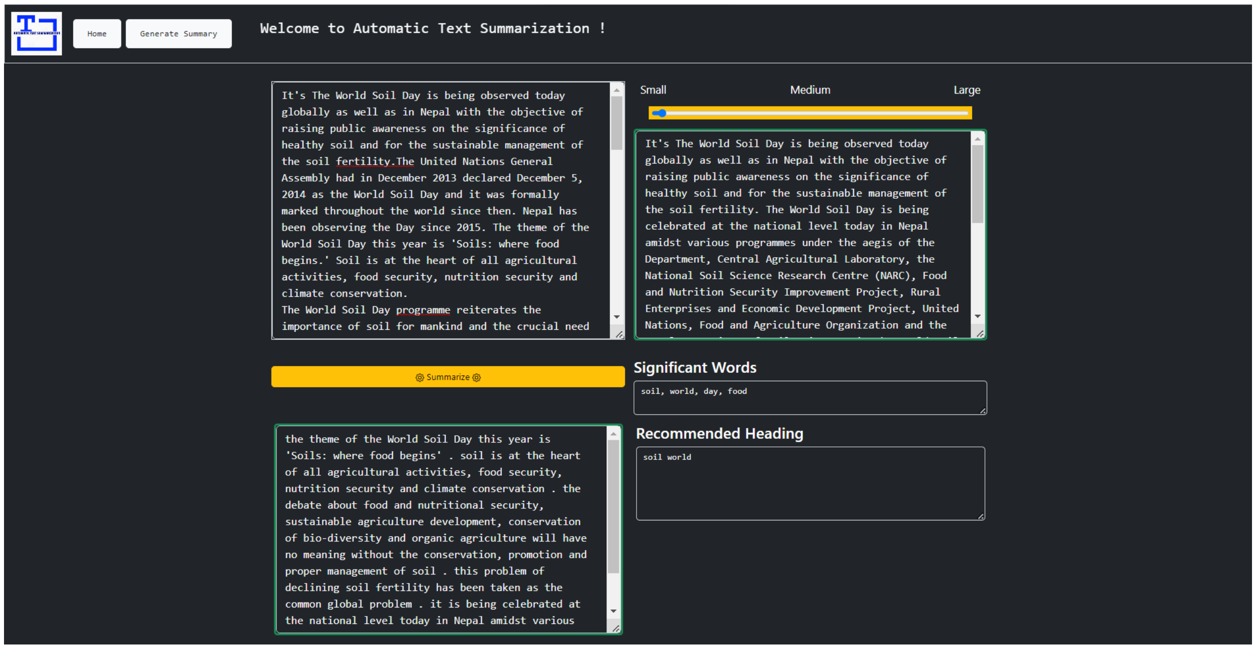
soil, world, day, food

Recommended Heading

soil world

Loading your abstractive summary hang on !!

Figure 7.7: Abstraction Summary Loading Screen



It's The World Soil Day is being observed today globally as well as in Nepal with the objective of raising public awareness on the significance of healthy soil and for the sustainable management of the soil fertility.The United Nations General Assembly had in December 2013 declared December 5, 2014 as the World Soil Day and it was formally marked throughout the world since then. Nepal has been observing the Day since 2015. The theme of the World Soil Day this year is 'Soils: where food begins.' Soil is at the heart of all agricultural activities, food security, nutrition security and climate conservation.
The World Soil Day programme reiterates the importance of soil for mankind and the crucial need

It's The World Soil Day is being observed today globally as well as in Nepal with the objective of raising public awareness on the significance of healthy soil and for the sustainable management of the soil fertility. The World Soil Day is being celebrated at the national level today in Nepal amidst various programmes under the aegis of the Department, Central Agricultural Laboratory, the National Soil Science Research Centre (NARC), Food and Nutrition Security Improvement Project, Rural Enterprises and Economic Development Project, United Nations, Food and Agriculture Organization and the

Significant Words

soil, world, day, food

Recommended Heading

soil world

the theme of the World Soil Day this year is 'Soils: where food begins' . soil is at the heart of all agricultural activities, food security, nutrition security and climate conservation . the debate about food and nutritional security, sustainable agriculture development, conservation of bio-diversity and organic agriculture will have no meaning without the conservation, promotion and proper management of soil . this problem of declining soil fertility has been taken as the common global problem . it is being celebrated at the national level today in Nepal amidst various

Figure 7.8: Abstraction Summary Finally Loaded

8. Limitations and Future Enhancement

Despite the promising results achieved by the text summarization model developed in this project, there are several limitations that should be considered when interpreting the findings.

8.1 Limitations

- The current summarization model may not work well for all types of text, such as highly technical or specialized content.
- The evaluation metrics used to assess the quality of the summaries may not provide a complete picture of their usefulness and effectiveness in real-world scenarios.

8.2 Future Enhancements

- The model can be improved by incorporating more advanced transformer models and experimenting with different text representation techniques.
- The project can be enhanced by developing a more interactive approach that incorporates user feedback to produce more personalized summaries.
- The development of a more comprehensive evaluation framework can help provide a more nuanced assessment of the quality of summaries.
- The project can be expanded to handle different types of text such as scientific papers or social media posts by using specialized approaches.
- Exploring the potential of incorporating more diverse data sources, such as images or videos, to produce more informative summaries.

References

[1] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NIPS)*, 2017.

[2] N Moratanch and S Chitrakala. A survey on extractive text summarization. In *2017 international conference on computer, communication and signal processing (ICCCSP)*, pages 1–6. IEEE, 2017.

[3] Jaime Carbonell and Jade Goldstein. The use of mmr, diversity-based reranking for reordering documents and producing summaries. In *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 335–336, 1998.

[4] Günes Erkan and Dragomir R Radev. Lexrank: Graph-based lexical centrality as salience in text summarization. *Journal of artificial intelligence research*, 22:457–479, 2004.

[5] Ramesh Nallapati, Bowen Zhou, Caglar Gulcehre, Bing Xiang, et al. Abstractive text summarization using sequence-to-sequence rnns and beyond. *arXiv preprint arXiv:1602.06023*, 2016.

[6] Luyang Huang, Marshall White, and BS Student. Natural language processing for understanding novel text summarization and media bias.

[7] Chenliang Li, Weiran Xu, Si Li, and Sheng Gao. Guiding generation for abstractive text summarization based on key information guide network. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 55–60, 2018.

[8] Romain Paulus, Caiming Xiong, and Richard Socher. A deep reinforced model for abstractive summarization. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 308–317, 2017.

[9] Zichao Liu, Zhaochun Ren, Lifeng Shang, Wenhao Li, Jiawei Han, and Jingjing Liu. Generating high-quality and informative conversation responses with sequence-to-sequence models. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1653–1662, 2018.

[10] Dragomir R Radev, Kathleen R McKeown, Eduard Hovy, Hongyan Jing, Marek Stys, Daniel Tam, and Lucy Vanderwende. Introduction to the special issue on summarization. In *Proceedings of the ACL-02 workshop on automatic summarization*, pages 1–8, 2002.

[11] Ani Nenkova and Kathleen McKeown. Automatic summarization. *Foundations and Trends® in Information Retrieval*, 2(2):113–259, 2008.

[12] Xiaojun Wan, Jianfeng Yang, and Jianguo Xiao. Automatic summarization of chinese sports news using a trainable summarizer with a novel feature selection method. In *2009 International Conference on Natural Language Processing and Knowledge Engineering*, pages 1–5. IEEE, 2009.

[13] Regina Barzilay and Michael Elhadad. Using lexical chains for text summarization. *Advances in automatic text summarization*, pages 111–121, 1999.

[14] Makbule Ozsoy, Ferda Alpaslan, and Ilyas Cicekli. Text summarization using latent semantic analysis. *J. Information Science*, 37:405–417, 08 2011.

[15] Oguzhan Tas and Farzad Kiyani. A survey automatic text summarization. *PressAcademia Procedia*, 5(1):205–213, 2007.

[16] Abu Kaisar Mohammad Masum, Sheikh Abujar, Md Ashraful Islam Talukder, AKM Shahariar Azad Rabby, and Syed Akhter Hossain. Abstractive method of text summarization with sequence to sequence rnns. In *2019 10th international conference on computing, communication and networking technologies (ICCCNT)*, pages 1–5. IEEE, 2019.

[17] Hongyan Jing. Sentence reduction for automatic text summarization. 10 2000.

[18] Wafaa S. El-Kassas, Cherif R. Salama, Ahmed A. Rafea, and Hoda K. Mohamed. Automatic text summarization: A comprehensive survey. *Expert Systems with Applications*, 165:113679, 2021.

[19] Thomas K Landauer, Peter W Foltz, and Darrell Laham. An introduction to latent semantic analysis. *Discourse processes*, 25(2-3):259–284, 1998.

[20] Amélie Yavchitz, Isabelle Boutron, Aida Bafeta, Ibrahim Marroun, Pierre Charles, Jean Mantz, and Philippe Ravaud. Misrepresentation of randomized controlled trials in press releases and news coverage: a cohort study. 2012.

[21] Romain Paulus, Caiming Xiong, and Richard Socher. A deep reinforced model for abstractive summarization. *arXiv preprint arXiv:1705.04304*, 2017.

[22] Sebastian Gehrmann, Yuntian Deng, and Alexander M Rush. Bottom-up abstractive summarization. *arXiv preprint arXiv:1808.10792*, 2018.