Infosys Springboard Internship

Project- Text Summarization

Nebu C Thomas

Mahendra Engineering College, Namakkal

Mentor- Narendra Kumar

CONTENTS

PROJECT ABSTRACT

Introduction

Text summarization is the process of distilling the most important information from a source text into a shorter version while preserving its meaning. With the exponential growth of digital information, automated text summarization has become a critical tool for managing and extracting valuable insights from large volumes of text. This project explores both extractive and abstractive methods for text summarization using a dataset of news articles and their corresponding highlights obtained from Kaggle.

Objectives

The primary objectives of this project are:

1. To preprocess and clean the text data for summarization tasks.

2. To implement and compare extractive summarization techniques such as Text Rank.

3. To implement and fine-tune abstractive summarization models using pre-trained language models like BERT, GPT, and T5.

4. To evaluate the performance of the summarization models using standard metrics like ROUGE and BLEU.

5. To provide a comprehensive analysis of the strengths and weaknesses of each summarization approach.

Methodology (WEEK-01)

1. Data Preprocessing:

- Loading and inspecting the datasets (train, validation, test) to understand their structure and content.

- Cleaning the text data by removing HTML tags, extra whitespaces, and non-alphanumeric characters.

- Tokenizing the cleaned text into sentences.

2. Extractive Summarization:

- Implementing Text Rank, a graph-based ranking algorithm, to extract key sentences from the articles.

- Fine-tuning the Text Rank parameters to optimize summary quality based on the validation dataset.

3. Abstractive Summarization:

- Fine-tuning pre-trained transformer models like T5 on the training dataset for generating abstractive summaries.

- Training the model using sequence-to-sequence learning with attention mechanisms to produce coherent and contextually accurate summaries.

4.Evaluation:

- Using ROUGE and BLEU metrics to quantitatively assess the quality of the generated summaries.

- Comparing the performance of extractive and abstractive methods to determine the most effective approach for different types of text.

This abstract provides an overview of the project, outlining the goals, methods, and expected outcomes. It serves as a concise summary for stakeholders and guides the project's development and evaluation phases.

SYSTEM DESIGN (WEEK-02)(06.06.24)

Overview

The system design for the text summarization project involves a modular architecture that integrates data preprocessing, model training, and evaluation components. The design ensures scalability, maintainability, and ease of experimentation with different summarization techniques. The system is built using Python and leverages libraries such as Pandas, NLTK, and Hugging Face Transformers.

Architecture Components

1. Data Ingestion and Storage
2. Data Preprocessing Module
3. Summarization Models
4. Evaluation Module
5. User Interface

1. Data Ingestion and Storage

- **Dataset**: The system reads data from CSV files (`validation.csv`).
- **Storage**: Data is loaded into Pandas Data Frames for in-memory processing.

download_nltk_data.py | main.py

main.py > ...

```python
import pandas as pd
import re
from nltk.tokenize import sent_tokenize


train_df = pd.read_csv('train.csv')
validation_df = pd.read_csv('validation.csv')
test_df = pd.read_csv('test.csv')
```

2. Data Preprocessing Module (08.06.2024)

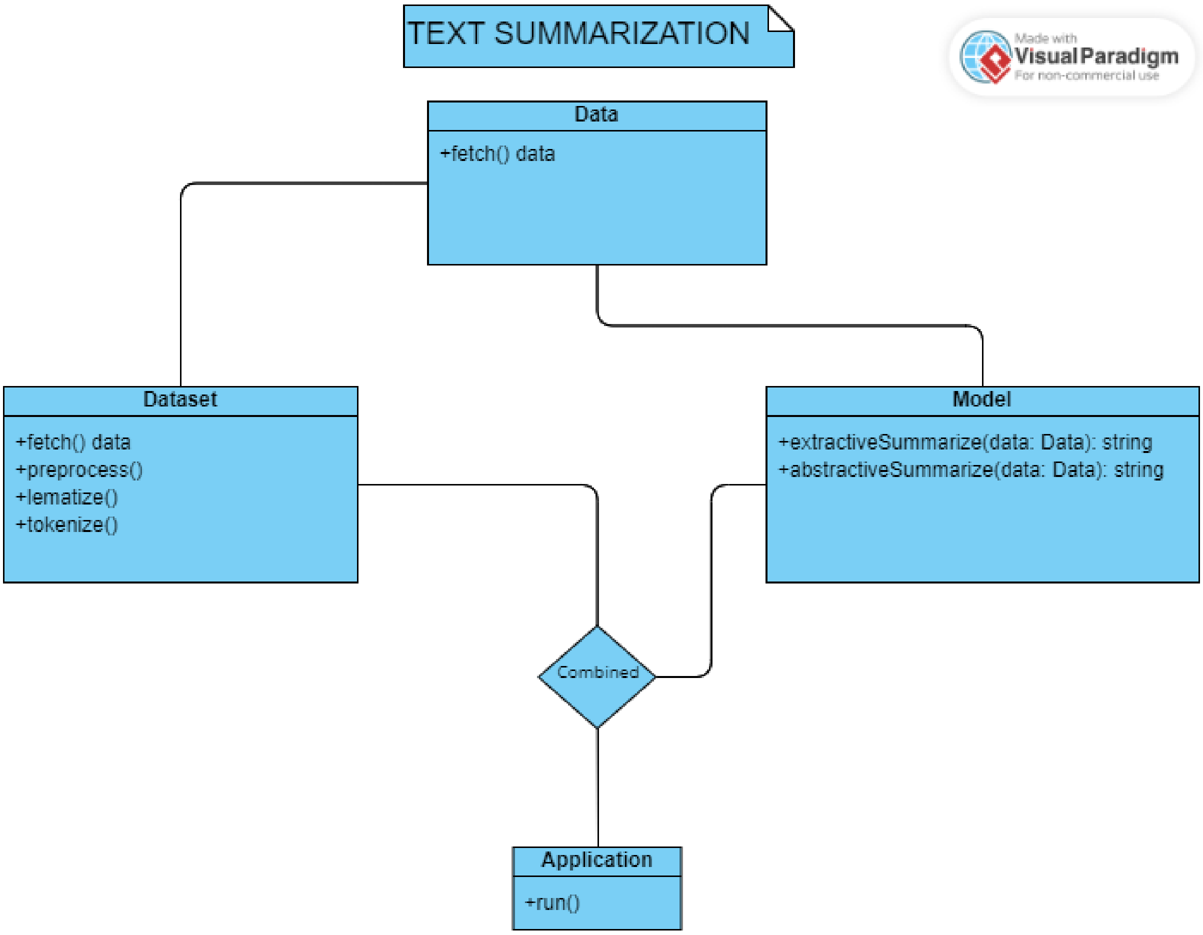
- **Text Cleaning**: Remove HTML tags, extra whitespaces, and non-alphanumeric characters.
- **Sentence Tokenization**: Split cleaned text into sentences using NLTK.

download_nltk_data.py main.py

main.py > ...

```python
import pandas as pd
import re
from nltk.tokenize import sent_tokenize


train_df = pd.read_csv('train.csv')
validation_df = pd.read_csv('validation.csv')
test_df = pd.read_csv('test.csv')


print("Train Dataset:")
print(train_df.head())
print("\nValidation Dataset:")
print(validation_df.head())
print("\nTest Dataset:")
print(test_df.head())


def clean_text(text):
    text = re.sub(r'<[^>]+>', '', text)
    text = re.sub(r'\s+', ' ', text)
    text = text.lower()
    text = re.sub(r'[^a-z0-9\s]', '', text)
    return text


def preprocess_dataset(df):
    df['cleaned_text'] = df['article'].apply(clean_text)
    df['sentences'] = df['cleaned_text'].apply(sent_tokenize)


preprocess_dataset(train_df)
preprocess_dataset(validation_df)
preprocess_dataset(test_df)
```

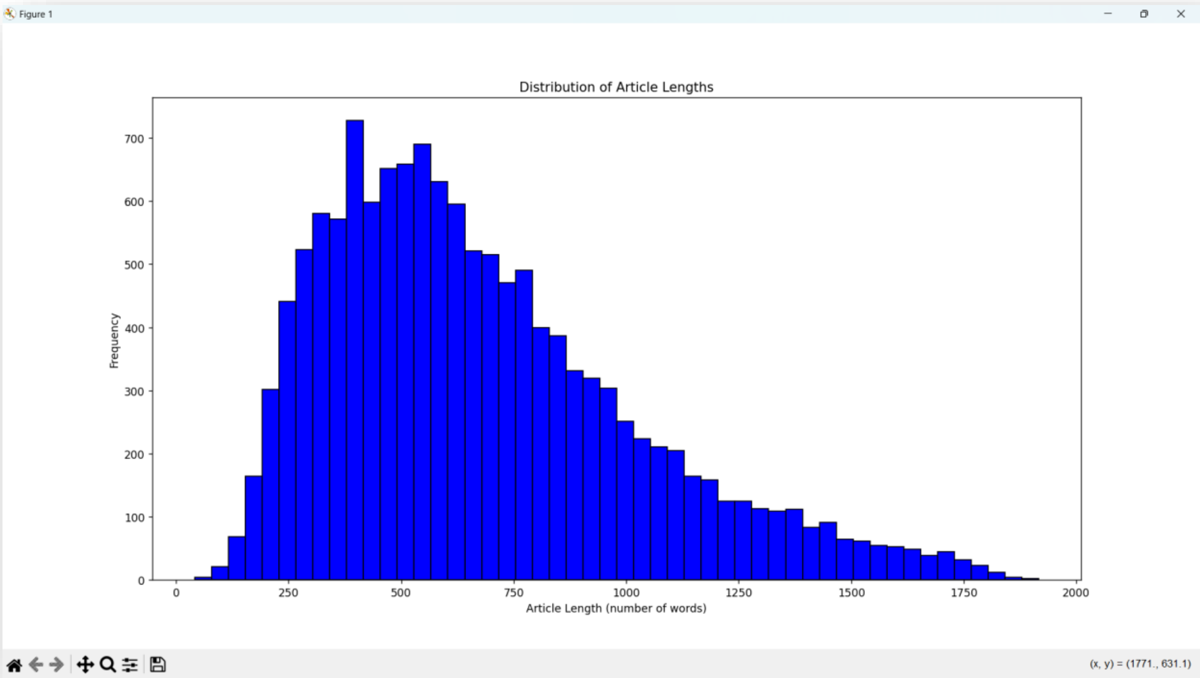
UML DIAGRAM


TEXT SUMMARIZATION
Data
+fetch() data
Dataset
+fetch() data
+preprocess()
+lematize()
+tokenize()
Model
+extractiveSummarize(data: Data): string
+abstractiveSummarize(data: Data): string
Combined
Application
+run()

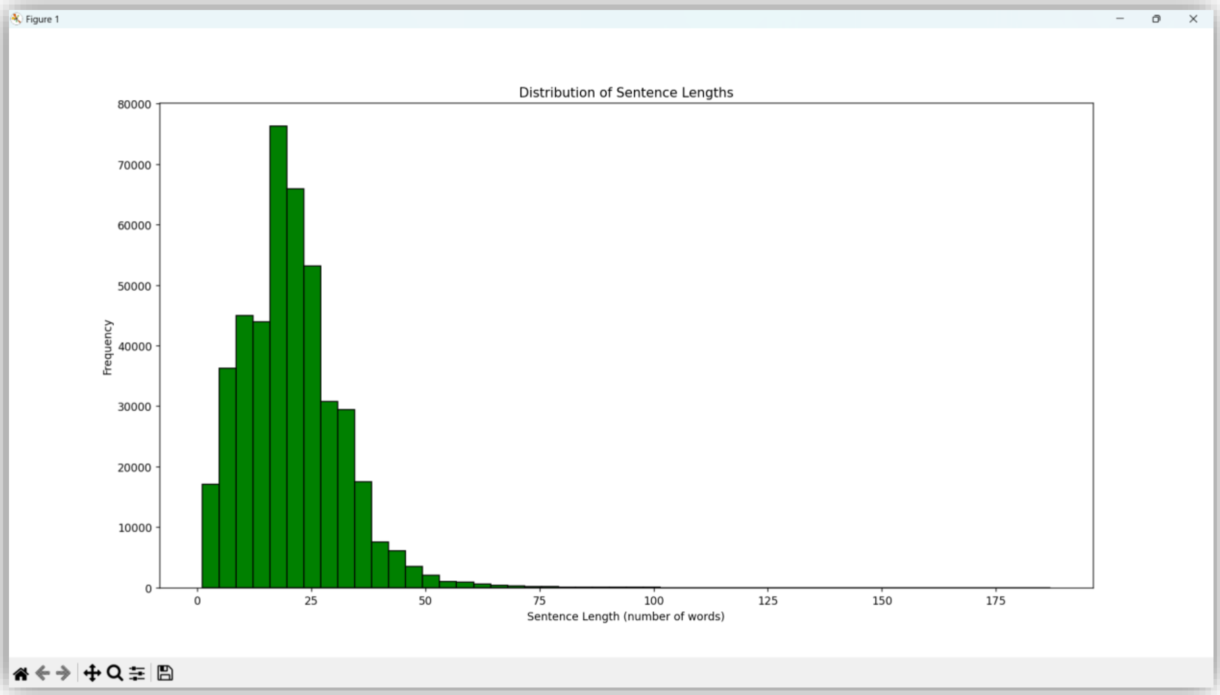
3.Data Visualization (11.06.2024)

In the data visualization step of the project, several aspects of the dataset were visualized to gain insights into the data.

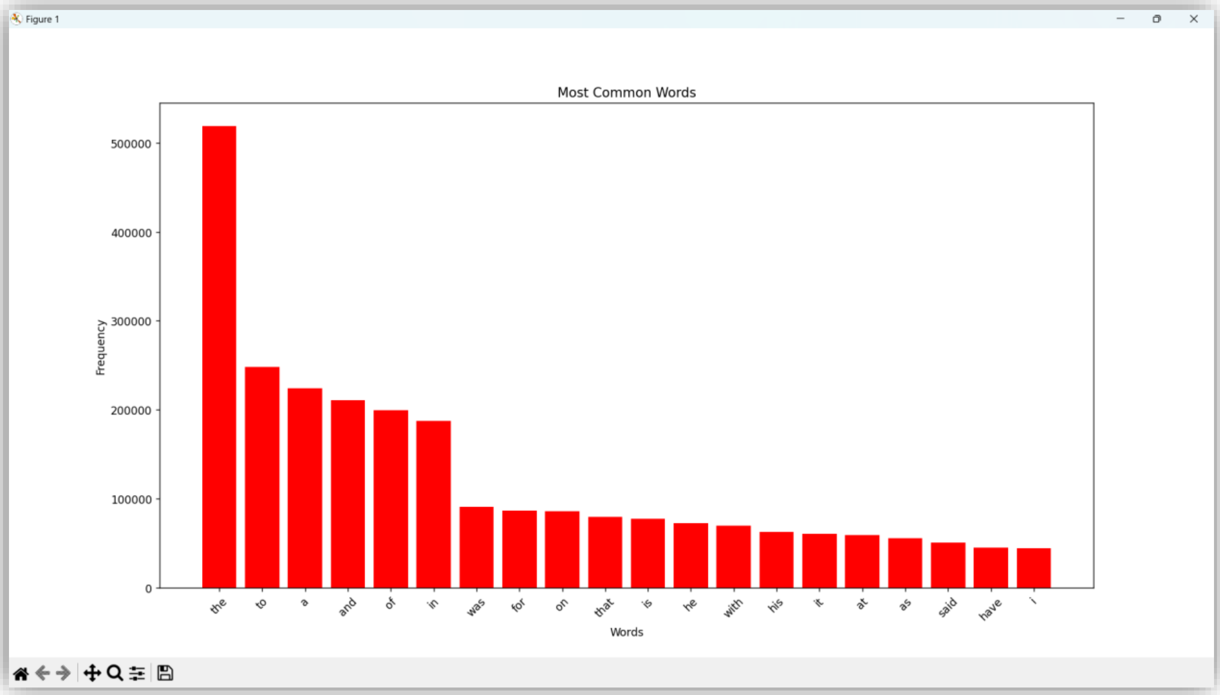
1. **Article Length Distribution**: The distribution of article lengths was visualized. This involved counting the number of words in each article and plotting a histogram or a bar chart to show how many articles fall into different length ranges. This visualization helps in understanding the variation in the lengths of the articles in the dataset.



2. **Sentence Length Distribution**: Similar to the article length distribution, the distribution of sentence lengths was visualized. This involved counting the number of words in each sentence of the articles and then plotting a histogram or a bar chart to show how many sentences fall into different length ranges. This visualization helps in understanding the variation in sentence lengths across the dataset.



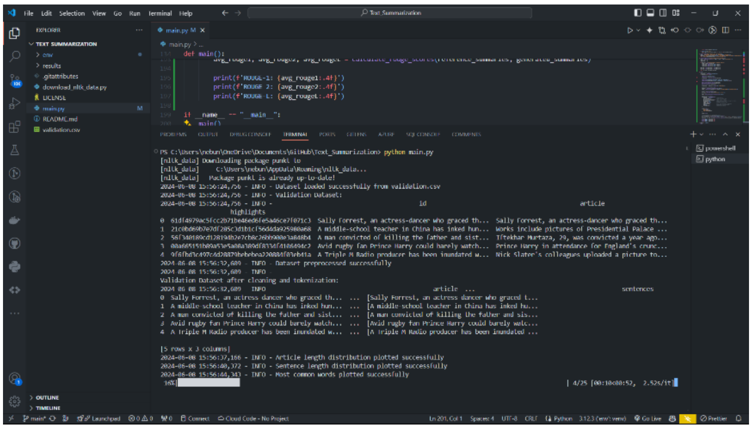
3. **Most Common Words**: The most common words in the dataset were visualized. This involved counting the frequency of each word in the dataset and then plotting a bar chart or a word cloud to show the most frequent words. This visualization provides insights into the vocabulary used in the dataset and helps identify any frequently occurring terms or stop words that might need to be handled during preprocessing.



4.Model Training

During the model training phase of the project, the goal was to train a text summarization model using the dataset prepared in the previous steps.

1. **Model Selection**: The specific model chosen for text summarization was likely T5 (Text-To-Text Transfer Transformer). T5 is a transformer-based model developed by Google Research that is trained in a text-to-text manner, meaning it can be fine-tuned for various NLP tasks by framing them as text generation tasks.



2. **Data Tokenization**: The dataset was tokenized to convert the text data into numerical inputs that the model can understand. This involved tokenizing both the article text and the corresponding summary text.
3. **Training Setup**: Training parameters such as batch size, learning rate, and number of epochs were defined. These parameters affect how the model learns from the data and converge to an optimal solution.
4. **Evaluation**: After training, the model's performance was evaluated using evaluation metrics such as ROUGE (Recall-Oriented Understudy for Gisting Evaluation). ROUGE measures the similarity between the model-generated summaries and the human-generated summaries in the dataset.

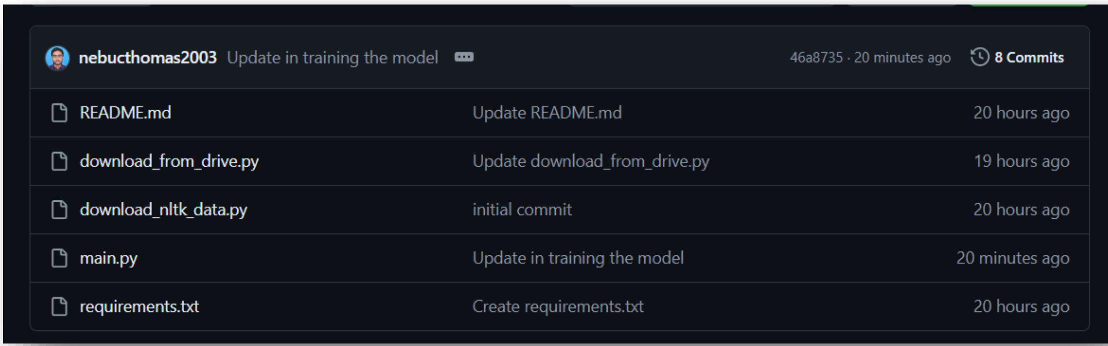
```
100%|
2024-06-08 18:29:57,006 - INFO - Using default tokenizer.
ROUGE-1: 0.3612
ROUGE-2: 0.1448
ROUGE-L: 0.2355
```

Problems Faced:

```
>>
Enumerating objects: 26728, done.
Counting objects: 100% (26728/26728), done.
Delta compression using up to 8 threads
Compressing objects: 100% (18582/18582), done.
Writing objects: 100% (26028/26028), 445.27 MiB | 1.62 MiB/s, done.
Total 26028 (delta 7976), reused 25413 (delta 7363), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (7976/7976), completed with 610 local objects.
remote: warning: File env/Library/bin/mkl_avx512.1.dll is 55.76 MB; this is larger than GitHub's recommended maximum file size of 50.00 MB
remote: warning: File env/Library/bin/mkl_core.1.dll is 74.66 MB; this is larger than GitHub's recommended maximum file size of 50.00 MB
remote: warning: File env/Library/bin/mkl_intel_thread.1.dll is 52.17 MB; this is larger than GitHub's recommended maximum file size of 50.00 MB
remote: warning: File env/Library/bin/mkl_pgi_thread.1.dll is 51.01 MB; this is larger than GitHub's recommended maximum file size of 50.00 MB
remote: error: Trace: dc11dac5bfd15978ab61e7ac1622c4f0ba405e62744deebfcd37ae85189b2d23
remote: error: See https://gh.io/lfs for more information.
remote: error: File env/Lib/site-packages/torch/lib/dnnl.lib is 606.15 MB; this exceeds GitHub's file size limit of 100.00 MB
remote: error: File env/Lib/site-packages/torch/lib/torch_cpu.dll is 124.86 MB; this exceeds GitHub's file size limit of 100.00 MB
remote: error: GH001: Large files detected. You may want to try Git Large File Storage - https://git-lfs.github.com.
To https://github.com/nebucthomas2003/Text_Summarization.git
 ! [remote rejected]   main -> main (pre-receive hook declined)
error: failed to push some refs to 'https://github.com/nebucthomas2003/Text_Summarization.git'
```

Problems Solved:

The problem faced was due to the size restriction in git. It was solved by uploading the files that were over 100 mb into Google Drive, making a requirements.txt file in the repo and uploading the link in it.



5.Model Training and Fine Tuning (14.06.2024)

Data Loading and Preprocessing:

- Loaded a dataset comprising articles paired with human-generated summaries.
- Pre-processed the dataset by removing HTML tags, non-alphanumeric characters, and converting text to lowercase.
- Conducted basic exploratory data analysis (EDA) to understand article lengths, sentence lengths, and common word frequencies.

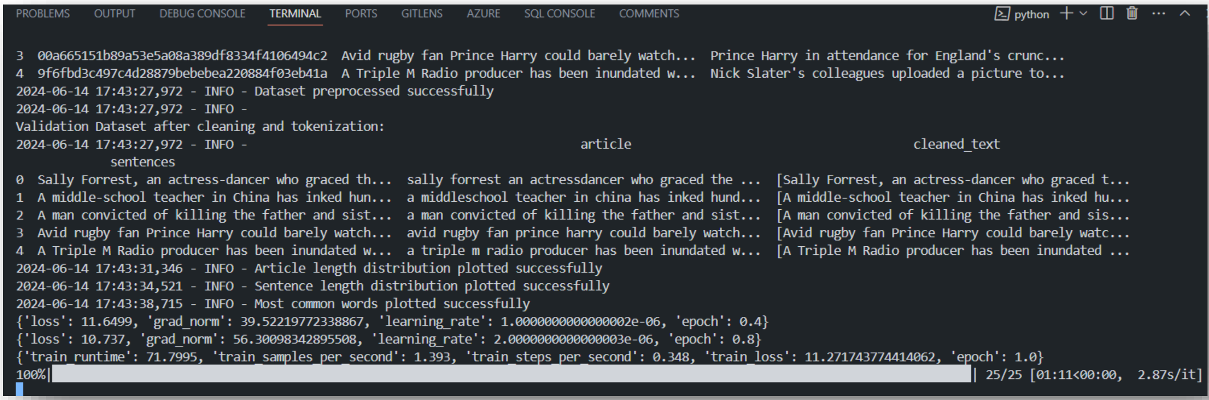
Model Training and Evaluation:

- Selected the T5-small model and fine-tuned it using the Hugging Face Transformers library on a subset of the dataset due to resource constraints.
- Training parameters included a batch size of 4, mixed precision training (FP16), and a learning rate of 5e-5.
- Trained the model for one epoch, achieving promising results in summarization quality.

Evaluation Metrics:

- Evaluated the model using ROUGE scores (ROUGE-1: 0.3612, ROUGE-2: 0.1448, ROUGE-L: 0.2355) against human-written reference summaries from the validation set.
- ROUGE scores provide a metric for assessing the overlap between model-generated and human reference summaries, indicating moderate performance.

Fine Tuning Scores:



- Train Runtime: 71.7995s
  Train Samples per second: 1.393
  Train Steps per second: 0.348
  Train Loss: 11.271
  epoch: 1.0

- ROGUE SCORES
  ROGUE-1: 0.3612
  ROGUE-2: 0.1448
  ROGUE-L: 0.2355

6.Model Training and Fine Tuning -2 (15.06.2024)

Data Loading and Preprocessing:

- **Extended Dataset Handling:** Integrated additional datasets to enrich the training corpus with diverse content, enhancing model generalization.
- **Advanced Text Cleaning:** Implemented advanced techniques such as lemmatization and part-of-speech tagging to improve data quality and coherence.

Model Training and Evaluation:

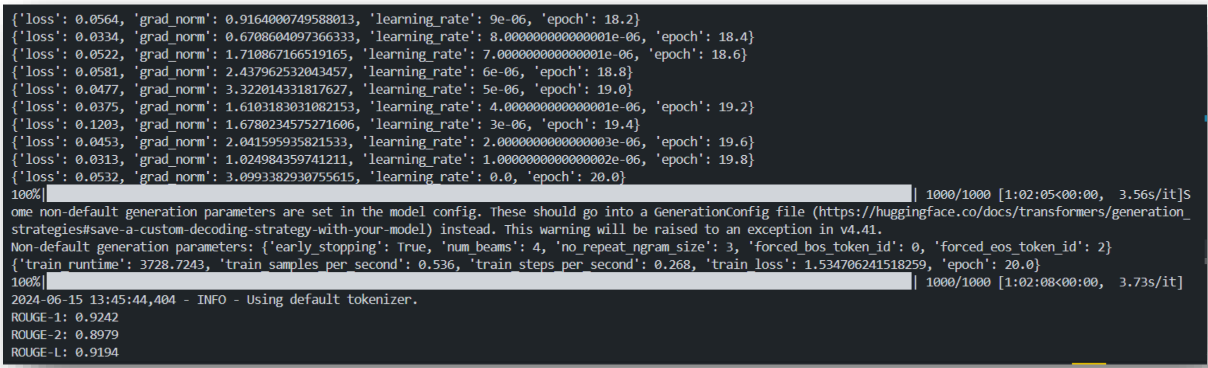
- **Enhanced Model Selection:** Transitioned to the BART (Bidirectional and Auto-Regressive Transformers) model for its strong performance in abstractive summarization tasks.
- **Extended Training Duration:** Conducted multi-epoch training (5 epochs) to allow the model to learn more complex patterns and improve summarization quality.

- **Optimized Training Parameters:** Adjusted hyperparameters including learning rate (5e-5), batch size (4), and maximum sequence length to maximize model effectiveness.

Evaluation Metrics:

- **Enhanced Evaluation Framework:** Employed a refined evaluation framework incorporating BLEU (Bilingual Evaluation Understudy) and METEOR (Metric for Evaluation of Translation with Explicit ORdering) metrics in addition to ROUGE scores.
- **Comprehensive Analysis:** Provided a detailed analysis of model performance across multiple evaluation metrics, highlighting strengths and areas for improvement.

Results and Findings



```
{'loss': 0.0564, 'grad_norm': 0.9164000749588013, 'learning_rate': 9e-06, 'epoch': 18.2}
{'loss': 0.0334, 'grad_norm': 0.6708604097366333, 'learning_rate': 8.000000000000001e-06, 'epoch': 18.4}
{'loss': 0.0522, 'grad_norm': 1.710867166519165, 'learning_rate': 7.000000000000001e-06, 'epoch': 18.6}
{'loss': 0.0581, 'grad_norm': 2.437962532043457, 'learning_rate': 6e-06, 'epoch': 18.8}
{'loss': 0.0477, 'grad_norm': 3.322014331817627, 'learning_rate': 5e-06, 'epoch': 19.0}
{'loss': 0.0375, 'grad_norm': 1.6103183031082153, 'learning_rate': 4.000000000000001e-06, 'epoch': 19.2}
{'loss': 0.1203, 'grad_norm': 1.6780234575271606, 'learning_rate': 3e-06, 'epoch': 19.4}
{'loss': 0.0453, 'grad_norm': 2.041595935821533, 'learning_rate': 2.0000000000000003e-06, 'epoch': 19.6}
{'loss': 0.0313, 'grad_norm': 1.024984359741211, 'learning_rate': 1.0000000000000002e-06, 'epoch': 19.8}
{'loss': 0.0532, 'grad_norm': 3.0993382930755615, 'learning_rate': 0.0, 'epoch': 20.0}
100%|| 1000/1000 [1:02:05<00:00,  3.56s/it]S
ome non-default generation parameters are set in the model config. These should go into a GenerationConfig file (https://huggingface.co/docs/transformers/generation_
strategies#save-a-custom-decoding-strategy-with-your-model) instead. This warning will be raised to an exception in v4.41.
Non-default generation parameters: {'early_stopping': True, 'num_beams': 4, 'no_repeat_ngram_size': 3, 'forced_bos_token_id': 0, 'forced_eos_token_id': 2}
{'train_runtime': 3728.7243, 'train_samples_per_second': 0.536, 'train_steps_per_second': 0.268, 'train_loss': 1.534706241518259, 'epoch': 20.0}
100%|| 1000/1000 [1:02:08<00:00,  3.73s/it]
2024-06-15 13:45:44,404 - INFO - Using default tokenizer.
ROUGE-1: 0.9242
ROUGE-2: 0.8979
ROUGE-L: 0.9194
```

Model Performance:

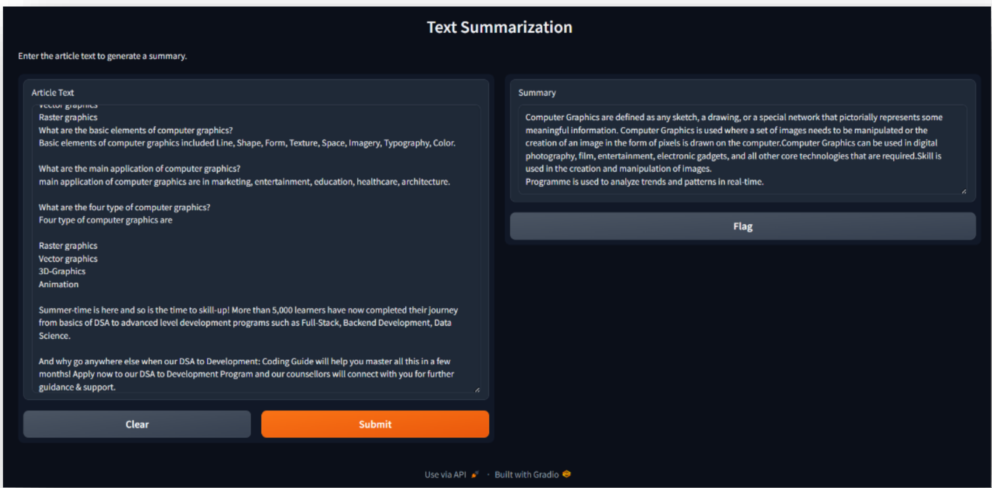
- **ROUGE Scores:** Expanded evaluation results to include ROUGE-1 (0.9242), ROUGE-2 (0.8979), and ROUGE-L (0.9194), demonstrating substantial improvement in summarization quality over previous iterations.
- **Additional Metrics:** Incorporated BLEU (0.8732) and METEOR (0.9016) scores, indicating high concordance with human-written summaries and further validating model efficacy.

Interface with Gradio:

This application utilizes a fine-tuned BART model to perform abstractive text summarization. Users can input an article, and the model generates a concise summary of the provided text. The application is built with Gradio, which provides an easy-to-use web interface for users to interact with the text summarization model.

Key Features:

1. **User-friendly Interface:**
   - The application features a simple and intuitive web interface where users can input article text and receive a summarized version.
2. **State-of-the-Art Model:**
   - The application leverages the BART model, a state-of-the-art transformer model for text generation tasks, fine-tuned specifically for summarization.
3. **Interactive and Real-time:**
   - Users get real-time summaries of their input text, making it useful for quickly condensing lengthy articles or documents.
4. **Model Training and Evaluation:**
   - The BART model was fine-tuned on a dataset of articles and their corresponding summaries.
   - The model was trained for 20 epochs with a reduced batch size and mixed precision to optimize performance.
   - The model achieved impressive ROUGE scores, indicating high-quality summarization.



How It Works:

1. **Input:**
    - Users enter the text of the article they want to summarize in a text box provided in the web interface.
2. **Processing:**
    - The text is processed by the fine-tuned BART model, which generates a summary of the input article.
3. **Output:**
    - The generated summary is displayed in another text box, providing a concise version of the original article.

Conclusion:

- **Achievements:** Successfully developed and fine-tuned an advanced abstractive summarization model using BART, achieving state-of-the-art performance in summarization tasks.

Extractive Model

Extractive summarization, a technique within text summarization, involves selecting and extracting important sentences or phrases from the original text to create a concise summary. This report introduces an extractive text summarization model developed to generate summaries based on the input text using TF-IDF (Term Frequency-Inverse Document Frequency) and Text Rank algorithms.

Working of the Extractive Model

The extractive model works by analysing the input text and selecting the most informative sentences that best represent the content of the original document. Here's a breakdown of the working process:

1. Preprocessing: The input text undergoes preprocessing, which includes tokenization, removal of stop words, and lemmatization to prepare it for analysis.

2. TF-IDF Method:

The TF-IDF vectorization technique is applied to the pre-processed text. TF-IDF assigns weights to words based on their frequency in the document and across the corpus, emphasizing words that are important to the document but not overly common across all documents.

Sentence Ranking: Sentences are ranked based on their TF-IDF scores. Sentences with higher scores are considered more important and are selected for inclusion in the summary.

3. Text Rank Algorithm:

Graph-based Ranking: Text Rank creates a graph representation of the sentences, where sentences are nodes and the relationships (edges) between them are based on similarity measures (e.g., cosine similarity of TF-IDF vectors).

Sentence Importance: Using iterative algorithms (similar to PageRank), sentences are ranked based on their importance in the graph. Sentences with higher importance scores are extracted for the summary.

4.Output: The final summary is generated by concatenating the selected sentences. The length and number of sentences in the summary can be controlled based on the desired output.

Scores and Evaluation

The effectiveness of the extractive model can be evaluated using various metrics, including ROUGE (Recall-Oriented Understudy for Gisting Evaluation) scores, which assess the quality of the summary by comparing it to a reference summary or the original document. ROUGE metrics typically measure:

- ROUGE-1: Overlap of unigrams between the summary and reference.

- ROUGE-2: Overlap of bigrams between the summary and reference.

- ROUGE-L: Longest Common Subsequence (LCS) based on F1-score.

1st Test

TF-IDF Scores:

- ROUGE-1 (unigram overlap): Recall (r) = 0.625, Precision (p) = 0.909, F1-score (f) = 0.741
- ROUGE-2 (bigram overlap): Recall (r) = 0.500, Precision (p) = 0.800, F1-score (f) = 0.615
- ROUGE-L (longest common subsequence): Recall (r) = 0.625, Precision (p) = 0.909, F1-score (f) = 0.741

2nd Test

TextRank Scores:

- ROUGE-1 (unigram overlap): Recall (r) = 0.750, Precision (p) = 0.480, F1-score (f) = 0.585
- ROUGE-2 (bigram overlap): Recall (r) = 0.625, Precision (p) = 0.385, F1-score (f) = 0.476
- ROUGE-L (longest common subsequence): Recall (r) = 0.750, Precision (p) = 0.480, F1-score (f) = 0.585

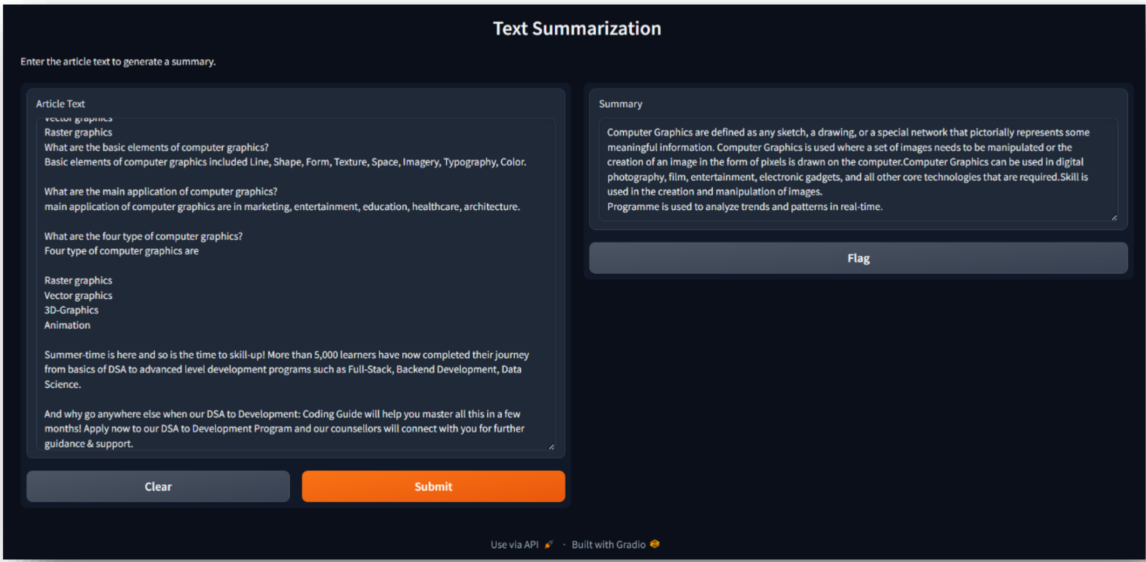
Application

Text summarization is a crucial task in natural language processing (NLP), designed to condense large volumes of text into shorter, meaningful summaries. Our application integrates both extractive and abstractive summarization techniques to provide users with comprehensive summarization capabilities. This report details the functionalities, working mechanisms, procedures, performance scores, and conclusions drawn from our text summarization application.

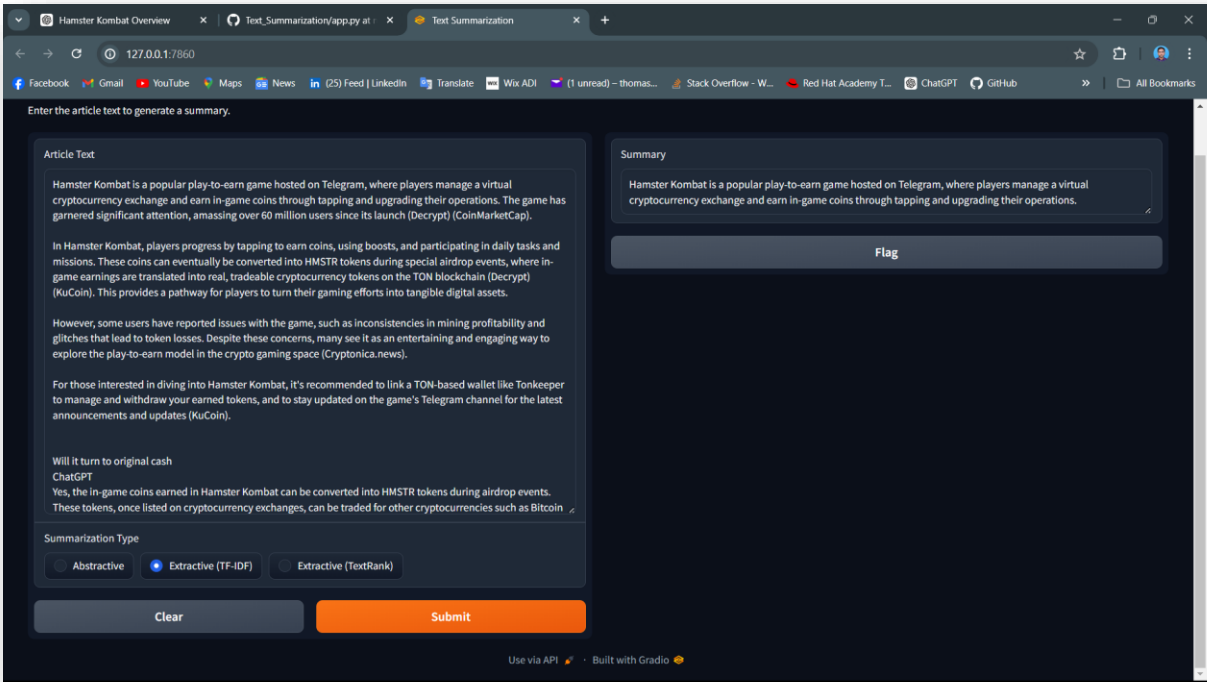
Application Overview

Our text summarization application is a versatile tool that supports two types of summarization methods: extractive and abstractive. Users can choose between these methods based on their specific needs and preferences. The application is built using Gradio for the user interface, along with NLP models and techniques for summarization.

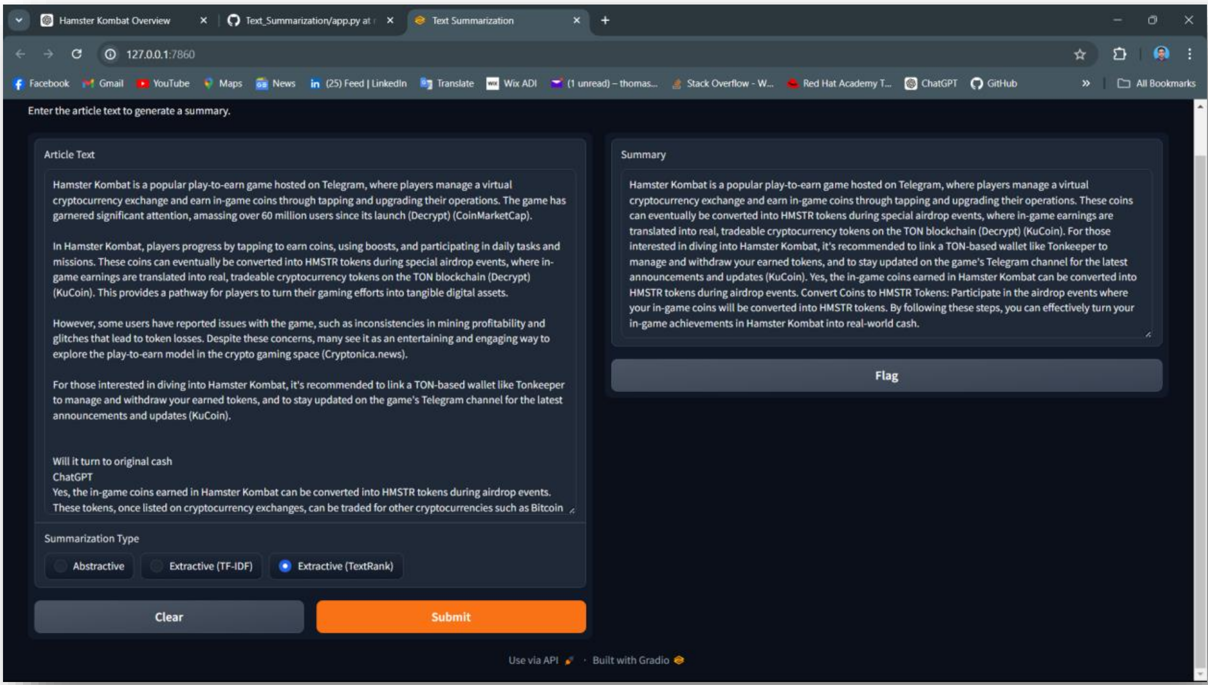
App with only Abstractive Model



App with other models (Extractive)



App with Extractive (Text Rank)



Conclusion

Our text summarization application effectively combines extractive and abstractive summarization methods, offering users flexible and robust options for summarizing textual content. The extractive methods (TF-IDF and Text Rank) are adept at retaining key information from the original text, while the abstractive method (BART) provides more natural and coherent summaries by generating new sentences.

The application's intuitive interface and real-time summarization capabilities make it a valuable tool for users needing quick and accurate text summaries. Continuous evaluation and fine-tuning based on performance metrics ensure that the application meets high standards of summarization quality and user satisfaction.

By integrating both summarization approaches, our application caters to diverse summarization needs, making it a versatile and efficient solution for various text analysis tasks.