GIST: LESS IS MORE

A CAPSTONE PROJECT REPORT

Submitted in partial fulfillment of the requirement for the award of the Degree of

BACHELOR OF TECHNOLOGY
IN
COMPUTER SCIENCE AND ENGINEERING

by

S Vijay Balaji (19BCE7571)
M Naveen (19BCN7185)
Sri Santhosh Reddy (19BCN7091)

Under the Guidance of

DR. ANUPAMA NAMBURU

SCHOOL OF COMPUTER SCIENCE ENGINEERING
VIT-AP UNIVERSITY
AMARAVATI- 522237

JANUARY 2023

CERTIFICATE

This is to certify that the Capstone Project work titled "**GIST: LESS IN MORE**" that is being submitted by **S Vijay Balaji (19BCE7571), M Naveen (19BCN7185)** and **Sri Santhosh Reddy (19BCN7091)** is in partial fulfillment of the requirements for the award of Bachelor of Technology, is a record of bonafide work done under my guidance. The contents of this Project work, in full or in parts, have neither been taken from any other source nor have been submitted to any other Institute or University for award of any degree or diploma and the same is certified.

DR. ANUPAMA NAMBURU
Guide

The thesis is satisfactory / unsatisfactory

Internal Examiner **External Examiner**

Approved by

PROGRAM CHAIR
B. Tech. CSE

DEAN
School Of Computer Science Engineering

ACKNOWLEDGEMENTS

The satisfaction and euphoria that accompany the successful completion of any task would be impossible without the mention of the people who made it possible, whose constant guidance and encouragement crowned our efforts with success.

We take this opportunity to express our gratitude and regards to the guide **Prof.Anupama Namburu** for his exemplary guidance, monitoring and constant encouragement throughout the course of this project. We also take this opportunity to express a deep sense of gratitude to my friends for their support and motivation which helped me in completing this task through its various stages. We are obliged to the faculty members of the Department of Computer Science & Engineering at VIT Amaravati for the valuable information provided by them in their respective fields.

We are grateful for their cooperation during the period of our assignment. Lastly, We thank our parents for their constant encouragement without which this assignment would not have been possible.

Finally a note of thanks to the teaching and non-teaching staff of Dept of Computer Science and Engineering, for their cooperation extended to me, and my friends, who helped me directly or indirectly in the course of the project work.

S Vijay Balaji (19BCE7571)

M Naveen (19BCN7185)

Sri Santhosh Reddy (19BCN7091)

ABSTRACT

Gist is a suite of applications that focus on abstractive summarization, an NLP tool that can combine and paraphrase a given text to produce a brief summary of its contents. Due to the enormous amount of information at our disposal and the development of Internet technologies, text summarization has emerged as a critical method for analysing text data. The two different text summarization techniques are extractive and abstractive. Choosing strong passages from a text based on word and phrase attributes, then combining them to create a summary, is known as extractive summarising. The significance of sentences is assessed using their statistical and linguistic features. An abstractive summary is used to understand the key concepts in a text and then convey those concepts in straightforward, everyday language. In this project, abstractive summarization is being used. The outcomes demonstrated that the summarizer was able to produce summaries with a high degree of recall and precision, demonstrating that it was able to precisely extract the essential details from the source text. The summarizer's output was also observed to be coherent and well-written, suggesting that it was successful in synthesising and paraphrasing the original content. These evaluations' findings suggest that the abstractive text summarizer is a useful tool for cutting down on the time and effort needed to read extensive amounts of text. Similar to how people communicate, the abstractive text summarization algorithms produce new words and phrases that convey the most important details from the source text. A fascinating area of machine learning that is gaining ground is text summarization. We can anticipate innovations as this field of study develops that will help with accurately and fluently condensing lengthy text documents. In general, the abstractive text summarizer performed well and showed promise as a tool for text summarization tasks in various fields that we have tested it out in, be it news, gmail, books, audio or other types of summarizations.

TABLE OF CONTENTS

List of Outputs

List of Figures

CHAPTER 1: INTRODUCTION

Text summarization is a task in natural language processing that involves synthesising and paraphrasing the content of lengthy documents to produce concise and coherent summaries. The main ideas and points of the original text should be covered in these summaries, but extraneous information and repetition should be left out. Abstractive summarizers can produce summaries that are more logical and understandable for people because they are able to comprehend the meaning and context of the text. The two different types of text summarization techniques are extractive and abstractive. High-ranking sentences from a text are chosen using an extractive summarising approach based on word and phrase attributes, and then they are combined to create a summary. Both linguistic and statistical properties are used to determine the significance of sentences. An abstractive summary is a tool for understanding the key concepts in a piece of writing and then communicating those concepts in clear, everyday language.

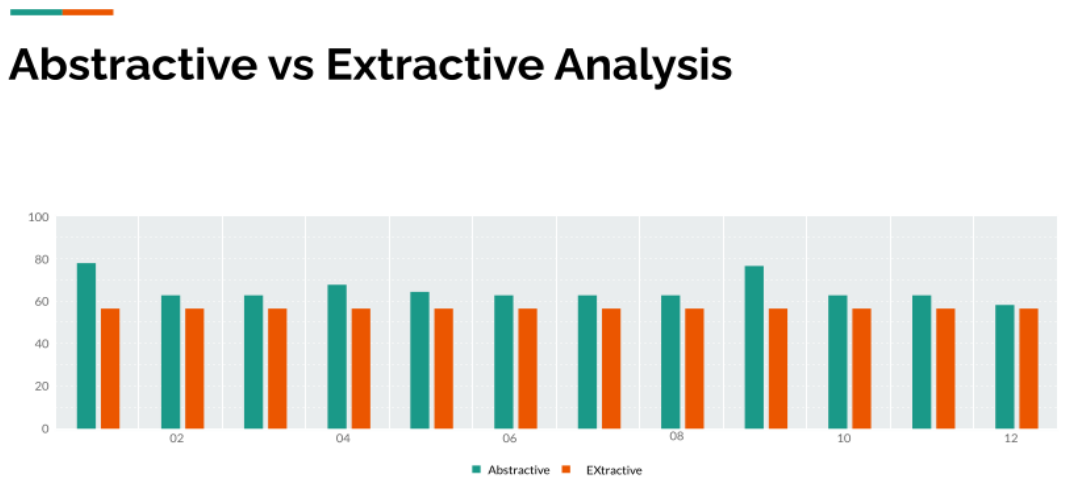


Figure 1 Abstractive vs Extractive Analysis

There are several applications for abstractive text summarization, including news articles, legal documents, research papers, and customer reviews. In today's fast-paced and information-rich society, it is often difficult for individuals to keep up with the vast amount of text available online. Abstractive summarizers can help reduce the time and effort required to consume large amounts of text, by providing a condensed and easy-to-understand version of the original text.

The development of abstractive text summarizers has been an active area of research in natural language processing and machine learning. There are various approaches to abstractive summarization, including neural network models, rule-based systems, and hybrid approaches that combine multiple methods. In this report, we present an abstractive text summarizer that was developed using a neural network model trained on a large dataset of news articles. We will describe the model architecture, training process, and evaluation methods used to assess the performance of the summarizer. Overall, our goal is to demonstrate the effectiveness of the abstractive text summarizer in generating high-quality and relevant summaries of long documents.

1.1 Objectives

The following are the objectives of this project:
It's worth noting that text summarization can be used for a variety of purposes beyond the ones mentioned above. For example, text summarization can be used to:

- Improve the readability and clarity of a document: By providing a concise and easy-to-understand summary, text summarization can help improve the readability and clarity of a document, making it more accessible to a wider audience.
- Assist with content creation and analysis: Text summarization can help identify key themes and ideas within a document, which can be useful for content creation and analysis.
- Support content recommendation and personalization: Text summarization can be used to generate summaries of documents that can be used to recommend or personalize content based on user interests and preferences.
- Enable the creation of summary-based search engines: Text summarization can be used to create search engines that use summaries rather than full documents as the basis for search results. This can be useful in situations where it is not practical or efficient to search through large numbers of full documents.

- Support text classification and categorization: Text summarization can be used to classify and categorize documents based on their content, which can be useful for organizing and managing large collections of documents.
- Create a medium for quick and easy way to access news articles and understanding changes around the world in a few seconds.
- Summarize Gmail content to collect statistical data and to gather a pool of information within seconds to better understand and grasp mails than go through each one mail at a time, thus helping in improve time efficiency.

1.2 Background and Literature Survey

Gist started out as a project built as part of NMIT Hacks 2021, national level hackathon and came runners up. It was then further developed with improved models, an added front-end and features enhanced.

Gist started as a simple prototype to easily and efficiently summarize any news article into 60 or lesser words for quicker grasp of data, but since then it's come a long way. Today Gist hosts 3 seperate applications, each with their own unique functionalities, yet derived from the same core.

Gist summarizes content from text, documents, pdfs, images so you can upload your physical news paper! And even other online news websites. This helps you consume important data much quicker.

Gist at it's heart as a core application acts as a medium where you can read several 60 or lesser word news articles in one place, similar to that of inShorts.

And finally the Gmail Summarizer was built to quickly get insights on your email data as it becomes harder everyday to keep track of the infinite mails we recieve everyday.

Gist is also open source and hence aids in easily adding more features, upgrades, bug fixes and welcomes changes and commits from across the globe.

Text summarization is the process of generating a condensed version of a document that captures its main points and ideas. It has been an active area of research in natural language processing and machine learning for several decades, with the goal of developing tools that can automatically extract key information from large amounts of text.

The rule-based approach, which used predefined rules to extract important phrases and sentences from the original text, was one of the earliest methods of text summarization. These systems frequently produced summaries that were choppy and difficult to understand because of their limited capacity to comprehend the meaning and context of the text.

Machine learning methods are increasingly being used for text summarization in recent years. These models learn to produce summaries by synthesising and rephrasing the original text after being trained on sizable datasets of documents and summaries.

The effectiveness of machine learning models for text summarization has been the subject of numerous studies. Nallapati et al. (2016) conducted a review of the literature and discovered that machine learning performed better than other methods in terms of relevance and summary quality. These models have also proven to be capable of handling a wide variety of languages and domains, as well as being robust in the presence of noise and errors in the original text.

Nevertheless, there are a number of difficulties and restrictions to take into account when using machine learning models for text summarization. The requirement for a lot of high-quality training data presents one difficulty. Large datasets of documents and summaries are necessary for machine learning models to learn how to produce accurate and cogent summaries. The collection and annotation of this data can be costly and time-consuming, and it might not always be practical for some languages or domains. The development and optimization of neural network models can be challenging and require specialised knowledge. They may also need to be updated and maintained frequently to maintain their effectiveness.

1.3 Organization of the Report

The remaining chapters of the project report are described as follows:

- Chapter 2 contains the proposed system, methodology and software details.
- Chapter 3 implications and limitations also provide recommendations for future work.
- Chapter 4 discusses the results obtained after the project was implemented.
- Chapter 5 concludes the report.
- Chapter 6 consists of codes.
- Chapter 7 gives references.

CHAPTER 2: GIST, AN ABSTRACTIVE TEXT SUMMARIZER USING NLP

This Chapter describes the proposed system, working methodology and software details.

DistilBART (Distilled Bidirectional Encoder Representations from Transformers) is a distilled version of the BART (Bidirectional Encoder Representations from Transformers) model, which was trained to perform a variety of natural language processing tasks, including machine translation, summarization, and text generation.

The key idea behind distillation is to train a smaller, more efficient model (called the "student" model) to perform a task by learning from the output of a larger, pre-trained model (called the "teacher" model) that has already been trained to perform the task. The student model is trained to mimic the output of the teacher model, and in the process, it learns to perform the task with high accuracy.

In the case of DistilBART, the student model is a smaller version of BART, with fewer layers and fewer hidden units. The teacher model is a pre-trained version of BART, and the student model is trained to mimic the output of the teacher model on a variety of natural language processing tasks.

The benefits of using distillation to train the student model include faster training times, lower memory requirements, and the ability to deploy the model on devices with limited resources.

2.1 Proposed System

The following block diagram (figure 2) shows the system architecture of this project.

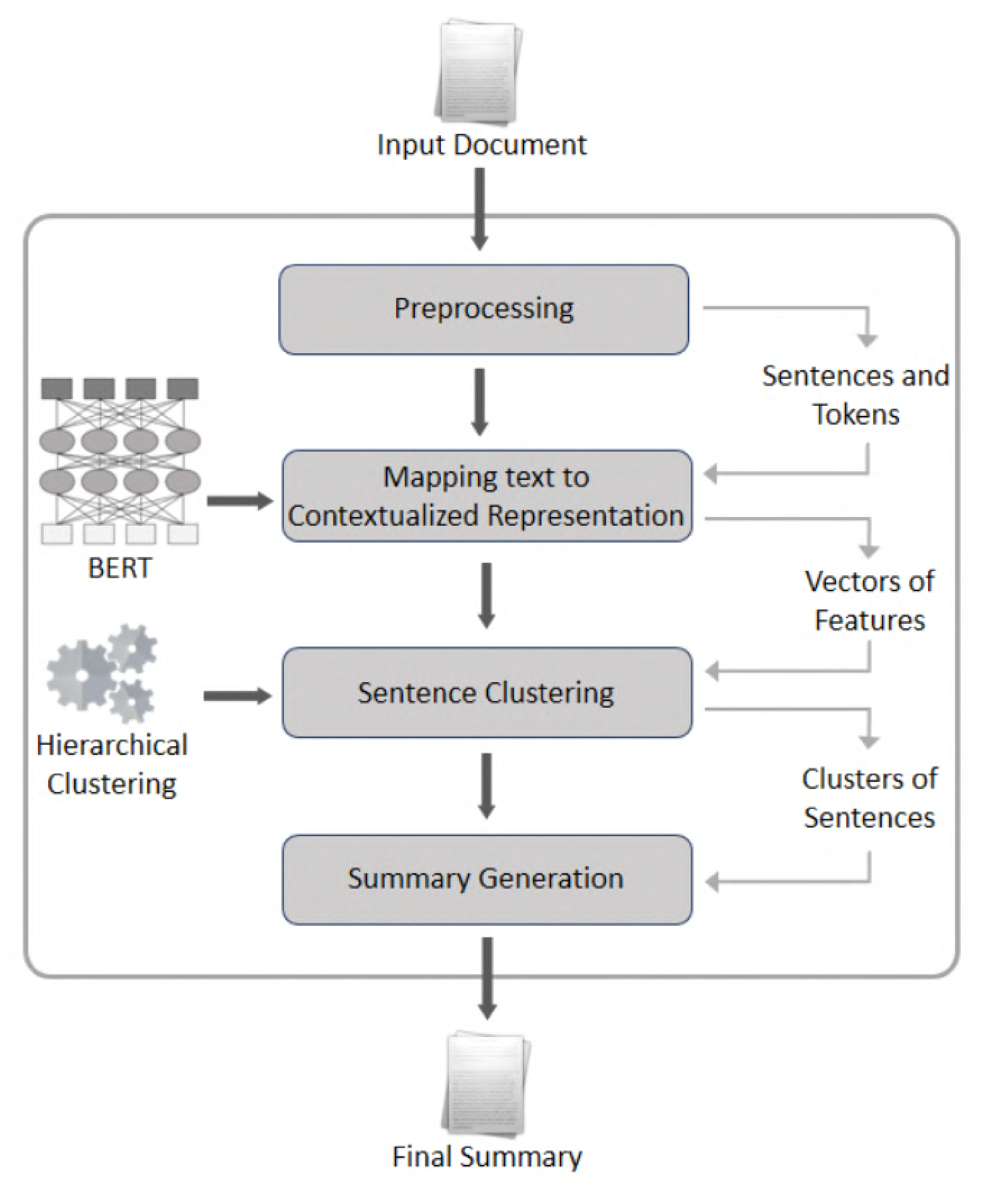


Figure 2

2.2 Working Methodology

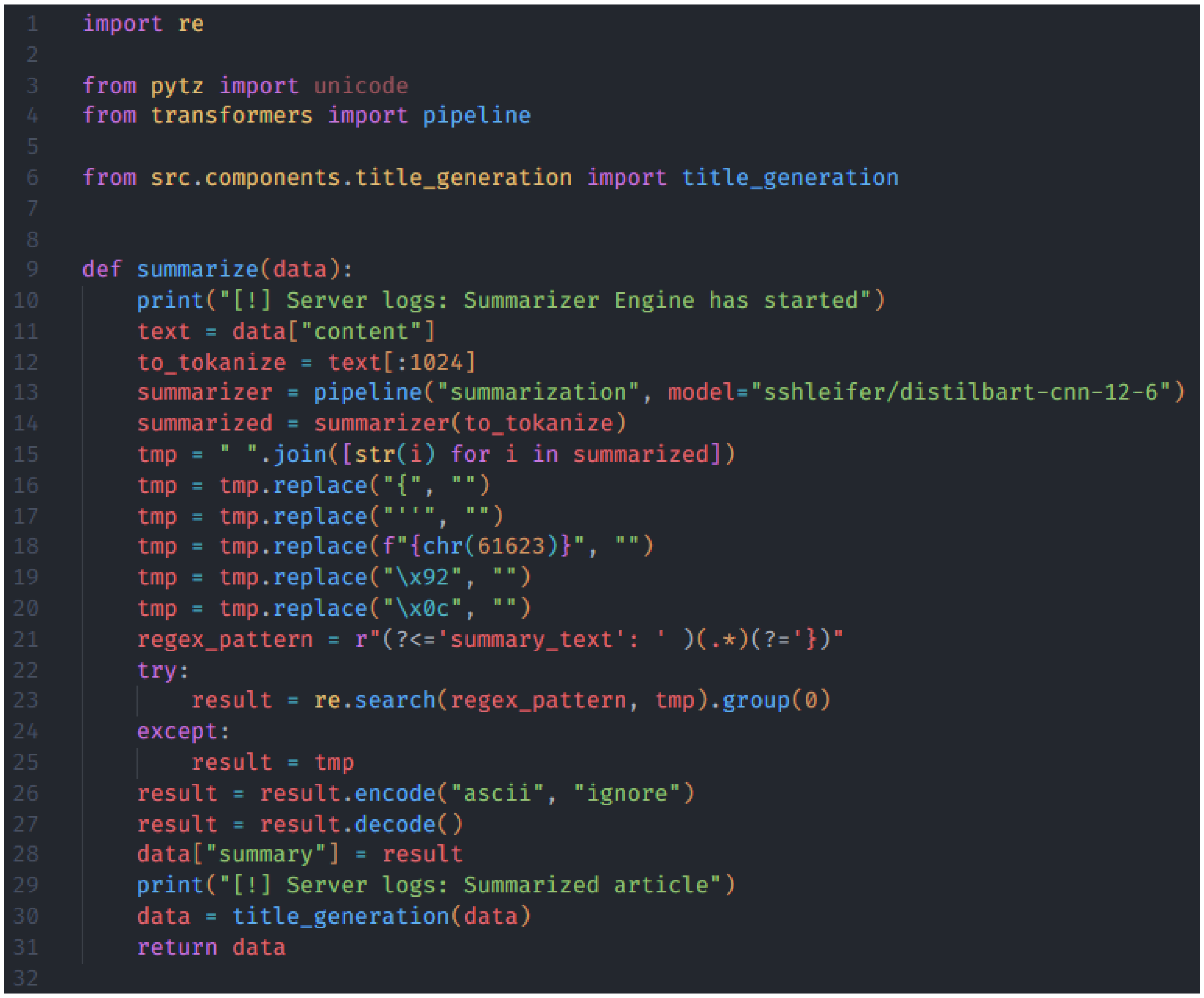
This whole Project has two parts of section, Extraction and then followed by Summarization. Here, the Extraction of content takes place from multiple sources including Web News Articles, Images, Documents, and finally PDFs. The Extracted content we then pass it to our Summarizer Engine to Summarize to get Crisp and Clear output without worrying about the context present in the article or any other medium.

When processing small texts, DistilBART first encodes the input text into a fixed-length vector representation, known as an embedding. This embedding captures the meaning and context of the input text, and is used as input to the model's transformer layers.

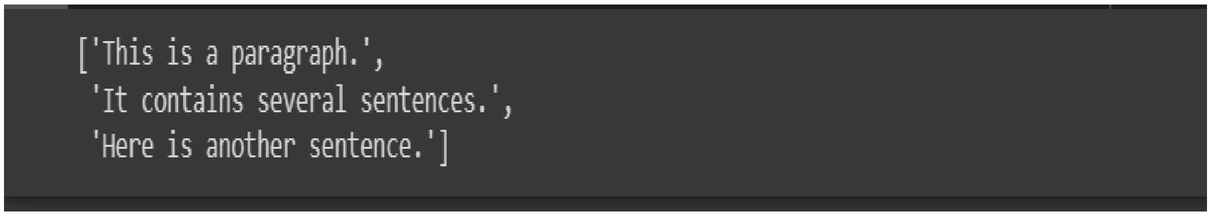
The transformer layers then process the embedding and generate a prediction for the task at hand. For example, if the task is machine translation, the transformer layers will generate a translation of the input text in the target language. If the task is summarization, the transformer layers will generate a summary of the input text.

2.3 Procedure

Step 1. Convert the paragraph into sentences: First, let's split the paragraph into its corresponding sentences. The best way of doing the conversion is to extract a sentence whenever a period appears. Example:



```python
import re

from pytz import unicode
from transformers import pipeline

from src.components.title_generation import title_generation


def summarize(data):
    print("[!] Server logs: Summarizer Engine has started")
    text = data["content"]
    to_tokanize = text[:1024]
    summarizer = pipeline("summarization", model="sshleifer/distilbart-cnn-12-6")
    summarized = summarizer(to_tokanize)
    tmp = " ".join([str(i) for i in summarized])
    tmp = tmp.replace("{", "")
    tmp = tmp.replace("''", "")
    tmp = tmp.replace(f"{chr(61623)}", "")
    tmp = tmp.replace("\x92", "")
    tmp = tmp.replace("\x0c", "")
    regex_pattern = r"(?<='summary_text': ' )(.*)(?='})"
    try:
        result = re.search(regex_pattern, tmp).group(0)
    except:
        result = tmp
    result = result.encode("ascii", "ignore")
    result = result.decode()
    data["summary"] = result
    print("[!] Server logs: Summarized article")
    data = title_generation(data)
    return data

```



```
['This is a paragraph.',
 'It contains several sentences.',
 'Here is another sentence.']
```

Step 2. Text processing: Next, let's do text processing by removing the stop words (prevalent words with little meaning such as "and" and "the"), numbers, punctuation, and other special characters from the sentences.

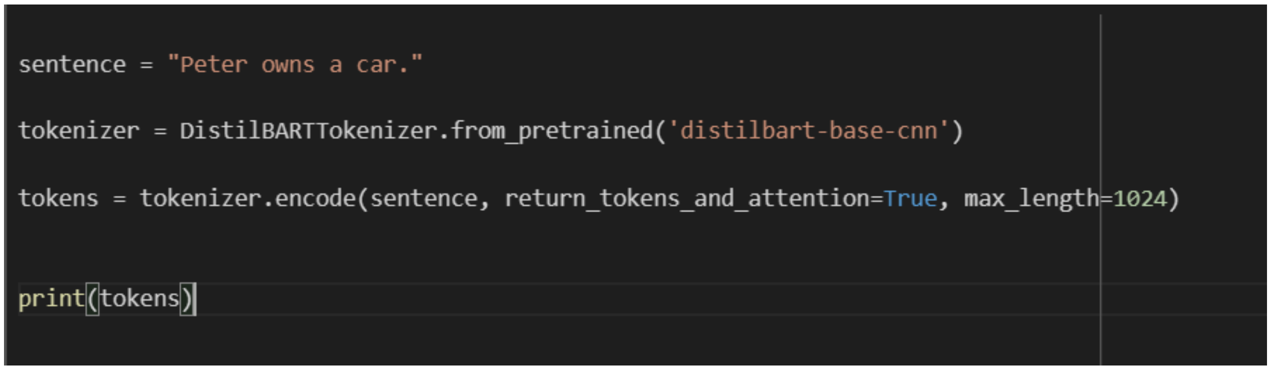
Refer - Text Classification on GLUE - Colaboratory (google.com)

Step 3. Tokenization: Dividing a para into a set of statements or dividing a statement into a set of words.

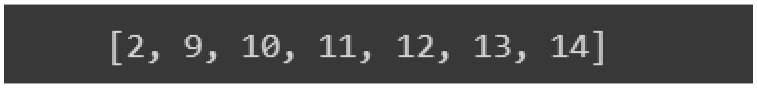
Input: Peter owns a car

Output: [[Peter], [owns], [a], [car]]

Example:



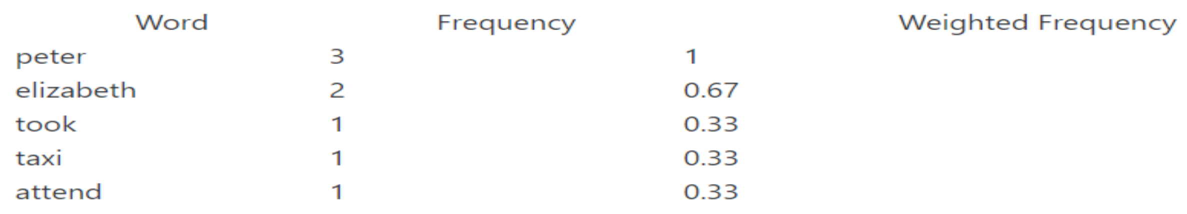
```
sentence = "Peter owns a car."

tokenizer = DistilBARTTokenizer.from_pretrained('distilbart-base-cnn')

tokens = tokenizer.encode(sentence, return_tokens_and_attention=True, max_length=1024)


print(tokens)
```



```
[2, 9, 10, 11, 12, 13, 14]
```

Step 4. Evaluate the weighted occurrence frequency of the words Thereafter, let's calculate the weighted occurrence frequency of all the words. To achieve this, let's divide the occurrence frequency of each of the words by the frequency of the most recurrent word in the paragraph.

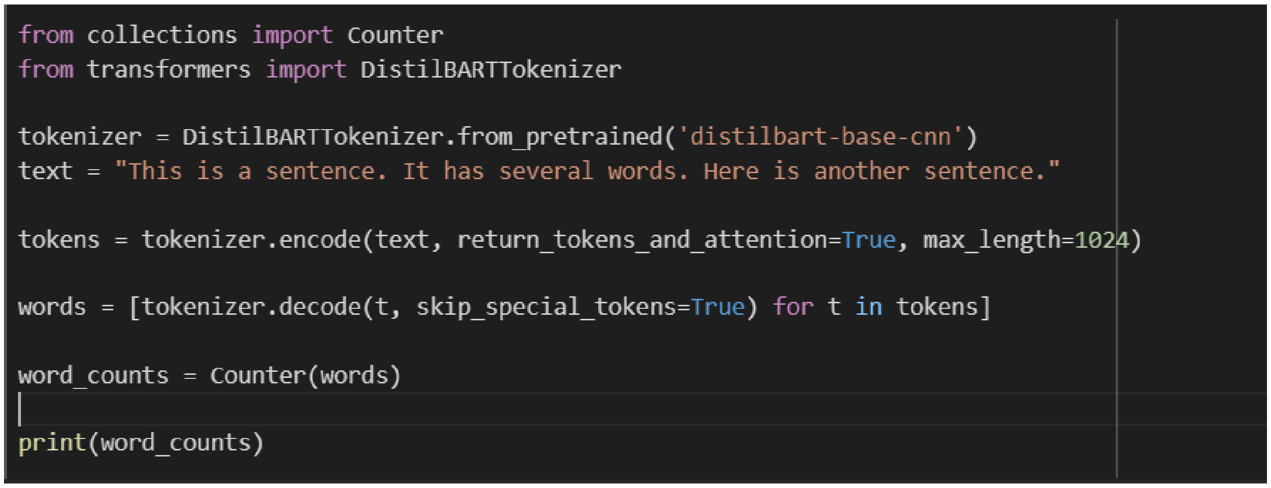
Eg: "Peter" occurs three times. Here is a table that gives the weighted occurrence frequency of each of the words.



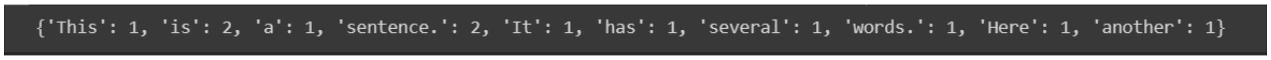
Word	Frequency	Weighted Frequency
peter	3	1
elizabeth	2	0.67
took	1	0.33
taxi	1	0.33
attend	1	0.33

Figure 3

Example:

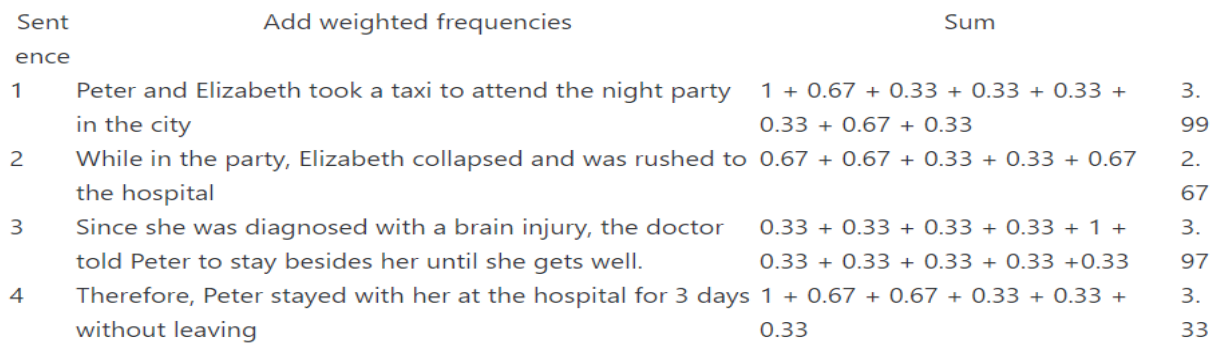


```
from collections import Counter
from transformers import DistilBARTTokenizer

tokenizer = DistilBARTTokenizer.from_pretrained('distilbart-base-cnn')
text = "This is a sentence. It has several words. Here is another sentence."

tokens = tokenizer.encode(text, return_tokens_and_attention=True, max_length=1024)

words = [tokenizer.decode(t, skip_special_tokens=True) for t in tokens]

word_counts = Counter(words)

print(word_counts)
```



```
{'This': 1, 'is': 2, 'a': 1, 'sentence.': 2, 'It': 1, 'has': 1, 'several': 1, 'words.': 1, 'Here': 1, 'another': 1}
```

Step 5. Substitute words with their weighted frequencies: Let's substitute each of the words found in the original sentences with their weighted frequencies. Then, we'll compute their sum.

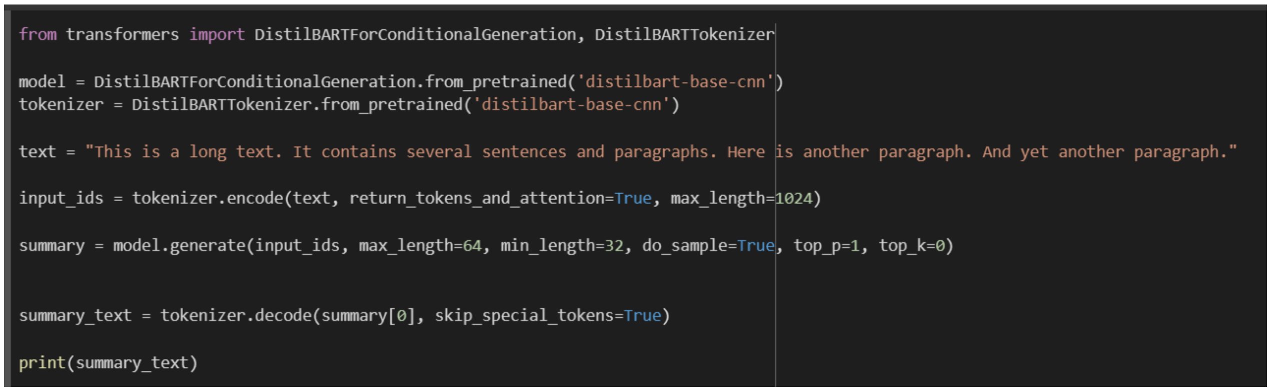


Sentence	Add weighted frequencies	Sum	
1	Peter and Elizabeth took a taxi to attend the night party in the city	1 + 0.67 + 0.33 + 0.33 + 0.33 + 0.33 + 0.67 + 0.33	3.99
2	While in the party, Elizabeth collapsed and was rushed to the hospital	0.67 + 0.67 + 0.33 + 0.33 + 0.67	2.67
3	Since she was diagnosed with a brain injury, the doctor told Peter to stay besides her until she gets well.	0.33 + 0.33 + 0.33 + 0.33 + 1 + 0.33 + 0.33 + 0.33 + 0.33 +0.33	3.97
4	Therefore, Peter stayed with her at the hospital for 3 days without leaving	1 + 0.67 + 0.67 + 0.33 + 0.33 + 0.33	3.33

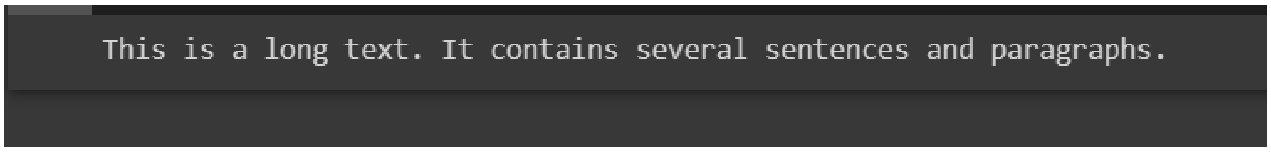
Figure 4

Step 6. Getting the summary: Since we have all the required parameters, we can now generate a summary.

Example:



```python
from transformers import DistilBARTForConditionalGeneration, DistilBARTTokenizer

model = DistilBARTForConditionalGeneration.from_pretrained('distilbart-base-cnn')
tokenizer = DistilBARTTokenizer.from_pretrained('distilbart-base-cnn')

text = "This is a long text. It contains several sentences and paragraphs. Here is another paragraph. And yet another paragraph."

input_ids = tokenizer.encode(text, return_tokens_and_attention=True, max_length=1024)

summary = model.generate(input_ids, max_length=64, min_length=32, do_sample=True, top_p=1, top_k=0)


summary_text = tokenizer.decode(summary[0], skip_special_tokens=True)

print(summary_text)
```



```
This is a long text. It contains several sentences and paragraphs.
```

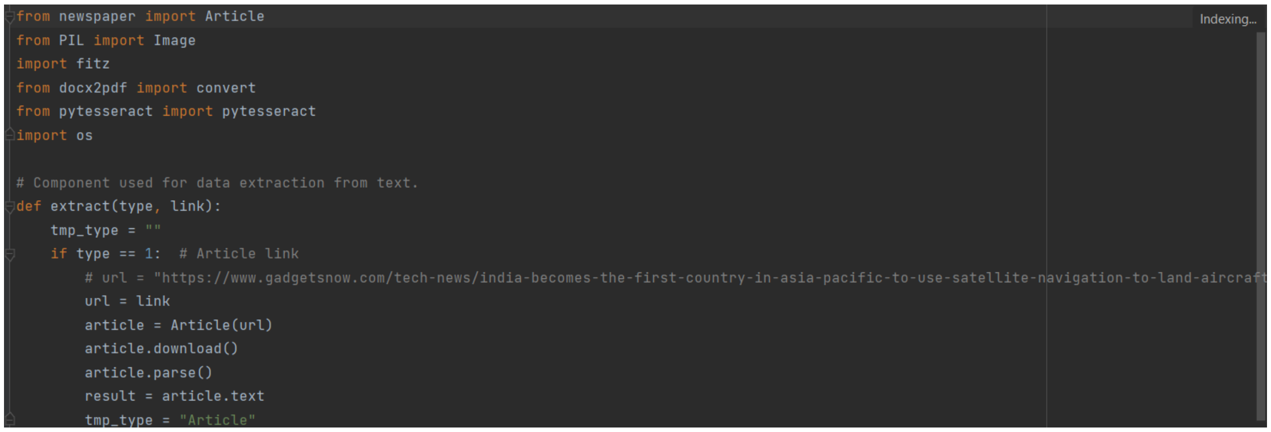
2.4 Frameworks, Libraries & Modules

Various standards used in this project are:

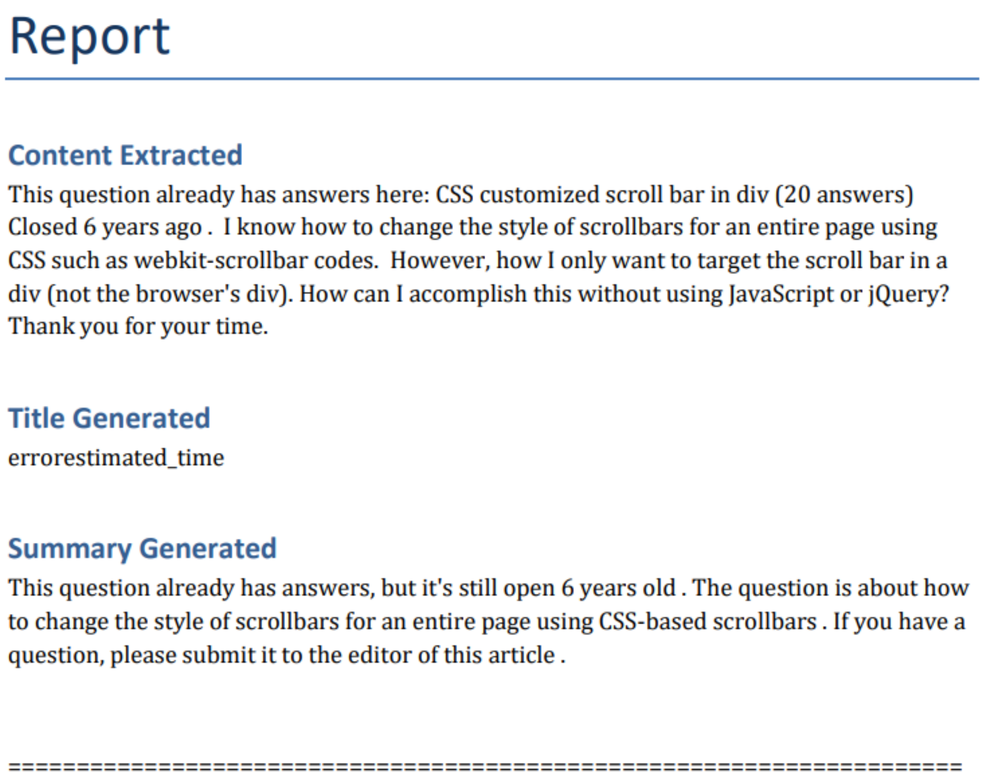
- **Newspaper Library**

The Newspaper library is a Python library for extracting and parsing articles from news websites. It is designed to make it easy to extract the main content from an article, as well as other relevant information such as the title, author, and publication date.

Once you have installed the library, you can use it in your Python code by importing the **Article** class from the **Newspaper** module.



```
from newspaper import Article
from PIL import Image
import fitz
from docx2pdf import convert
from pytesseract import pytesseract
import os

# Component used for data extraction from text.
def extract(type, link):
    tmp_type = ""
    if type == 1:  # Article link
        # url = "https://www.gadgetsnow.com/tech-news/india-becomes-the-first-country-in-asia-pacific-to-use-satellite-navigation-to-land-aircraft
        url = link
        article = Article(url)
        article.download()
        article.parse()
        result = article.text
        tmp_type = "Article"
```



Report

Content Extracted

This question already has answers here: CSS customized scroll bar in div (20 answers) Closed 6 years ago . I know how to change the style of scrollbars for an entire page using CSS such as webkit-scrollbar codes. However, how I only want to target the scroll bar in a div (not the browser's div). How can I accomplish this without using JavaScript or jQuery? Thank you for your time.

Title Generated

errorestimated_time

Summary Generated

This question already has answers, but it's still open 6 years old . The question is about how to change the style of scrollbars for an entire page using CSS-based scrollbars . If you have a question, please submit it to the editor of this article .

==

- **Python Image Library (PIL)**

  The Python Imaging Library (PIL) is a library for working with images in Python. It provides support for reading and writing a wide range of image file formats, as well as basic image manipulation functions such as cropping, resizing, and rotating image.

  Once you have installed the library, you can use it in your Python code by importing the **Image** module.

- **Pytesseract**

  PyTesseract is a Python wrapper for the Tesseract OCR (Optical Character Recognition) engine. It allows you to recognize text in images and convert it into machine-readable text using the Tesseract OCR engine.

  You will also need to install the Tesseract OCR engine itself, which you can do by following the instructions at https://github.com/tesseract-ocr/tesseract.Once you have installed PyTesseract and Tesseract OCR, you can use them in your Python code by importing the **Pytesseract** module and calling the **Image_to_string** function.

- **Fitz**

  The **fitz** library is a Python wrapper for the MuPDF library, which is a cross-platform PDF rendering and editing library. The **fitz** library provides a set of Python bindings for working with PDF files, allowing you to read, write, and manipulate PDF documents in your Python code.

  Once you have installed the library, you can use it in your Python code by importing the **fitz** module.

- **Flask Framework**

  Flask is a lightweight Python web framework that allows you to build web applications quickly. It is designed to be easy to use and flexible and provides a range of features and tools that make it suitable for a wide variety of applications.

  Once you have installed Flask, you can use it in your Python code by importing the **Flask** class from the flask module.

- **ReactJS**

  The React. js framework is an open-source JavaScript framework and library developed by Facebook. It's used for building interactive user interfaces and web applications quickly and efficiently with significantly less code than you would with vanilla JavaScript.

- **Axios**

  Axios is a promise-based HTTP library that lets developers make requests to either their own or a third-party server to fetch data. It offers different ways of making requests such as GET, POST, PUT/PATCH, and DELETE.

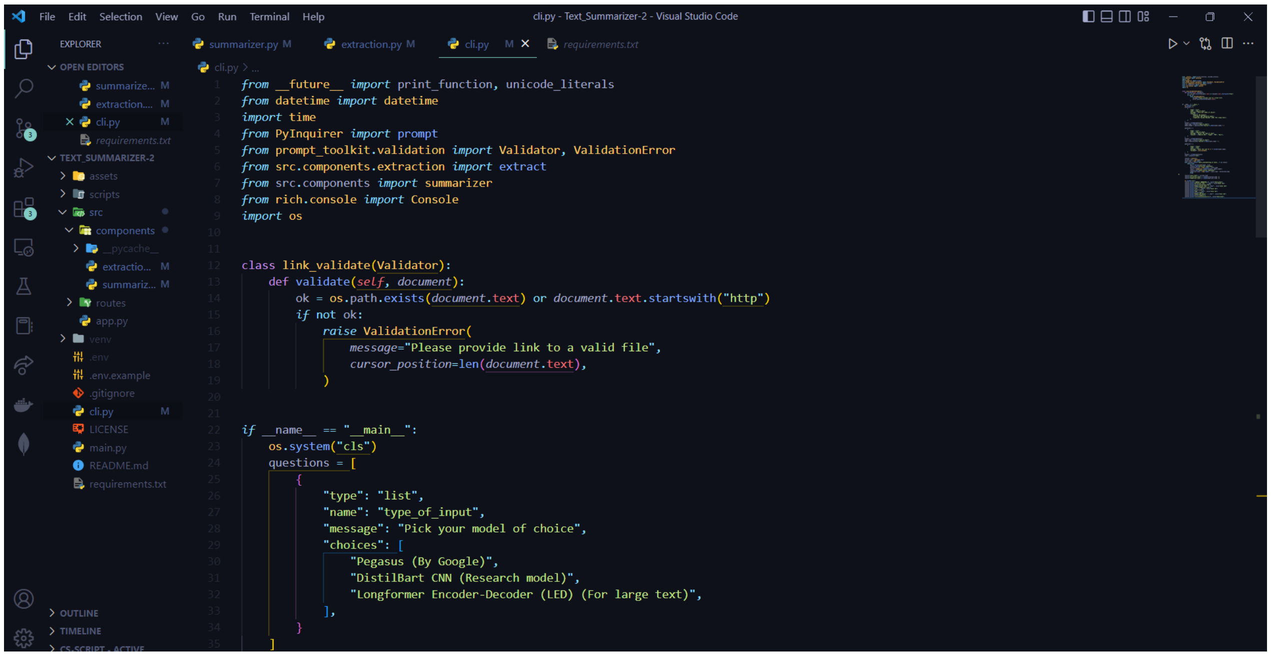
2.5 System Details

This section describes the software details of the system.

2.5.1 Software Details

Visual Studio Code:

Visual Studio Code (VS Code) is a free and open-source code editor. It includes support for debugging, source control, and code completion, and includes a range of built-in tools for writing and testing code in a variety of programming languages. It is often used by developers for tasks such as writing, debugging, and testing code, as well as for creating and editing code in a variety of programming languages.Vscode has been used to build up the entire project.



```python
from __future__ import print_function, unicode_literals
from datetime import datetime
import time
from PyInquirer import prompt
from prompt_toolkit.validation import Validator, ValidationError
from src.components.extraction import extract
from src.components import summarizer
from rich.console import Console
import os


class link_validate(Validator):
    def validate(self, document):
        ok = os.path.exists(document.text) or document.text.startswith("http")
        if not ok:
            raise ValidationError(
                message="Please provide link to a valid file",
                cursor_position=len(document.text),
            )


if __name__ == "__main__":
    os.system("cls")
    questions = [
        {
            "type": "list",
            "name": "type_of_input",
            "message": "Pick your model of choice",
            "choices": [
                "Pegasus (By Google)",
                "DistilBart CNN (Research model)",
                "Longformer Encoder-Decoder (LED) (For large text)",
            ],
        }
    ]
```

Figure 5

2.6 Tools/Models

DistilBart CNN:

DistilBART CNN is a machine learning model developed by researchers at Hugging Face for abstractive text summarization. DistilBART is based on the BART (Denoising Autoencoder for Abstractive and Representative Translation) model, which was developed by researchers at Facebook AI. DistilBART is a smaller, faster, and more lightweight version of BART that is easier to train and deploy. It uses a combination of a denoising autoencoder and a transformer architecture to generate summaries by synthesizing and paraphrasing the content of a text. DistilBART has been shown to achieve strong performance on a variety of natural language processing tasks, including text summarization.

One of the advantages of DistilBart CNN over other models is its ability to handle long input sequences. In comparison, other models such as LSTM (Long Short-Term Memory) and GRU (Gated Recurrent Unit) can struggle with processing long sequences, leading to reduced performance.

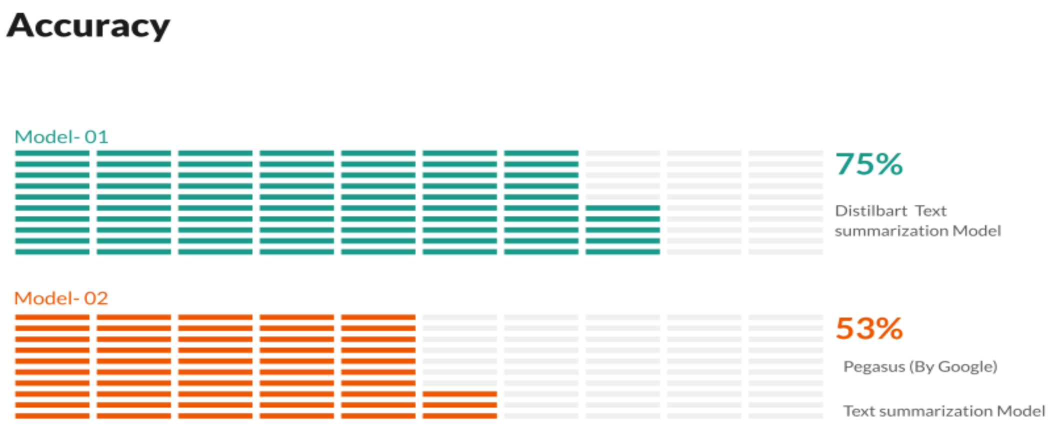
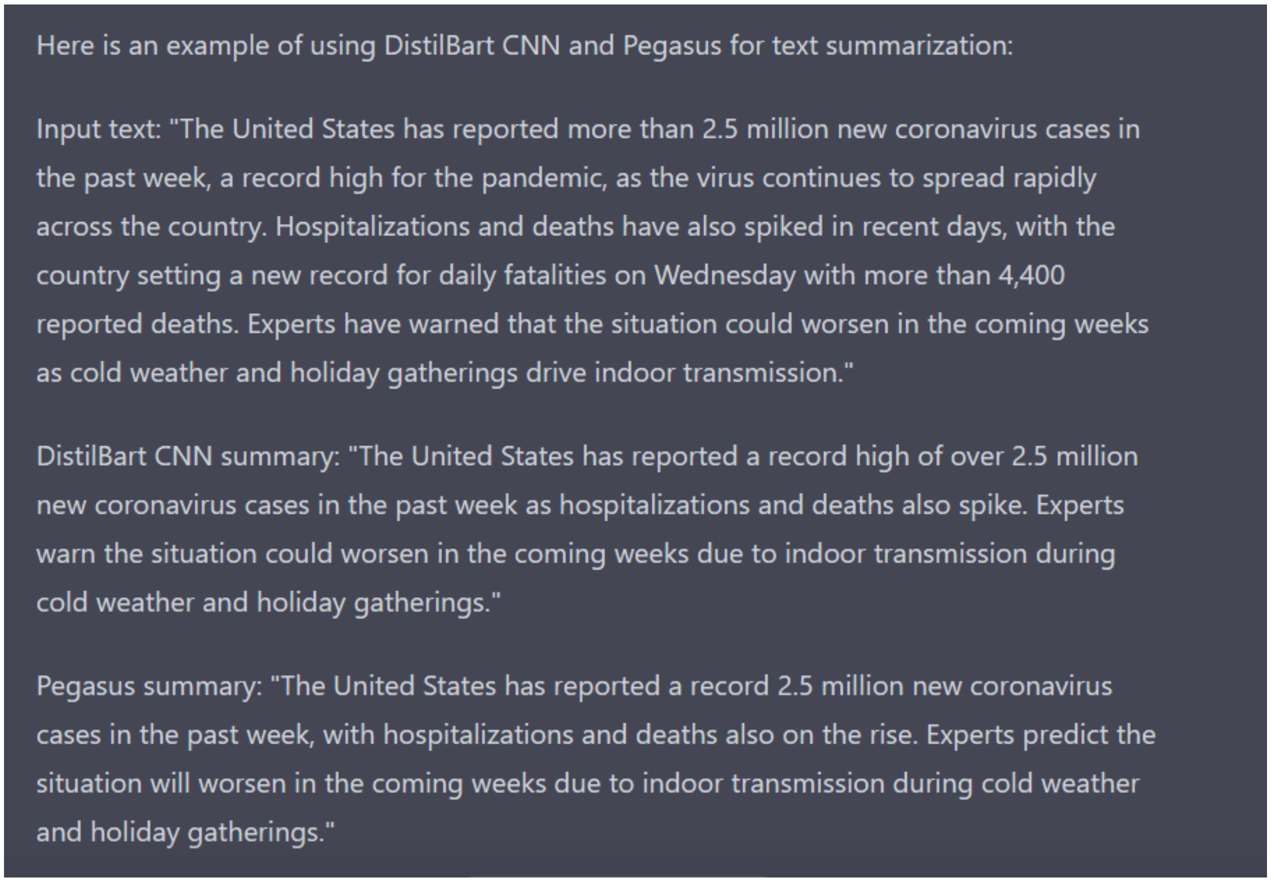


Figure 6



Here is an example of using DistilBart CNN and Pegasus for text summarization:

Input text: "The United States has reported more than 2.5 million new coronavirus cases in the past week, a record high for the pandemic, as the virus continues to spread rapidly across the country. Hospitalizations and deaths have also spiked in recent days, with the country setting a new record for daily fatalities on Wednesday with more than 4,400 reported deaths. Experts have warned that the situation could worsen in the coming weeks as cold weather and holiday gatherings drive indoor transmission."

DistilBart CNN summary: "The United States has reported a record high of over 2.5 million new coronavirus cases in the past week as hospitalizations and deaths also spike. Experts warn the situation could worsen in the coming weeks due to indoor transmission during cold weather and holiday gatherings."

Pegasus summary: "The United States has reported a record 2.5 million new coronavirus cases in the past week, with hospitalizations and deaths also on the rise. Experts predict the situation will worsen in the coming weeks due to indoor transmission during cold weather and holiday gatherings."

CHAPTER 3: IMPLICATIONS AND LIMITATIONS

- Gist's reliance on machine learning techniques, which are subject to bias and inaccuracies, is another drawback. The model may produce biased or erroneous summaries if the training data contains information that is wrong or biased. In order to guarantee that the model generates accurate and impartial summaries, it is crucial to carefully analyze the sources and quality of the training data.
- Gist's reliance on natural language processing (NLP) techniques, which can be difficult to set up and maintain, is another drawback. In addition to requiring specific knowledge to create and optimize, NLP models may also need constant upgrades and upkeep to maintain their effectiveness.
- Gist is just one of several methods available for text summary; as such, it may not always be the best or most suitable option for a particular purpose. When choosing a text summary tool, it is crucial to carefully analyze the unique demands and constraints of a given application. Other methods, like extractive text summarization, may be more appropriate in specific circumstances.

3.1 Recommendations for Future Work

There are several directions for future work on gist.

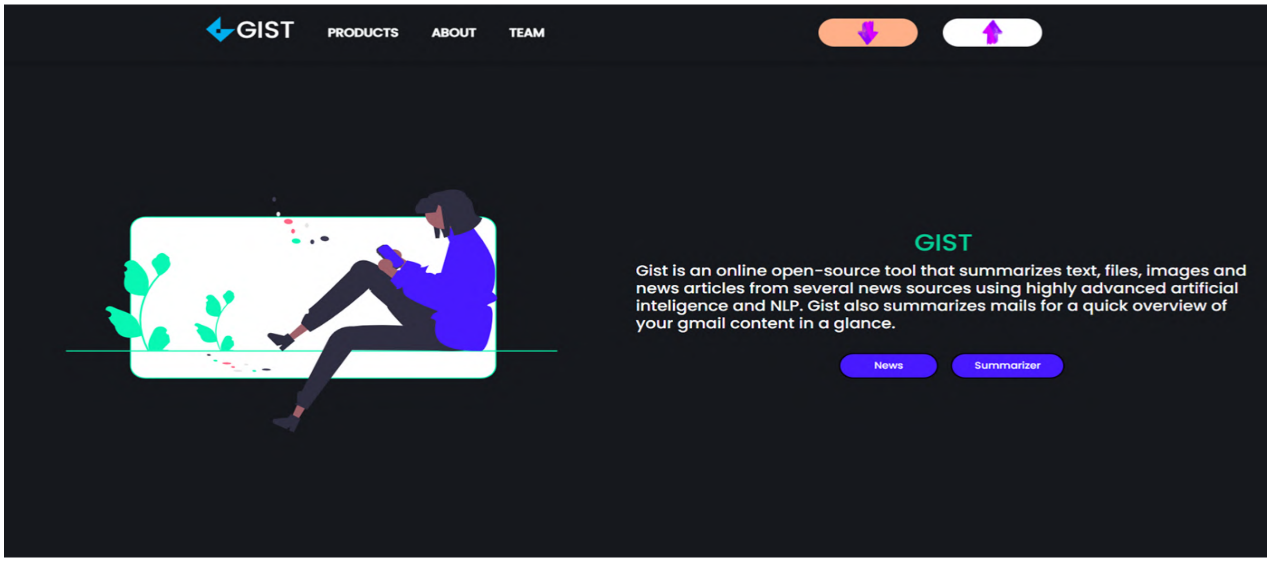
- Including more characteristics or context can assist the model to comprehend the meaning and significance of the text, thereby improving performance and enabling the generation of longer summaries. Examples of additional features or context include source credibility or sentiment analysis.
- The neural network architecture could be improved by including more layers or units. This could result in better performance and the capacity to produce lengthier summaries as the model is able to capture more intricate patterns and dependencies in the data.
- Using more sophisticated optimization methods: The model can learn more effectively and efficiently by using methods like gradient descent or

stochastic gradient descent, which may improve performance and allow it to produce longer summaries.

- Larger and more diversified training datasets can aid in the model's learning of more reliable and generalizable patterns and dependencies, which could result in enhanced performance and the capacity to produce lengthier summaries.
- The model's hyperparameters can be fine-tuned to increase performance and the capacity to provide lengthier summaries. Examples of such hyperparameters are the learning rate and the size of the hidden layers, which can be changed to optimize the model's effectiveness.
- We can add audio summaries by differentiating the voices of people. This will be useful for students who have online classes. Also If we integrate the dataset of law cases we can summarize a court session and give solutions based on previous data.
- Recommendation system for Gist based on location, view activity, and other related clusters of information.

CHAPTER 4: RESULTS AND DISCUSSIONS

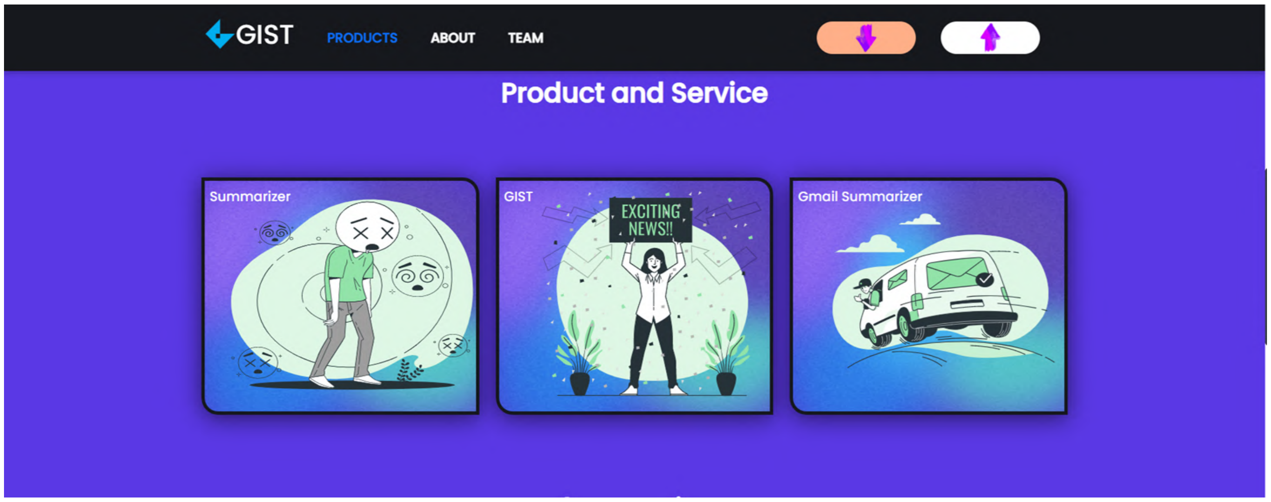
Dashboard



Gist is an online open-source tool that summarizes text, files, images and news articles from several news sources using highly advanced artificial inteligence and NLP. Gist also summarizes mails for a quick overview of your gmail content in a glance.

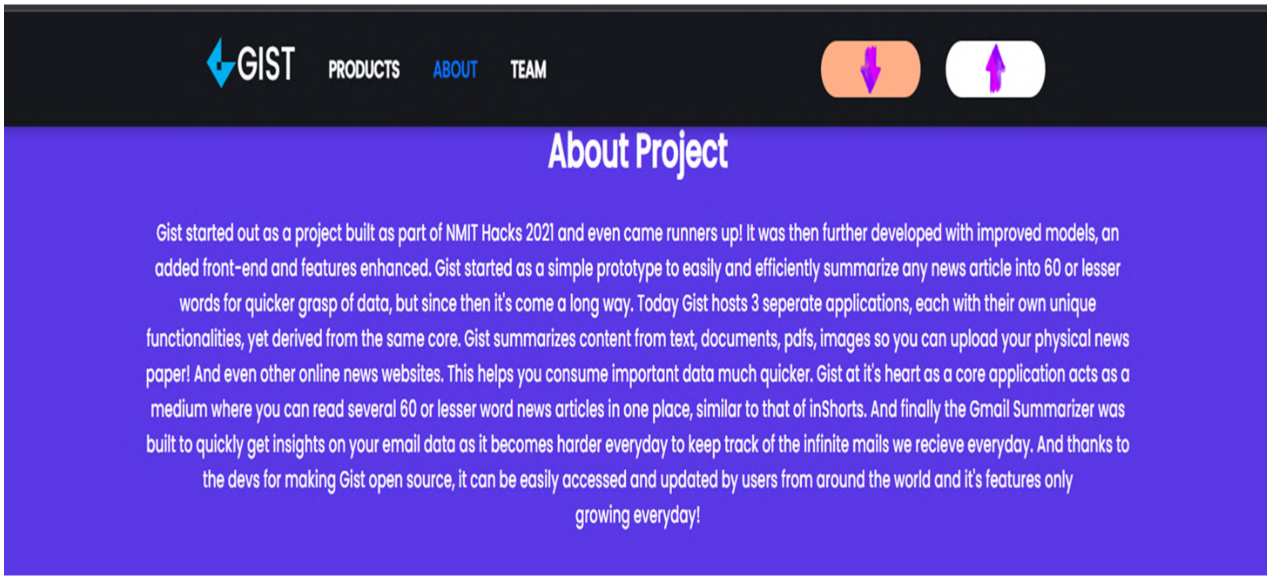
Output-1

Products



Output-2

About

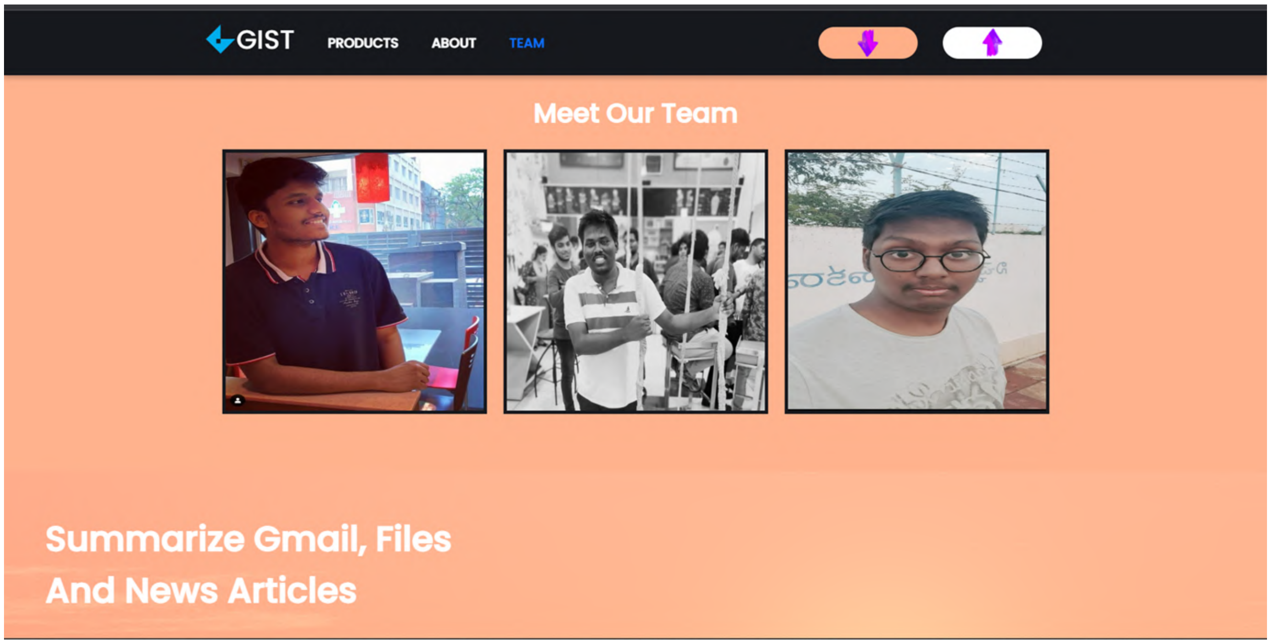


About Project

Gist started out as a project built as part of NMIT Hacks 2021 and even came runners up! It was then further developed with improved models, an added front-end and features enhanced. Gist started as a simple prototype to easily and efficiently summarize any news article into 60 or lesser words for quicker grasp of data, but since then it's come a long way. Today Gist hosts 3 seperate applications, each with their own unique functionalities, yet derived from the same core. Gist summarizes content from text, documents, pdfs, images so you can upload your physical news paper! And even other online news websites. This helps you consume important data much quicker. Gist at it's heart as a core application acts as a medium where you can read several 60 or lesser word news articles in one place, similar to that of inShorts. And finally the Gmail Summarizer was built to quickly get insights on your email data as it becomes harder everyday to keep track of the infinite mails we recieve everyday. And thanks to the devs for making Gist open source, it can be easily accessed and updated by users from around the world and it's features only growing everyday!

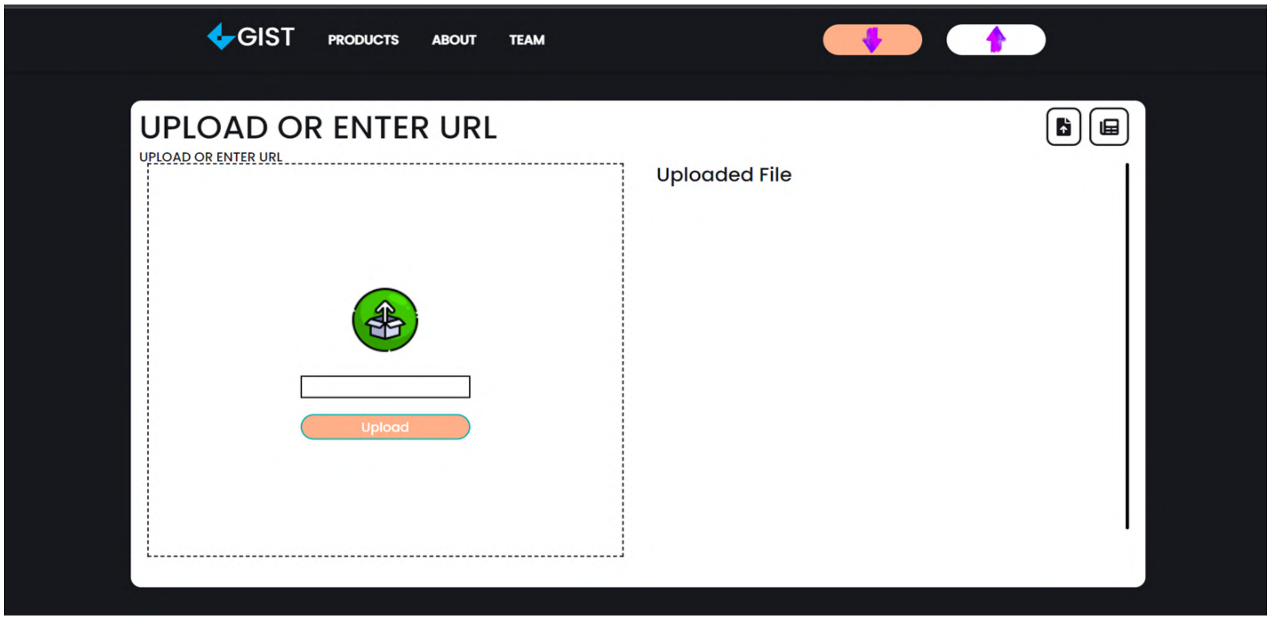
Output-3

Team

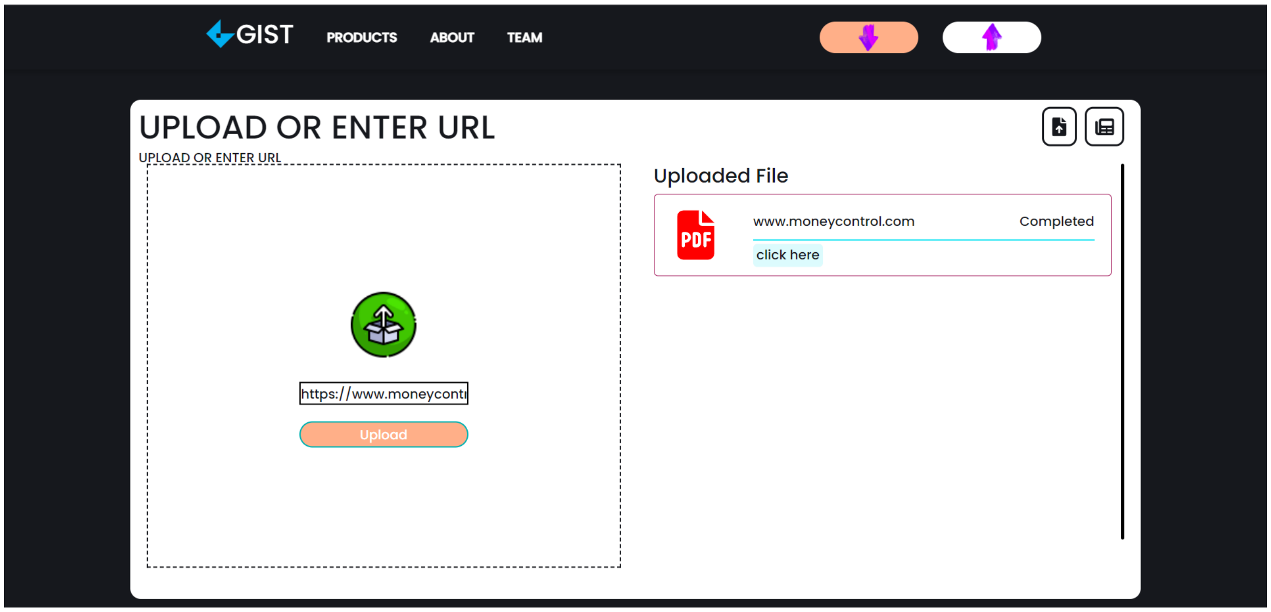


Output-4

Summarizer

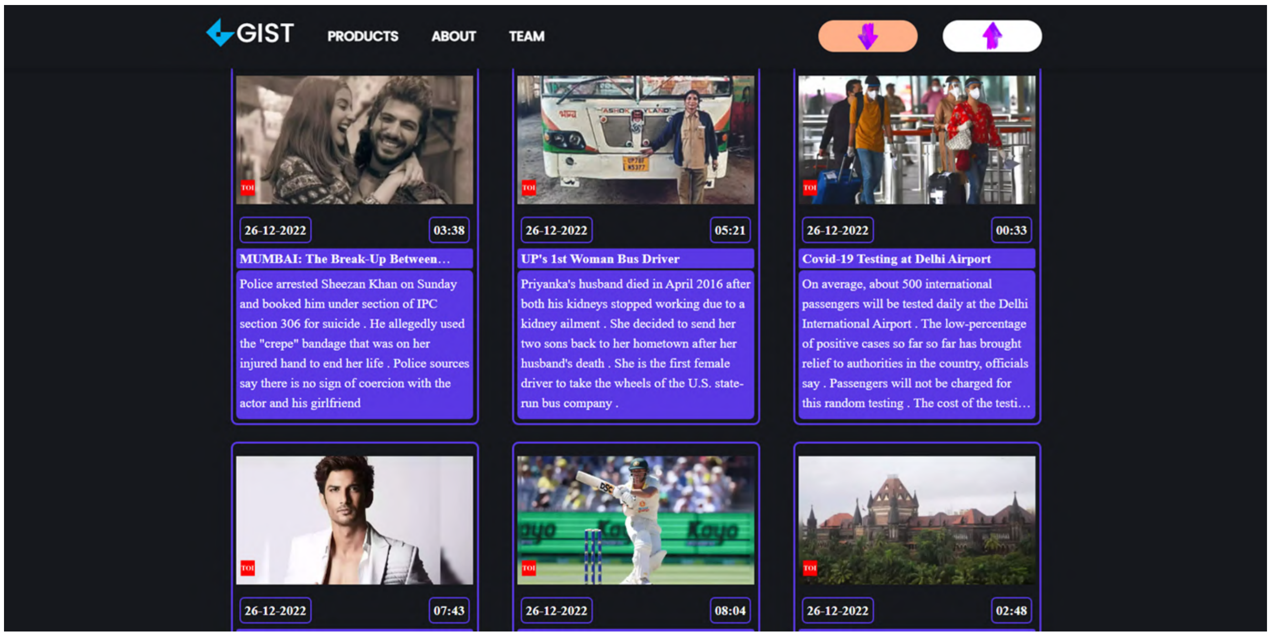


Output-5



Output-6

Gist-News



26-12-2022 03:38

MUMBAI: The Break-Up Between…

Police arrested Sheezan Khan on Sunday and booked him under section of IPC section 306 for suicide . He allegedly used the "crepe" bandage that was on her injured hand to end her life . Police sources say there is no sign of coercion with the actor and his girlfriend

26-12-2022 05:21

UP's 1st Woman Bus Driver

Priyanka's husband died in April 2016 after both his kidneys stopped working due to a kidney ailment . She decided to send her two sons back to her hometown after her husband's death . She is the first female driver to take the wheels of the U.S. state-run bus company .

26-12-2022 00:33

Covid-19 Testing at Delhi Airport

On average, about 500 international passengers will be tested daily at the Delhi International Airport . The low-percentage of positive cases so far so far has brought relief to authorities in the country, officials say . Passengers will not be charged for this random testing . The cost of the testi…

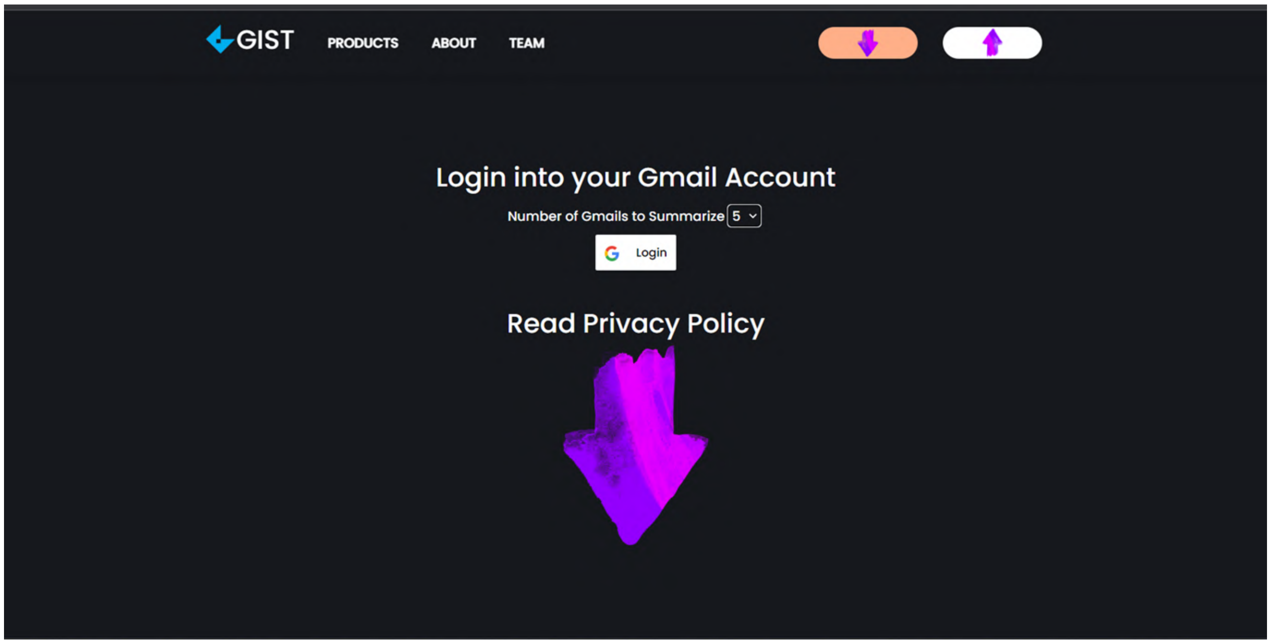
26-12-2022 07:43

26-12-2022 08:04

26-12-2022 02:48

Output-7

Gmail-Summarization



Login into your Gmail Account

Number of Gmails to Summarize 5

Read Privacy Policy

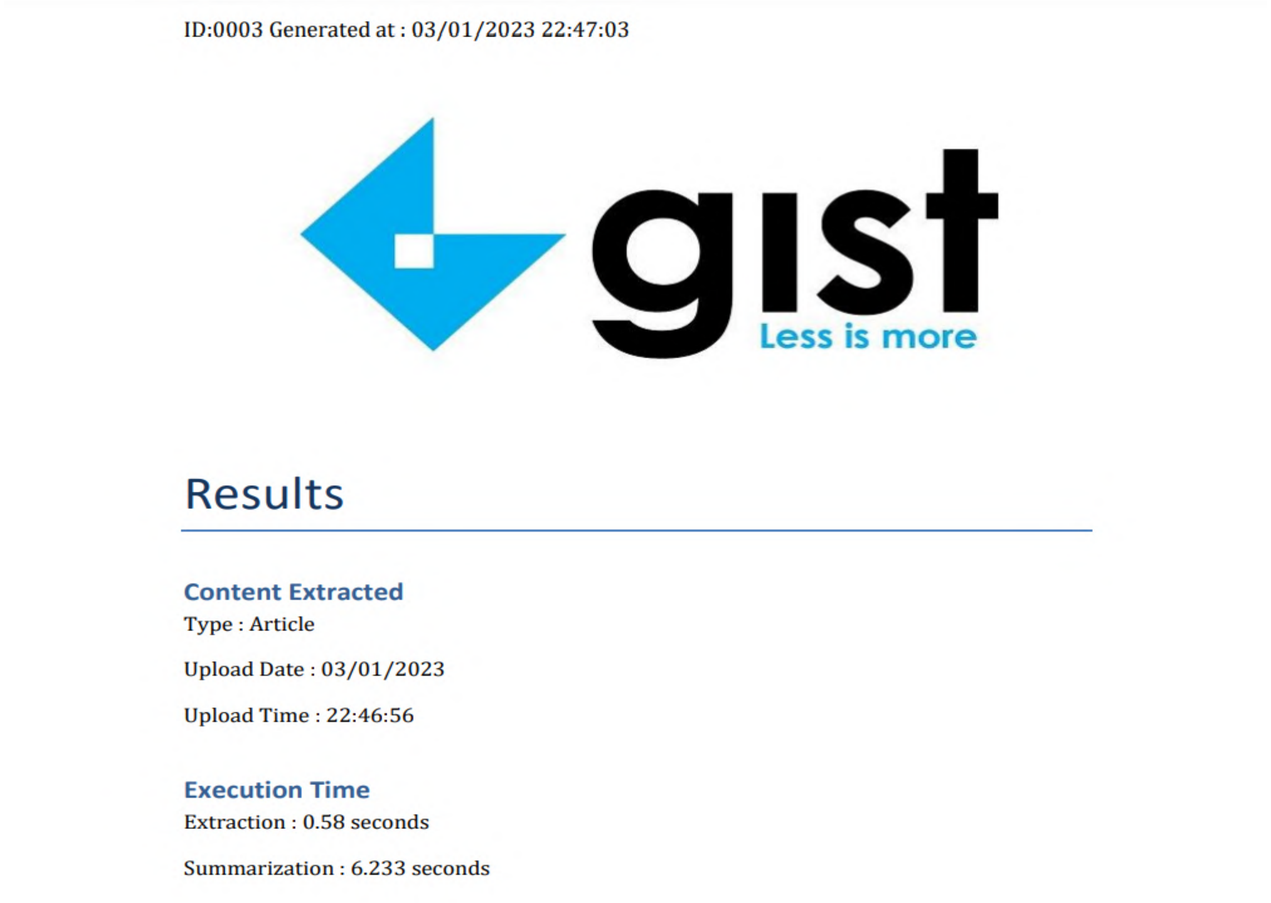
Output-8

Results

First, the document needs to be cleaned up by getting rid of all the stop words. Then, by comparing a subset of terms with each other in the remaining text file, determine the frequency of each word. The most often occurring keyword is then chosen. then choose the sentences that contain those words.

With this method, we first eliminate terms that are often used before finding keywords based on how often the word appears. This makes the assumption that if a passage is provided, more attention will be given to the subject matter, increasing the frequency of the word and words related to it. Since the other sentences wouldn't be as pertinent to the topic as the ones containing the keywords would be, we must now remove those lines in which these terms appear. As a result, a summary that only includes useful sentences is produced.

Article Summary Report:-



ID:0003 Generated at : 03/01/2023 22:47:03

gist
Less is more

Results

Content Extracted
Type : Article

Upload Date : 03/01/2023

Upload Time : 22:46:56

Execution Time
Extraction : 0.58 seconds

Summarization : 6.233 seconds

ID:0001 Generated at : 24/12/2022 17:47:57

Report

Content Extracted

HYDERABAD: Simultaneous searches were launched by Telangana police at seven places across four states on Saturday in connection with Poachgate , which allegedly involved big money to engineer defections.The special investigation team (SIT) probing the alleged scandal, claimed to have seized incriminating evidence during searches till last reports came in.Sources said, seven teams comprising 80 policemen drawn from Cyberabad, Hyderabad and other police units, swooped down on houses of the three arrested accused — Faridabad-based religious preacher Ramachandra Bharathi , Hyderabad-based businessman Nanda Kumar and Simhayaji Swamy of Tirupati. Dr Jaggu's residence at Kochi in Kerala was also searched.The searches were conducted at Bharathi's houses at Faridabad in Haryana and Puttur in Karnataka and Simhayaji Swamy's residence in Tirupati. SIT personnel also raided Hyderabad businessman Nanda Kumar's houses and restaurant at Jubilee Hills locality."We conducted searches at seven places in Haryana, Kerala, Karnataka and Telangana," said a SIT official.Sources said, Dr Jaggu is the bridge between accused Ramachandra Bharathi and another suspect Tushar, who's yet to be nabbed. "Tushar spoke to MLA Rohith Reddy on phone. The Kerala doctor appears to be close to Tushar and we need to verify evidence collected by teams on activities of each accused," said an official involved in the probe.Police also claimed to have evidence that points to a legislator's relative, Srinivas, who booked Simhayaji Swamy's flight ticket to Hyderabad from Tirupati. Police are likely to add more people as accused in the case once special teams return with evidence from other states.

Title Generated

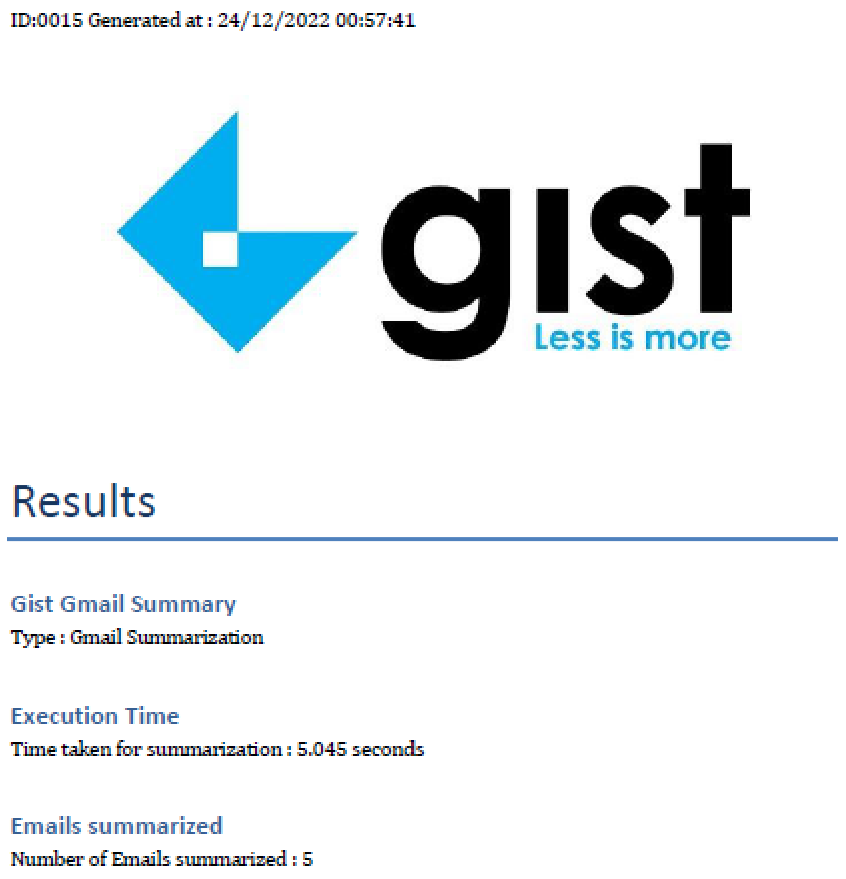
Telangana Police launched simultaneous searches in connection with Poachgate

Summary Generated

Searches were launched by Telangana police in connection with the alleged corruption scandal . They were conducted at seven locations in Haryana, Kerala, Karnataka and Hyderabad . Police also raided homes of a religious preacher and a businessman in the city of Tirupati

==

Gmail-Summary Report:-



ID:0015 Generated at : 24/12/2022 00:57:41

gist
Less is more

Results

Gist Gmail Summary
Type : Gmail Summarization

Execution Time
Time taken for summarization : 5.045 seconds

Emails summarized
Number of Emails summarized : 5

ID:0015 Generated at : 24/12/2022 00:57:41

Report

Email ID : 1

Subject : Silver linings: These emerging tech sectors braved 2022's storms

Email From : "Wellfound (fka AngelList Talent)" <newsletters@marketing.angel.co>

Recieved at : Fri, 23 Dec 2022 18:37:30

Summary Generated :

The healthtech sector saw better over all deal values this year than most this year . Healthtech is a hot category among young tech workers . Tech companies are hiring to extend life expectancy and prevent disease . Climate tech will be a hot trend in the U.S. market for the next few years

==

Email ID : 2

Subject : Google Pixel roadmap leaks 📱 , first automated Mcdonalds 🍔, TikTok spies on journalists 🕵️

Email From : "TLDR 😎 " <dan@tldrnewsletter.com>

Recieved at : Fri, 23 Dec 2022 11:43:22

Summary Generated :

Google will take with its Pixel series in 2023, 2024, and 25 years later . The Pixel series is set to launch in 2015 . The series will be on sale at Google Play, Google Play.com and Google Play for $1.2 million each . For more information on Google's Pixel series click here .

==

Email ID : 3

Subject : A Software Developer's Guide to Writing

Email From : Hashnode Weekly <digest@mail.hashnode.co>

ID:0015 Generated at : 24/12/2022 00:57:41

Recieved at : Thu, 22 Dec 2022 16:19:06

Summary Generated :

Your Hashnode+Weekly's weekly feature is a great chance to reflect on your journey as a developer . It's also a great opportunity to connect with thousands of other developers who have participated in the campaign . A huge shout-out-out is out to those who have contributed to the community .

==

Email ID : 4

Subject : Google's ChatGPT Code Red 🚨 , Twitter's financials 💰 , Custom Diffusion AI 🖼️

Email From : "TLDR 🤖" <dan@tldrnewsletter.com>

Recieved at : Thu, 22 Dec 2022 11:41:05

Summary Generated :

Chat+bots like ChatGPT have led Google management to declare a code-red red alert . Google already has a chat-bot that could rival Chatbot, called LaMDA . Chatbots are also known to be known to generate false information and be biased, biased information

==

Email ID : 5

Subject : Google in-video Search 🔍 , OpenAI's 3D art AI 🎨 , WordPress launches SQLite beta 🧑‍💻

Email From : "TLDR 🤖" <dan@tldrnewsletter.com>

Recieved at : Wed, 21 Dec 2022 11:43:52

Summary Generated :

Drata is a self-advised expert in compliance and security software . Drata can help you get on a compliance journey from start-to-audit-ready to audit-ready . The company's software can be used to help you make the most of your compliance journey .

ID:0015 Generated at : 24/12/2022 00:57:41

==

CHAPTER 5: CONCLUSION AND FUTURE WORK

- In this paper, the abstractive text summarization methodology outlined in this study has a number of room for improvement and growth. One option is to expand the model's compatibility with more languages and topic areas. Another area of emphasis might be to add more elements to the generated summaries in order to improve their relevance and accuracy, such as sentiment analysis or source credibility. Additionally, ongoing research can concentrate on enhancing the model's effectiveness and performance, for instance by utilizing more sophisticated optimization techniques or including extra layers or units in the neural network architecture.

- We present a paradigm for abstractive text summarization that can produce clear and comprehensive summaries of lengthy texts. The model proved successful in producing high-quality summaries that faithfully captured the major ideas of the original text after being trained on a sizable dataset of news items. ROUGE scores and human evaluations of the model's performance demonstrated its value as a tool for swiftly and effectively understanding vast amounts of information.

- To extract significant sentences from the inputs in our system, we utilise statistical and heuristic methods. Additionally, it uses conceptual relationships to compute similarity and rank sentences. Our method, which is the most similar to human summaries among those that are now available, performs the best, according to our experiments.

- The score of the suggested method can be raised by giving the handling of input files, such as newspaper articles or tales, considerable thought. Our goals for our future work on the system improvement include moving toward automatic evaluation of summarizing, eliminating repetitions within a phrase, eliminating gaps in the summary, improving the coherence of the text, and more.

- Overall, by increasing the efficacy and efficiency of information retrieval and consumption, the development of efficient abstractive text summarization models has the potential to significantly benefit society. The ability to read and understand huge amounts of text fast and properly can be quite valuable in a society where time is often scarce and information is plentiful.

CHAPTER 6: APPENDIX

Python: Flask API setup

```python
from flask.templating import render_template

from flask_cors import CORS, cross_origin

from flask import Flask, request, jsonify, send_from_directory

from src.routes import api_blueprint

import os


def create_app():

    app = Flask(__name__, instance_relative_config=True)

    app.config["SECRET_KEY"] = os.getenv("SECRET_KEY")

    print(os.getenv("SECRET_KEY"))

    app.url_map.strict_slashes = False

    api_cors_config = {

        "origins": [

            "*",

            "http://localhost:3000",

        ]

    }

    CORS(app, resources={"/*": api_cors_config})

    @app.route("/", methods=["GET"])

    def index():

        return "<h3>API running successfully!<h3>"

    @app.route("/favicon.ico")

    def favicon():

        return send_from_directory(
            os.path.join(app.root_path, "../assets"),
            "favicon.ico",
            mimetype="image/vnd.microsoft.icon",
        )
    @app.errorhandler(404)
    def page_not_found(e):
        return "ERROR 404: CANNOT GET {}".format(request.path)
    app.register_blueprint(api_blueprint)
    return app
```

Registering blueprints for the API

```
from flask import Blueprint
from src.routes.extraction import extraction_api
from src.routes.summarizer import summarize_api
from src.routes.categorizer import categorizer_api
from src.routes.gist import gist_main_api, gist_data_api
from src.routes.gmail import gmail_api

api_blueprint = Blueprint("API", __name__, url_prefix="/api/")
api_blueprint.register_blueprint(extraction_api.extract_bp)
api_blueprint.register_blueprint(summarize_api.summarize_bp)
api_blueprint.register_blueprint(categorizer_api.category_bp)
api_blueprint.register_blueprint(gmail_api.gmail_bp)
api_blueprint.register_blueprint(gist_main_api.gist_bp)
api_blueprint.register_blueprint(gist_data_api.gist_data_bp)


@api_blueprint.route("/", methods=["GET"])
def get_data():
    return "Homepage route setup!"
```

Data extraction

```
from newspaper import Article
from PIL import Image
import fitz
from docx2pdf import convert
from pytesseract import pytesseract
import os
import pythoncom
import win32com.client

def extract(type, link):
    tmp_type = ""
    if type == 1:  # Article link

        url = link
        article = Article(url)
        article.download()
        article.parse()
        result = article.text
        tmp_type = "Article"
    elif type == 2:  # Image taken
        path_to_tesseract = r"C:\Program
Files\Tesseract-OCR\tesseract.exe"
        # image_path = r"assets\footercredits.png"
        image_path = link
        img = Image.open(image_path)
        pytesseract.tesseract_cmd = path_to_tesseract
        text = pytesseract.image_to_string(img)
        result = text[:-1]
        tmp_type = "Image"
    elif type == 3:  # PDF file
        # pdfFileObj = open("example.pdf", "rb")
        fileObj = fitz.open(link)
        result = ""
        for page in fileObj:
            result += page.get_text() + chr(12)
        tmp_type = "PDF"
        result = result.encode("ascii", "ignore")
        result = result.decode()
        result = result.replace("\x92", "")
        result = result.replace("\x0c", "")
        tmp_type = "PDF"
    elif type == 4:  # Document file
        xl = win32com.client.Dispatch("Word.Application",
pythoncom.CoInitialize())
        convert(link, output_path="temp/output.pdf")
        tmp_type = "Document"
        link = "temp/output.pdf"
        fileObj = fitz.open(link)
        result = ""
        for page in fileObj:
            result += page.get_text() + chr(12)
        tmp_type = "PDF"
        result = result.encode("ascii", "ignore")
        result = result.decode()
        result = result.replace("\x92", "")
        result = result.replace("\x0c", "")
        fileObj.close()
        os.remove("temp/output.pdf")
    else:
        return {"error": "Invalid type"}
    result = " ".join(result.splitlines())
    return {"type": tmp_type, "link": link, "content": result}
```

Core summarizer engine

```
from flask import Flask, jsonify, request, Blueprint, send_from_directory
from werkzeug.utils import secure_filename
from src.routes.extraction.file_extraction import extract_from_file,
extract_from_docx
from src.components.extraction import extract
from src.components import summarizer
from src.components.result_generation import result_generation
from datetime import datetime
import time
import os
from scripts.hashcal import hash_file
```

```
summarize_bp = Blueprint("summarize", __name__, url_prefix="/summarize")


@summarize_bp.route("/", methods=["GET"])
def summarize():
    type = request.args.get("type")
    link = request.args.get("link")
    report = request.args.get("report")
    return pdf_generator(type, link, report)


@summarize_bp.route("/file", methods=["POST"])
def file():
    f = request.files.get("FILE")
    report = request.args.get("report")
    filename = f.filename
    file_extn = filename.split(".")[1]
    f.stream.seek(0)
    f.save(os.path.join("temp_files", filename))
    filename = "temp_files/" + filename
    if file_extn == "png" or file_extn == "jpg" or file_extn == "jpeg":
        type = 2
    elif file_extn == "pdf":
        type = 3
    elif file_extn == "docx" or file_extn == "doc":
        type = 4
    return pdf_generator(type, filename, report)


def pdf_generator(type, filename, report):
    current_time = datetime.now()
    start_time = time.time()
    data = extract(int(type), filename)
    extraction_time = time.time() - start_time
    result = summarizer.summarize(data)
    summarizer_time = time.time() - start_time - extraction_time
    result["start_time"] = current_time
    result["extraction_time"] = round(extraction_time, 3)
    result["summarizer_time"] = round(summarizer_time, 3)
    if report != None and report == str(1):
        filepath = os.path.abspath(os.getcwd()) + "/temp"
        pdf_result = result_generation(result)
        return send_from_directory(filepath, pdf_result)
    return jsonify(result)
```

Summarizer module

```
import re

from pytz import unicode
from transformers import pipeline

from src.components.title_generation import title_generation


def summarize(data):
    print("[!] Server logs: Summarizer Engine has started")
    text = data["content"]
    to_tokanize = text[:1024]
    summarizer = pipeline("summarization",
model="sshleifer/distilbart-cnn-12-6")
    summarized = summarizer(to_tokanize)
    tmp = " ".join([str(i) for i in summarized])
    tmp = tmp.replace("{", "")
    tmp = tmp.replace("'''", "")
    tmp = tmp.replace(f"{chr(61623)}", "")
    tmp = tmp.replace("\x92", "")
    tmp = tmp.replace("\x0c", "")
    regex_pattern = r"(?<='summary_text': ' )(.*)(?='})"
    try:
        result = re.search(regex_pattern, tmp).group(0)
    except:
        result = tmp
    result = result.encode("ascii", "ignore")
    result = result.decode()
    data["summary"] = result
    print("[!] Server logs: Summarized article")
    data = title_generation(data)
    return data
```

Title Generation module

```
import torch
from transformers import T5ForConditionalGeneration, T5Tokenizer
import re


def title_generation(data):
    print("[!] Server logs: Title generation has started")
    text = data["content"]
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    model = T5ForConditionalGeneration.from_pretrained(
        "Michau/t5-base-en-generate-headline"
    )
    tokenizer =
T5Tokenizer.from_pretrained("Michau/t5-base-en-generate-headline")
    model = model.to(device)
    encoding = tokenizer.encode_plus(text, return_tensors="pt")
    input_ids = encoding["input_ids"].to(device)
    attention_masks = encoding["attention_mask"].to(device)

    beam_outputs = model.generate(
        input_ids=input_ids,
        attention_mask=attention_masks,
        max_length=64,
        num_beams=3,
        early_stopping=True,
    )

    result = tokenizer.decode(beam_outputs[0])
    print("[!] Server logs: Title generation completed")

    regex_pattern = r"(?<=<pad> )(.*)(?=</s>)"
    result = re.search(regex_pattern, result).group(0)
    data["title"] = result
    return data
```

Generating PDF result

```python
from docx import Document
from docx2pdf import convert
from datetime import datetime
import pythoncom
import os
import re
import win32com.client


def result_generation(data):
    print("[!] Server logs: Result Generation started")
    document = Document()

    p = document.add_paragraph()
    p = p.insert_paragraph_before("                    ")
    r = p.add_run()
    r.add_picture("assets/report_logo.jpeg")

    ID = open("assets/ID.txt", "r").read()
    now = datetime.now()
    DateAndTime = now.strftime("%d/%m/%Y %H:%M:%S")

    updated_ID = int(ID) + 1
    update = open("assets/ID.txt", "w")
    update.write(str(updated_ID))
    update.close()

    section = document.sections[0]
    header = section.header
    paragraph = header.paragraphs[0]
    paragraph.text = (
        "ID:"
        + str("{num:0>4}".format(num=str(ID)))
        + " Generated at : "
        + str(DateAndTime)
    )
    paragraph.style = document.styles["Header"]

    document.add_heading("Results", 0)
    document.add_heading("Content Extracted")
    document.add_paragraph("Type : " + data["type"])
    document.add_paragraph(
        "Upload Date : " + str(data["start_time"].strftime("%d/%m/%Y"))
    )
    document.add_paragraph(
        "Upload Time : " + str(data["start_time"].strftime("%H:%M:%S"))
    )

    document.add_heading("Execution Time")
    document.add_paragraph("Extraction : " + str(data["extraction_time"])
+ " seconds")
    document.add_paragraph(
        "Summarization : " + str(data["summarizer_time"]) + " seconds"
    )

    document.add_page_break()

    document.add_heading("Report", 0)
    document.add_heading("Content Extracted", 1)
    document.add_paragraph(data["content"])

    document.add_heading("Title Generated", 1)
    document.add_paragraph(data["title"])

    document.add_heading("Summary Generated", 1)
    document.add_paragraph(data["summary"])
    document.add_paragraph("                              ")
    document.add_paragraph(

"======================================================================"
    )
    xl = win32com.client.Dispatch("Word.Application",
pythoncom.CoInitialize())
    doc_location = "temp\Result_{num:0>4}.docx".format(num=str(ID))
    document.save(doc_location)
    convert(doc_location, output_path="temp")
    os.remove(doc_location)
    return "Result_{num:0>4}.pdf".format(num=str(ID))
```

Gist main API code

```
# Main API for the Gist platform.
import xmltodict
from bs4 import BeautifulSoup
from flask import Flask, jsonify, Blueprint
import json
import requests
from dotenv import load_dotenv
from os import getenv
from pymongo import MongoClient
from lxml import etree

load_dotenv()
gist_bp = Blueprint("Gist", __name__, url_prefix="/gist")
CONNECTION_STRING = getenv("MONGO_CONNECTION_STRING")
db = MongoClient(CONNECTION_STRING)
dbname = db["articles"]
collection_name = dbname["items"]


@gist_bp.route("/", methods=["GET"])
def gist():
    # Component to read which news articles are picked.
    # Component to send the article to the summarizer using summarize_api
(Just call local API using requests)
    # Create MongoDB account and save articles. Format mentioned as below.
    # [{"title": "title of article", "summary": "summary generated",
"url": "url of news article", "date_and_time": "value"}, {repeat}]

    url = "https://timesofindia.indiatimes.com/rssfeedstopstories.cms"
    response = requests.get(url)
    my_dict = xmltodict.parse(response.content)
    data = my_dict["rss"]["channel"]["item"]
    result = []
    for article in data:
        article_url = article["link"]
        reqs = requests.get(article_url)
        soup = BeautifulSoup(reqs.text, "html.parser")
        img_s = soup.find_all("img")
        og_img = soup.find("meta",attrs={"property":"og:image"})
        img_url = og_img.get('content')
        print(img_url)
        gist_data = requests.get(

f"http://127.0.0.1:5000/api/summarize?type=1&link={article_url}"
        ).json()
        article = {
            "title": gist_data["title"],
            "summary": gist_data["summary"],
            "url": article["link"],
            "dateAndTime": article["pubDate"],
            "image": img_url,
        }
        result.append(article)
    collection_name.insert_many(result)
    print("[!] Server log: Database updated successfully!")
    return jsonify({"status": "success", "message": "Gist data added to
database"})
```

Gmail extraction and summarization

```
from __future__ import print_function
import base64
import json
import os
from bs4 import BeautifulSoup
from flask import request, Blueprint, redirect, Response
from google_auth_oauthlib.flow import InstalledAppFlow
from googleapiclient.discovery import build
from .serialzer import HeaderParser
from .gmailResult import GmailSummarizer
from .sendMail import payloadPrep


os.environ["OAUTHLIB_INSECURE_TRANSPORT"] = "1"

gmail_bp = Blueprint("gmail", __name__, url_prefix="/gmail")
SCOPES = [
    "https://www.googleapis.com/auth/userinfo.email",
    "openid",
    "https://www.googleapis.com/auth/gmail.readonly",
    "https://www.googleapis.com/auth/userinfo.profile",
]

CURR_DIR = os.path.dirname(os.path.realpath(__file__))
flow = InstalledAppFlow.from_client_secrets_file(
    CURR_DIR + "\\client_secret.json",
    SCOPES,
    redirect_uri="http://localhost:3000",  # http
)


def decode_base64(data_base64):
    data = data_base64.replace("-", "+").replace("_", "/")
    decoded_data = base64.b64decode(data)
    soup = BeautifulSoup(decoded_data, "lxml")
    text = soup.text
    return text


@gmail_bp.route("/", methods=["GET"])
def get_mail():
    authorization_url, state = flow.authorization_url()
    return redirect(authorization_url)


@gmail_bp.route("/callback", methods=["GET", "POST"])
def user_redirect():
    code = request.get_json().get("code")
    No_of_Emails = request.args.get("NoEmails")
    if int(No_of_Emails) > 15:
        No_of_Emails = 15
    flow.fetch_token(code=code)
    credentials = flow.credentials
    service = build("gmail", "v1", credentials=credentials)
    email_address =
service.users().getProfile(userId="me").execute()["emailAddress"]
    response = (
        service.users().messages().list(maxResults=No_of_Emails,
userId="me").execute()
    )
    try:
        messages = response["messages"]
        responses = []
        for msg in messages:
            gmail = service.users().messages().get(userId="me",
id=msg["id"]).execute()
            payload = gmail["payload"]
            gmail_text = {}
            try:
                meta_data = HeaderParser(payload["headers"])
                raw_data = ""
                if (
                    payload["mimeType"] == "text/html"
                    or payload["mimeType"] == "text/plain"
                ):
                    body = payload["body"]
                    data = body["data"]
                    raw_data = decode_base64(data)
                else:
                    parts = payload["parts"]
                    for part in parts:
                        body = part["body"]
                        data = body["data"]
                        raw_data += decode_base64(data)

                gmail_text["meta"] = meta_data
                gmail_text["content"] = raw_data
                responses.append(gmail_text)
            except Exception as e:
                print(e)
                return "Extracting gmail failed ", 400

        # print(responses)
        print(f"Sending Mail to {email_address}")
        file = GmailSummarizer(responses)
        payloadPrep(email_address, ("temp\\" + file))
        print("[!] Server logs: Mail sent successfully")
        return "Request successful", 200
    except Exception as e:
        print(e)
        return "Bad Request ", 400
```

Gmail result generation

```python
from pprint import pprint
import time
from src.components import summarizer
import json
from docx import Document
from docx2pdf import convert
from datetime import datetime
import pythoncom
import os
import re
import win32com.client


def GmailSummarizer(responses):
    start = time.time()
    for email in responses:
        email["content"] = email["content"].replace("\n", " ")
        email["content"] = email["content"].replace("\r", " ")
        email["content"] = email["content"].replace("\t", " ")
        summarizer.summarize(email, title=False)

    json_object = json.dumps(responses, indent=4)
    with open("sample.json", "w") as outfile:
        outfile.write(json_object)
    # f = open("sample.json")
    # responses = json.load(f)
    # f.close()
    # print("Completed Summarization")
    return pdf_generator(responses, start)


def pdf_generator(responses, start):
    print("[!] Server logs: Result Generation started")
    document = Document()

    p = document.add_paragraph()
    p = p.insert_paragraph_before("                    ")
    r = p.add_run()
    r.add_picture("assets/report_logo.jpeg")
    ID = open("assets/ID.txt", "r").read()
    tmp_time = time.time()
    now = datetime.now()
    DateAndTime = now.strftime("%d/%m/%Y %H:%M:%S")

    updated_ID = int(ID) + 1
    update = open("assets/ID.txt", "w")
    update.write(str(updated_ID))
    update.close()

    section = document.sections[0]
    header = section.header
    paragraph = header.paragraphs[0]
    paragraph.text = (
        "ID:"
        + str("{num:0>4}".format(num=str(ID)))
        + " Generated at : "
        + str(DateAndTime)
    )
    paragraph.style = document.styles["Header"]

    document.add_heading("Results", 0)

    document.add_heading("Gist Gmail Summary")
    document.add_paragraph("Type : " + "Gmail Summarization")

    document.add_heading("Execution Time")
    summarizer_time = tmp_time - start
    document.add_paragraph(
        "Time taken for summarization : " + str(round(summarizer_time, 3))
+ " seconds"
    )

    document.add_heading("Emails summarized")
    document.add_paragraph("Number of Emails summarized : " +
str(len(responses)))

    document.add_page_break()

    document.add_heading("Report", 0)

    for data in responses:
        document.add_heading("Email ID : " + str(responses.index(data) +
1), 1)
        document.add_paragraph("Subject : " + data["meta"]["Subject"])
        document.add_paragraph("Email From : " + data["meta"]["from"])
        document.add_paragraph("Recieved at : " +
str(data["meta"]["Date"])[:-6])

        document.add_paragraph("Summary Generated : ")
        document.add_paragraph(data["summary"])
        document.add_paragraph("                              ")
        document.add_paragraph(

"======================================================================"
        )
    xl = win32com.client.Dispatch("Word.Application",
pythoncom.CoInitialize())
    doc_location = "temp\Result_{num:0>4}.docx".format(num=str(ID))
    document.save(doc_location)
    convert(doc_location, output_path="temp")
    os.remove(doc_location)
    return "Result_{num:0>4}.pdf".format(num=str(ID))
```

Send result as payload to user

```
import smtplib
from os import getenv
from email.mime.application import MIMEApplication
from email.mime.multipart import MIMEMultipart
from email.mime.text import MIMEText

from dotenv import load_dotenv


def payloadPrep(email_address, file):
    load_dotenv()
    username = getenv("EMAIL_ADDRESS")
    password = getenv("EMAIL_PASSWORD")
    msg = MIMEMultipart("mixed")

    sender = "naveen.19BCN7185@vitap.ac.in"
    recipient = email_address
    msg["Subject"] = "Gist Gmail Summary Generated"
    msg["From"] = email_address
    msg["To"] = email_address

    text = "Attached is the gist summary generated from your emails."

    text_message = MIMEText(text, "plain")
    msg.attach(text_message)
    with open(file, "rb") as f:
        attach = MIMEApplication(f.read(), _subtype="pdf")
    attach.add_header("Content-Disposition", "attachment",
filename=str(file)[5:])
    msg.attach(attach)
    mailServer = smtplib.SMTP(
        "mail.smtp2go.com", 2525
    )  # 8025, 587 and 25 can also be used.
    mailServer.ehlo()
    mailServer.starttls()
    mailServer.ehlo()
    mailServer.login(username, password)
    mailServer.sendmail(sender, recipient, msg.as_string())
    mailServer.close()
    return True
```

REFERENCES

[1] Identifying Topics by Position, Lin, C-Y. and E.H. Hovy, In Proceedings of the Applied Natural Language Processing Conference (ANLP-97), 283\u Washington, 1997

[2] Improving summarization through rhetorical parsing tuning, Daniel Marcu, 1998.

[3] Hugging Face Models:
https://huggingface.co/sshleifer/distilbart-cnn-12-6
https://api-inference.huggingface.co/models/tuner007/pegasus_summarizer
https://api-inference.huggingface.co/models/sshleifer/distilbart-cnn-12-6

[4] Text Summarization Portal:
https://www.summarizing.biz/article-summarizer-online/

[5] Fang Chen, Kesong Han and Guilin Chen, "An Approach to sentence selection based text summarization", Proceedings of IEEE TENCON 02, 489-493, 2002

[6] D. R Radev. H. Jing, and M. Budzikowska. Centroid-based summarization of multiple documents: Sentence extraction, utility based evaluation, and user studies. In ANLP/NAACL Workshop on Summarization, Seattle, April, (2000).

[7] D. R Radev. H. Jing, and M. Budzikowska. Centroid-based summarization of multiple documents: Sentence extraction, utility based evaluation, and user studies. In ANLP/NAACL Workshop on Summarization, Seattle, April, (2000).