TEXT SUMMARIZATION

MENTOR : Mr. Narendra Kumar

INTERN : Mr. Sounak Patra

4th Year B.Tech (CSE), Future Institute of Engineering and Management, Kolkata, West Bengal

Author Details:

Email ID : sounak.patra07@gmail.com

Git hub : https://github.com/SounakPatra7/Text-Summarization

LinkedIn : www.linkedin.com/in/sounak-patra-3a29a8291

Contact Number : 7003048733

# ACKNOWLEDGEMENT

I Sounak Patra, would like to express my sincere gratitude to Infosys that has given me this wonderful opportunity to work as an Artificial Intelligence (AI/ML) Intern, Summer 2024 at Infosys Springboard. This internship has been very rewarding for me in terms of the knowledge that I have gained from it as well as its contribution towards my professional development.

I would also like to express my deep gratitude and I am thankful to Mr. Narendra Kumar, my mentor at Infosys for his guidance which is priceless, feedbacks that are insightful and encouragement which never stops. His expertise and assistance have been crucial in shaping this project.

I would like to thank all those who supported me throughout my internship period and helped me with completing this project. Moreover, I am grateful to the members of my team in group 4 and colleagues for their cooperation and support that made this experience truly enriching.

I would like to acknowledge Future Institute of Engineering and Management for their help and encouragement.

Finally, I would like to take this opportunity to sincerely thank my family for continuously standing by me throughout the journey, irrespective of my ups-and-downs

INDEX

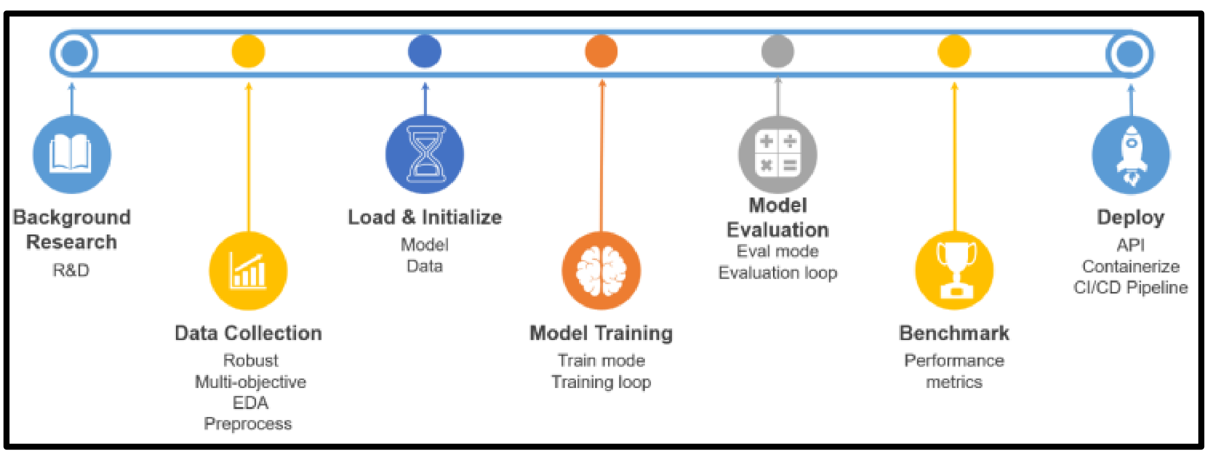
PROBLEM STATEMENT

- **Developing an automated text summarization system that can accurately and efficiently condense large bodies of text into concise summaries is essential for enhancing business operations.**
- **This project aims to deploy NLP techniques to create a robust text summarization tool capable of handling various types of documents across different domains.**
- **The system should deliver high-quality summaries that retain the core information and contextual meaning of the original text.**

PROJECT STATEMENT

- **Text Summarization focuses on converting large bodies of text into a few sentences summing up the gist of the larger text.**
- **There is a wide variety of applications for text summarization including News Summary, Customer Reviews, Research Papers, etc.**
- **This project aims to understand the importance of text summarization and apply different techniques to fulfill the purpose.**

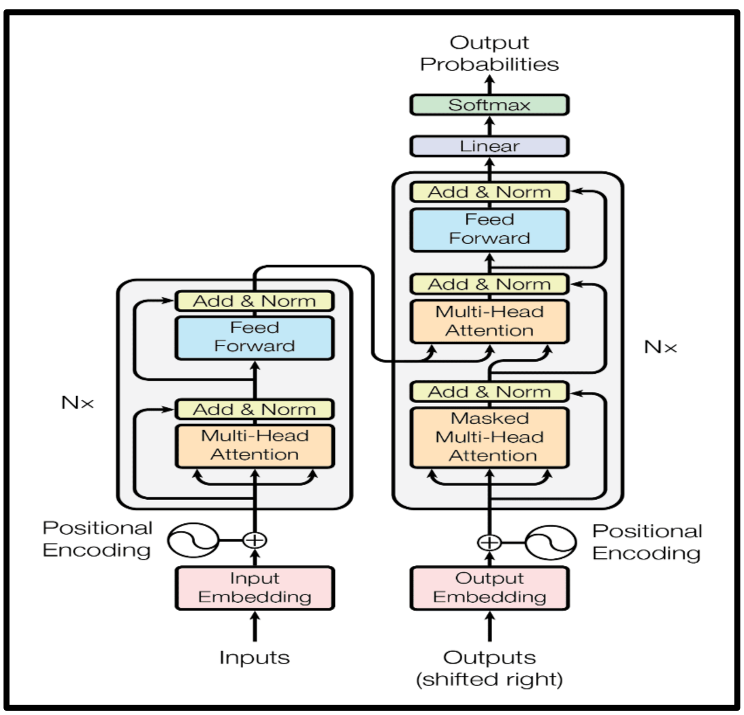
APPROACH TO SOLUTION



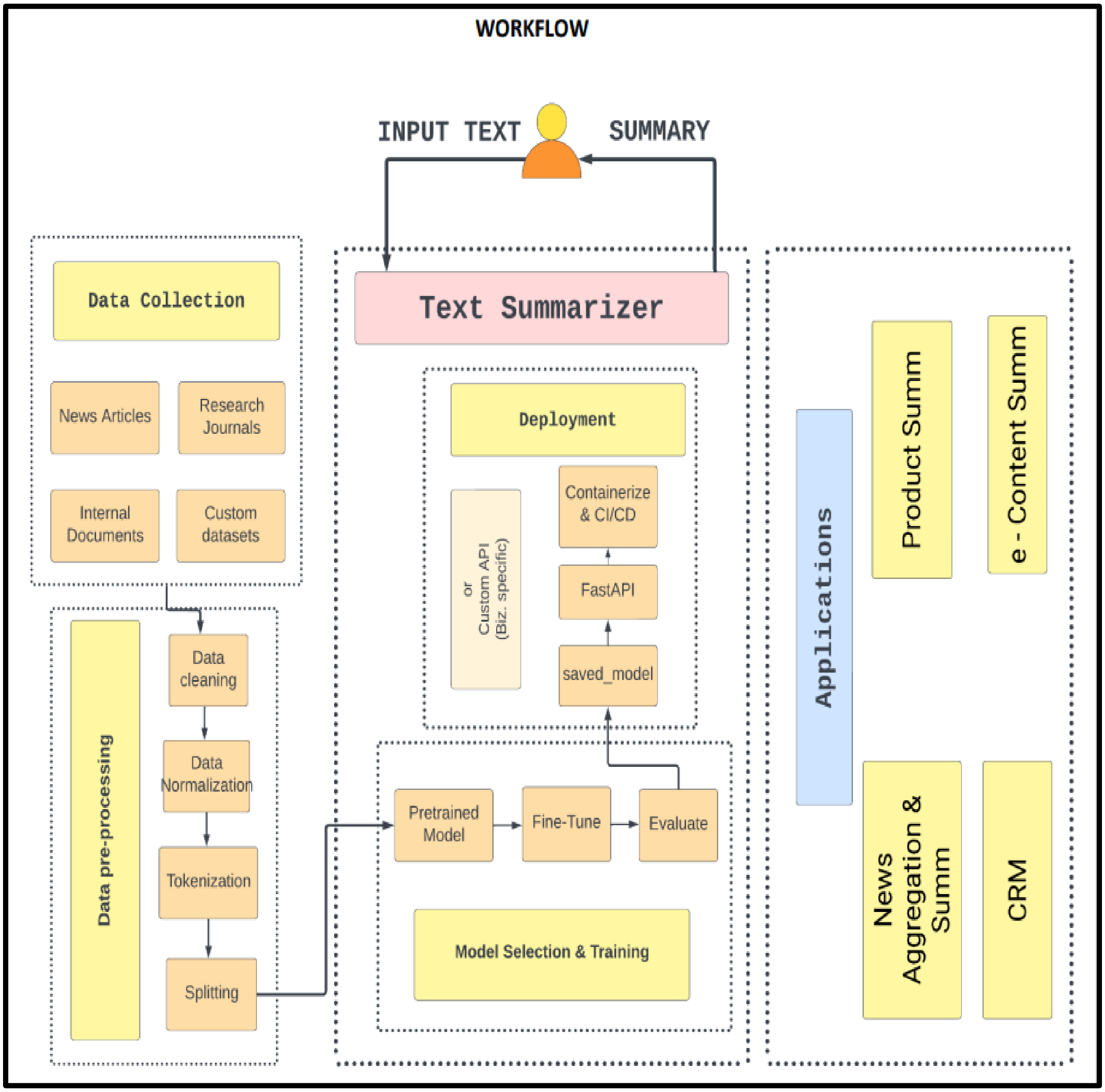
SOLUTION

Selected Deep Learning Architectures

- **Implementation methods:**
  - **From Scratch**
    - **Build Model**
      - **NN**
    - **Initialize normalized W and B**
    - **Train model with extensive data**
    - **Hence,**
      - **Computationally intensive**
      - **Sub optimal usage of resources**
      - **Out of Scope**
  - **Using Pre-trained Model**
    - **Load Model and its Parameters**
    - **Re-train with specific dataset**
    - **Evaluate**
    - **Hence,**
      - **Innovation can be done at intended tasks**
      - **Optimal utilization of resources**



DESIGN WORKFLOW (ABSTRACTIVE)



DATA COLLECTION

Datasets have been collected from the Huggingface datasets library.

Data collected from different source :

- **alexfabbri/multi_news**
  Multi-News, consists of news articles and human-written summaries of these articles from the site newser.com. Each summary is professionally written by editors and includes links to the original articles cited.

- **knkarthick/dialogsum**
  DialogSum is a large-scale dialogue summarization dataset, consisting of 13,460 (Plus 100 holdout data for topic generation) dialogues with corresponding manually labeled summaries and topics.

- **knkarthick/samsum**
  The SAMSum dataset contains about 16k messenger-like conversations with summaries. Conversations were created and written down by linguists fluent in English. Linguists were asked to create conversations similar to those they write on a daily basis, reflecting the proportion of topics of their real-life messenger conversations.

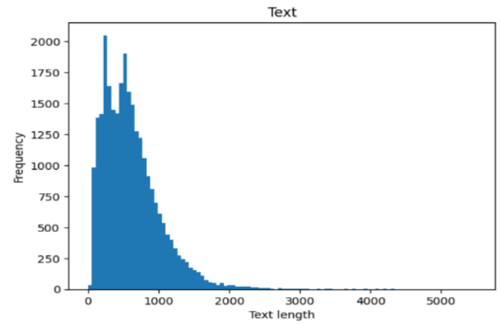
Data were integrated from different sources, to ensure robust and multi-objective data. The objective of the data spans including News articles, Conversations, etc between persons.

The quality and consistency of the raw data is very vital for the model training, to achieve benchmark performance
metrics – ROUGE.

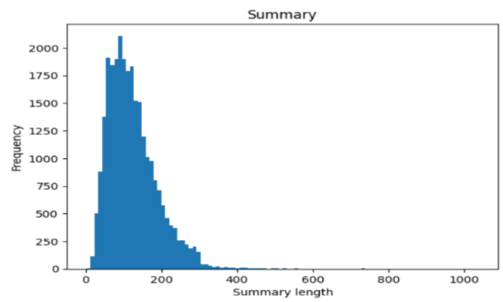
DATA PRE-PROCESSING

Dataset before pre-processing:

```
count    27192.000000
mean       615.244999
std        415.543855
min          0.000000
25%        305.000000
50%        540.000000
75%        814.000000
max       5492.000000
Name: text, dtype: float64
```

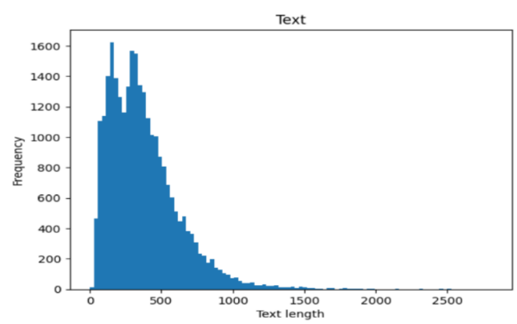



```
count    27192.000000
mean       124.155781
std         64.692599
min          3.000000
25%         76.000000
50%        112.000000
75%        159.000000
max       1039.000000
Name: summary, dtype: float64
```

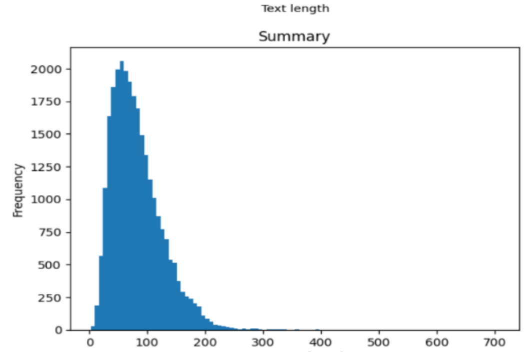


Dataset after pre-processing:

```
count    27192.000000
mean       376.641586
std        251.860155
min          0.000000
25%        190.000000
50%        330.000000
75%        496.250000
max       2810.000000
Name: text, dtype: float64
```




```
count    27192.000000
mean        82.862570
std         43.924925
min          3.000000
25%         51.000000
50%         75.000000
75%        106.000000
max        708.000000
Name: summary, dtype: float64
```



MODEL SELECTION

Transformer model from Huggingface library is selected :

➔ **PEGASUS**

Pegasus (Pre-training with Extracted Gap-sentences for Abstractive Summarization Sequence-to-Sequence Models) is a model developed by Google Research, specifically designed for abstractive text summarization. Pegasus is particularly effective because it is pre-trained on large-scale datasets using a novel self-supervised objective, which involves generating summaries from the masked sentences in a document. For this project, we used the google/pegasus-cnn_dailymail checkpoint, which is fine-tuned on the CNN/Daily Mail dataset.

Key Features:

- ➔ **Pre-trained for abstractive summarization tasks.**
- ➔ **Fine-tuned on a large dataset for improved performance.**
- ➔ **Utilizes transformers architecture, making it suitable for handling long text sequences.**

Implementation:

1. **Data Loading**
   Loaded the datasets from Hugging Face Datasets library.
2. **Data Preprocessing**
   Tokenized the text using the NLTK library to split the documents into sentences.
   Encoded the text using the Pegasus tokenizer to convert it into a format suitable for input to the model.
3. **Model Training**
   Load pre-trained model.
   - **Google /Pegasus-cnn_dailymail**

Defined evaluation metrics.

Configured training arguments.

Initialized Trainer.

Trained model and obtained training history.

```
# Training arguments for Trainer
from transformers import TrainingArguments, Trainer

trainer_args = TrainingArguments(
    output_dir='/kaggle/working/t5-dialogsum', num_train_epochs=2, warmup_steps=500,
    per_device_train_batch_size=1, per_device_eval_batch_size=1,
    weight_decay=0.01, logging_steps=10,
    evaluation_strategy='steps', eval_steps=500, save_steps=1e6,
    gradient_accumulation_steps=16
)
```

```
# Trainer for Seq2Seq model training
trainer = Trainer(model=model_t5, args=trainer_args,
                  tokenizer=tokenizer, data_collator=seq2seq_data_collator,
                  train_dataset=dataset_dialogsum_pt["train"],
                  eval_dataset=dataset_dialogsum_pt["validation"])
```

```
# Train Seq2Seq model
trainer.train()
```

4. **Model Saving**

   Saved fine-tuned PEGASUS model and tokenizer

Results

The model was trained with whole dataset for 2 epochs for 1:15:11, (HH:MM:SS) in 1556 steps.

o Train loss = 1.32 (final)

o ROUGE1 score = 40.45 (Last checkpoint)

o Transformer model for abstractive text summarization was successfully trained with the dataset.

Step	Training Loss	Validation Loss
500	1.227700	1.165420
1000	1.162500	1.088809
1500	1.140000	1.074511

```
TrainOutput(global_step=1556, training_loss=1.3240080149437283, metrics={'train_runtime': 4634.9125, 'train_samples_per_second': 5.377, 'train_steps_per_second': 0.336, 'total_flos': 1.4448795020673024e+16, 'train_loss': 1.3240080149437283, 'epoch': 1.9980738362760835})
```

MODEL EVALUATION

We will use the following metric to validate and evaluate our model:

- **ROUGE (Recall-Oriented Understudy for Gisting Evaluation): This metric helps in measuring the quality of summaries by comparing the overlap of n-grams, word sequences, and word pairs between the generated summary and a reference summary.**
- **Working:**
  - **Load the validation data.**
  - **feature - load & tokenize & convert to tensor**
  - **Generated summary IDs with specified parameters**
  - **Decoded summary IDs to text and skip special tokens**
  - **Generated summaries for the validation set**
  - **computed rouge metrics based on generated summary and original summary (target)**

RESULTS

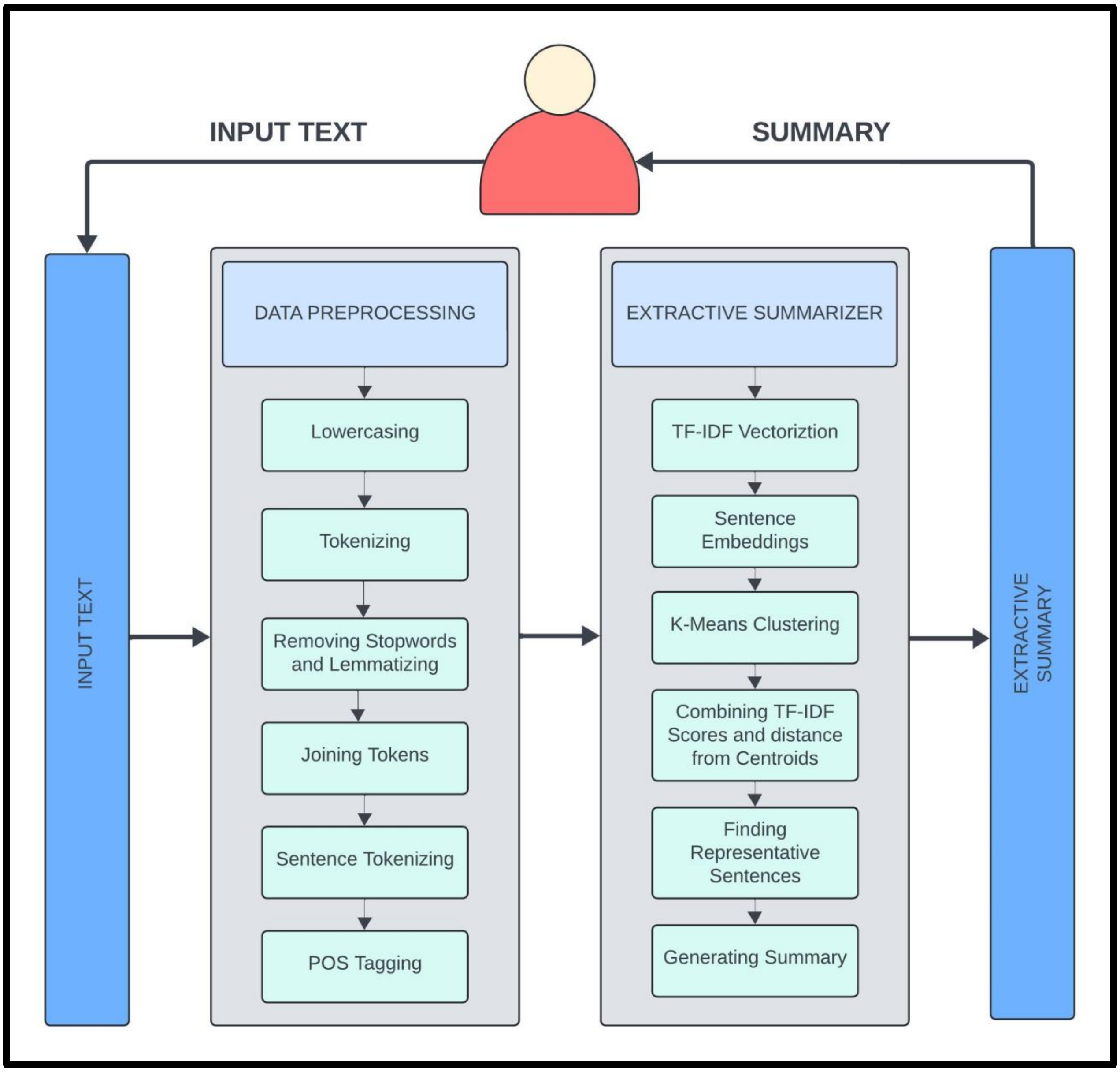
- **Performance Metrics : Before-Fine Tuning**

	rouge1	rouge2	rougeL	rougeLsum
pegasus	0.246641	0.06343	0.186421	0.186551

- **Performance Metrics : After-Fine Tuning**

	rouge1	rouge2	rougeL	rougeLsum
pegasus	0.404575	0.165404	0.332067	0.332024

DESIGN WORKFLOW (EXTRACTIVE)



APPROACH

- **The existing ways for extractive summarization: Text Rank, TF-IDF, ML, DL – models.**

- **Rather than choosing computationally intensive deep-learning models, utilizing a rule-based approach will result in optimal solution. Utilized a new-and-novel approach of combining the matrix obtained from TF-IDF and KMeans Clustering methodology.**

- **Convert the articles/passages into a list of sentences using nltk's sentence tokenizer.**

- **For each sentence, extract contextual embeddings using Sentence transformer.**

- **Apply K-means clustering on the embeddings. The idea is to cluster the sentences that are contextually similar to each other & pick one sentence from each cluster that is closest to the mean(centroid).**

- **For each sentence embedding, calculate the distance from centroid. The distance would be zero if centroid itself is the actual sentence embedding.**

- **For each cluster, select the sentence embedding with lowest distance from centroid & return the summary based on the order in which the sentences appear in the original text.**

IMPLEMENTATION

- **Preprocesses the input text to get POS-tagged sentences.**
- **Data Preprocessing:**
  - **Lowercasing**
  - **Stop Words Removal.**
  - **Lemmatization.**
  - **Tokenization.**
  - **POS Tagging.**
- **Convert our article into a list of sentences using nltk tokenizer.**
- **Implementing TF-IDF Vectorization.**
- **Initialize the Sentence Transformer with STS (Sentence Text Similarity) model.**
- **It takes the inputs as sentences & returns the dense vectors.**
- **By using the STS model we are getting the embeddings for each sentence.**
- **Clustering text embeddings using nltk's KMeansCluster.**
- **Computing the distance between sentence embedding & centroid for each cluster.**
- **To Compute the distance, scipy's distance_matrix function is used.**
- **Combining the TF-IDF scores and distance from centroid score to get more accurate result.**
- **The final step is to generate summary. This can be done by following steps:**
  - **Group the sentences based on Combined column.**
  - **Sort the group ascending order based on combined_score column select the first row.**
  - **Sort the sentences based on their sequence in the original text.**

MODEL EVALUATION

We will use the following metric to validate and evaluate our model:

- **ROUGE (Recall-Oriented Understudy for Gisting Evaluation): This metric helps in measuring the quality of summaries by comparing the overlap of n-grams, word sequences, and word pairs between the generated summary and a reference summary.**

• Rule-based approach for extractive summarization was implemented and evaluated successfully.

• ROUGE1 (F-Measure) = 40.20

```
ROUGE-1 Score:  0.4020618556701031
ROUGE-2 Score:  0.15625000000000003
ROUGE-L Score:  0.2268041237113402
```

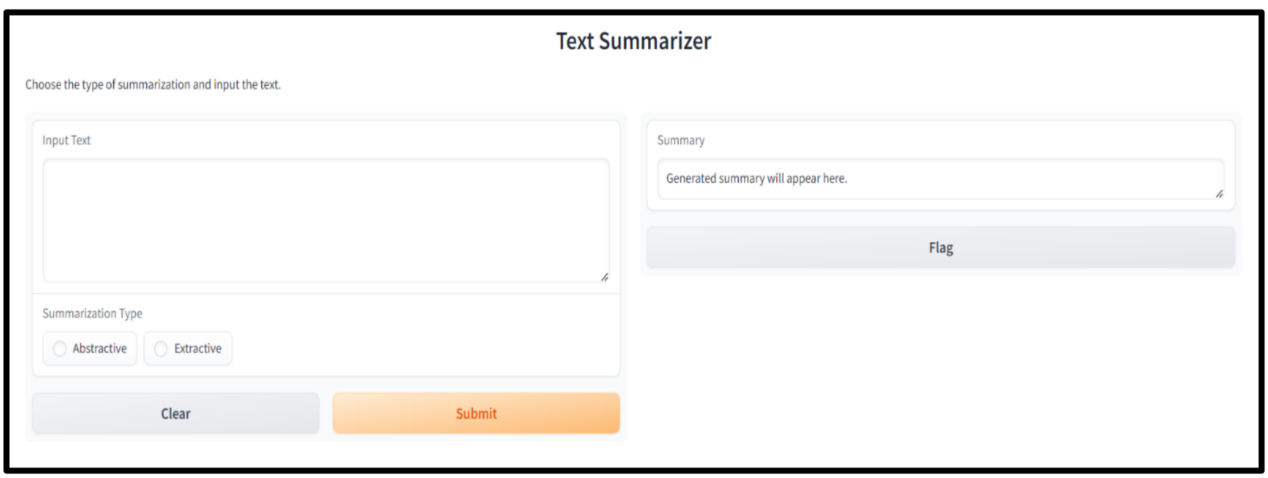
TESTING

- **Text Summarization Application (Both Abstractive & Extractive)**

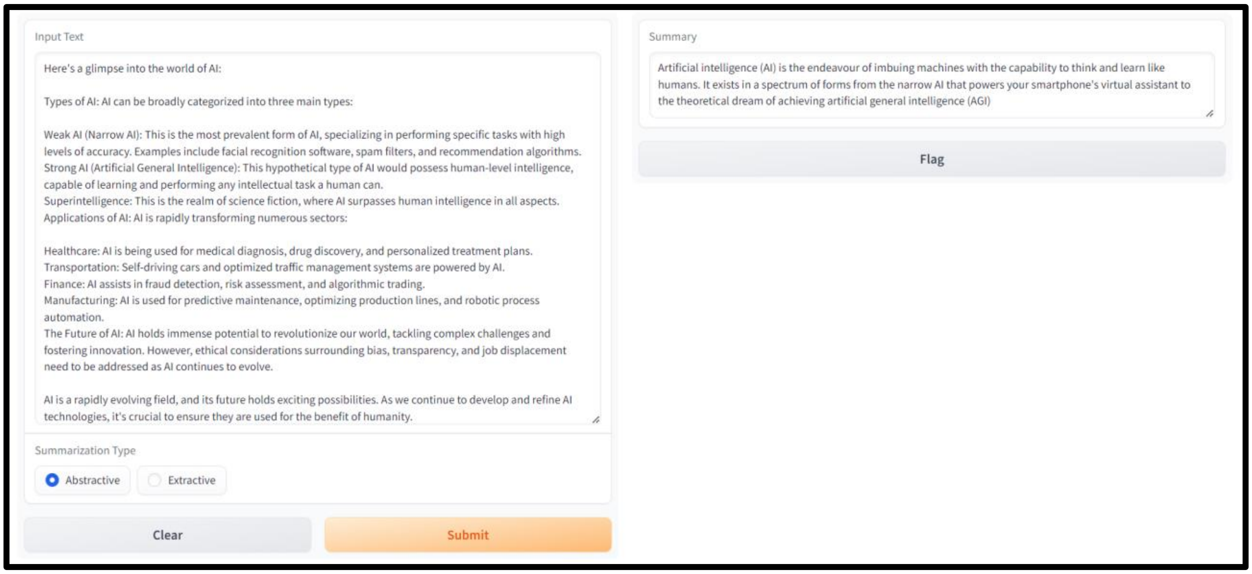
Making the UI using Gradio:

Gradio - Gradio is an open-source Python package that allows you to quickly build a demo or web application for your machine learning model, API, or any arbitary Python function.

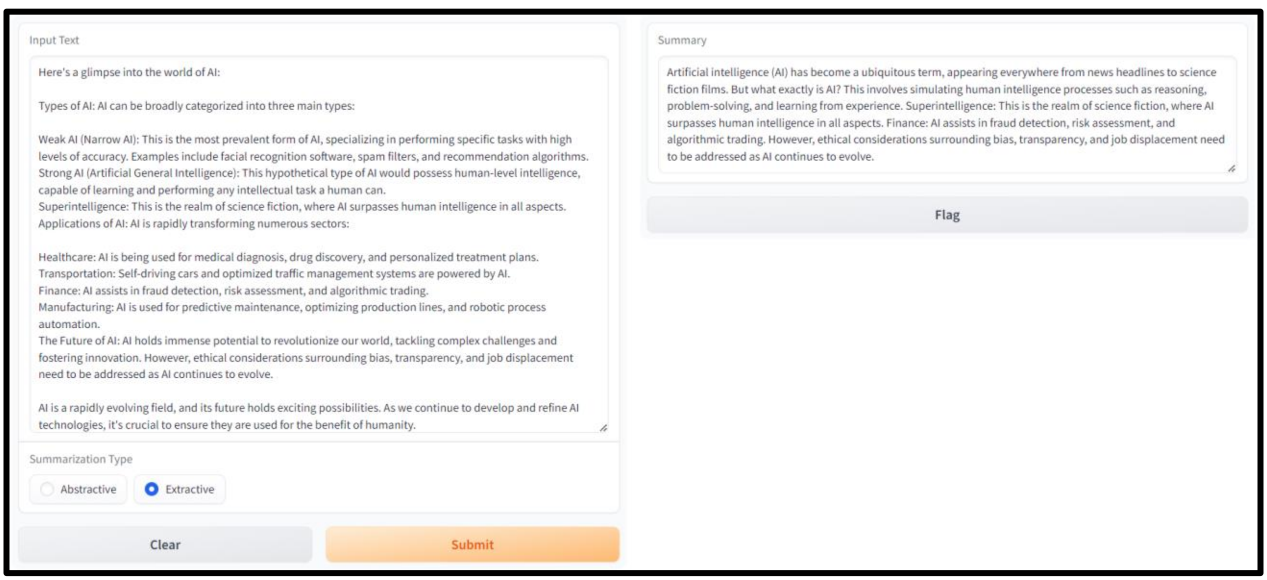
- **Loading the saved model for abstractive summarization**
- **Defining function for abstractive summarization**
- **Defining function for extractive summarization (rule-based approach)**
- **Defining the function to choose between the type of summarization**
- **Defining the Gradio interface**
- **Launching the interface using public URL**



- **ABSTRACTIVE TEXT SUMMARIZATION**



- **EXTRACTIVE TEXT SUMMARIZATION**



DEPLOYMENT

- **File Structure :**

```
│    app.py
│    Dockerfile
│    extractive_summary.py
│    requirements.txt
│
├───saved_model
│        config.json
│        generation_config.json
│        model.safetensors
│        special_tokens_map.json
│        spiece.model
│        tokenizer_config.json
│
└───templates
         index.html
```

API ENDPOINTS

- **Utilized the FastAPI framework to create a web application for text summarization.**
- **Developed to handle text input.**
- **Created API endpoints using FastAPI, for inferencing the summarized model.**
- **Defined a function to summarize text using the saved model for abstractive text summarization.**
- **Encodes the text, generates a summary, and decodes the summary back to text.**
- **API Endpoints:**
  - **Summarize Text:**
    - **Accepts direct text input from a form.**
    - **Generates summaries.**
    - **Returns them as a JSON response**
- **Developed to run the FastAPI application using Uvicorn, listening on all available IP addresses on port 8000.**

```python
app = FastAPI()
templates = Jinja2Templates(directory="templates")

# Abstractive summarization model
model = PegasusForConditionalGeneration.from_pretrained('saved_model')
tokenizer = PegasusTokenizer.from_pretrained('saved_model')

def abstractive_summarization(text):
    tokens = tokenizer.encode("summarize: " + text, return_tensors="pt", max_length=512, truncation=True)
    summary_ids = model.generate(tokens, max_length=150, min_length=40, length_penalty=2.0, num_beams=4, early_stopping=True)
    summary = tokenizer.decode(summary_ids[0], skip_special_tokens=True)
    return summary

@app.get("/", response_class=HTMLResponse)
async def read_root(request: Request):
    return templates.TemplateResponse("index.html", {"request": request, "summary": "", "text": "", "error_message": None})

@app.post("/summarize", response_class=HTMLResponse)
async def summarize(request: Request, text: str = Form(None), summary_type: str = Form(...)):
    if not text:
        return templates.TemplateResponse("index.html", {"request": request, "summary": "", "text": "", "error_message": "No text input provided."})
    summary = ""
    if summary_type == "abstractive":
        summary = abstractive_summarization(text)
    else:
        summary = extractive_summarization(text)
    return templates.TemplateResponse("index.html", {"request": request, "summary": summary, "text": text, "error_message": None})

if __name__ == '__main__':
    import uvicorn
    uvicorn.run(app, host='0.0.0.0', port=8000)
```

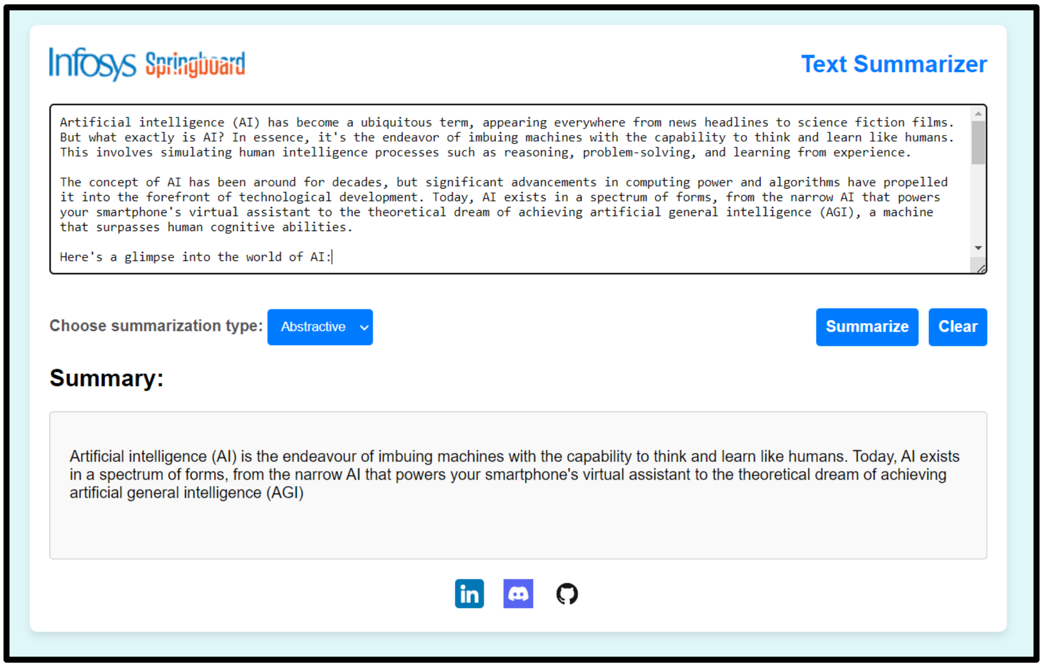
EXTRACTIVE SUMMARY SCRIPT

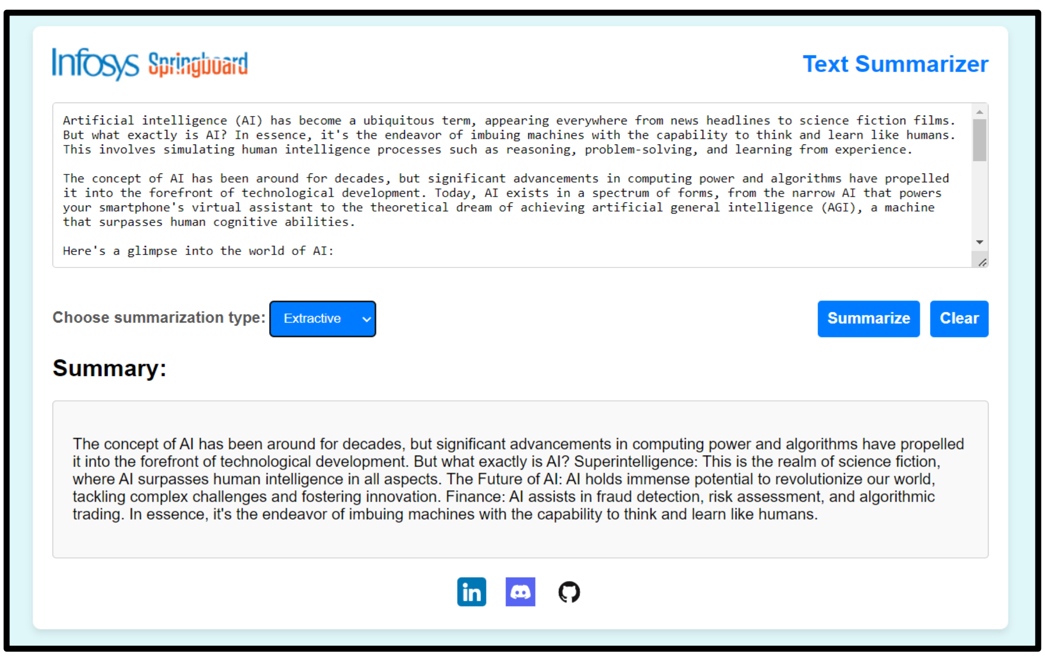
- **Implemented an extractive summarizer module as python script for application.**
- **Developed to utilize the same programmatic approach of src/extractive-model.ipynb.**
- **This script is designed to:**
    - **Preprocess text.**
    - **Generate sentence embeddings using Sentence Transformer.**
    - **Extract important features using TF-IDF.**
    - **Find the distance from centroid and TF-IDF scores and combine them.**
    - **Summarize the text by selecting representative sentences from different clusters.**

```python
# Initialize Sentence Transformer model
model = SentenceTransformer('stsb-roberta-base')

def extractive_summarization(article):
    nltk.download('punkt')
    # Tokenize the article into sentences
    sentences = nltk.sent_tokenize(article)
    sentences = [sentence.strip() for sentence in sentences]

    # Create a DataFrame with the sentences
    df = pd.DataFrame(sentences, columns=['sentences'])

    # Compute TF-IDF scores
    vectorizer = TfidfVectorizer()
    tfidf_matrix = vectorizer.fit_transform(df['sentences'])
    tfidf_scores = np.sum(tfidf_matrix, axis=1)

    # Normalize TF-IDF scores
    normalized_tfidf_scores = tfidf_scores / np.sum(tfidf_scores)
    df['tfidf_score'] = normalized_tfidf_scores

    # Get embeddings for each sentence
    df['embeddings'] = df['sentences'].apply(lambda sent: model.encode([sent])[0])

    # Determine number of clusters and iterations
    n_clusters=int(len(df)/3)
    iterations=25

    # Convert embeddings into numpy array
    X = np.array(df['embeddings'].tolist())

    # Perform clustering
    kcluster = KMeansClusterer(n_clusters, distance=nltk.cluster.util.cosine_distance, repeats=iterations, avoid_empty_clusters=True)
    assigned_clusters = kcluster.cluster(X, assign_clusters=True)

    # Assign clusters and centroids to the DataFrame
    df['Cluster'] = assigned_clusters
    df['Centroid'] = df['Cluster'].apply(lambda x: kcluster.means()[x])

    # Calculate distance from centroid for each sentence
    df['distance_from_centroid'] = df.apply(lambda row: distance_matrix([row['embeddings']], [row['Centroid'].tolist()])[0][0], axis=1)

    # Sort sentences by combined score of distance from centroid and TF-IDF score
    df['combined_score'] = df['distance_from_centroid'] * df['tfidf_score']

    # Select top sentence from each cluster based on combined score
    sents = df.sort_values(by='combined_score', ascending=True).groupby('Cluster').head(1)['sentences'].tolist()

    # Create the final summary
    summary = ' '.join(sents)
    return summary
```

USER INTERFACE

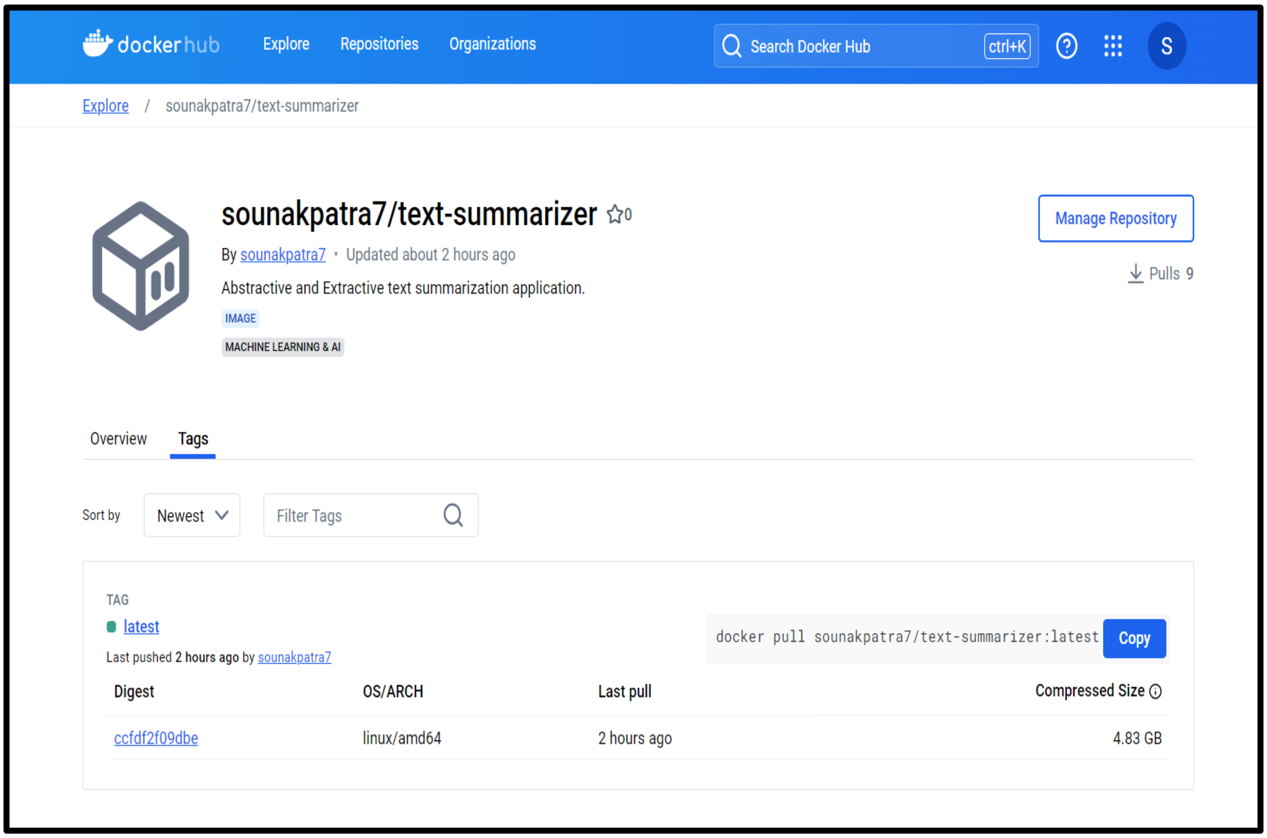
Developed to provide a user-friendly interface for text summarization application





CONTAINERIZATION

- **Developed a Dockerfile to build the docker image for the FastAPI application.**
- **The containerized docker image packages the entire application and its dependencies along with the saved model. Making the application easy to run consistently across different environments, hence removes the bottlenecks in production environments.**
- **Built the image & pushed into docker hub.**



- **Deployed using Docker Desktop at http://localhost:8000/**

CONCLUSION

In this project, we developed a comprehensive text summarization application that supports both extractive and abstractive summarization techniques.

By leveraging state-of-the-art models like Pegasus for abstractive summarization, we ensured high-quality and coherent summaries. The application was implemented using a combination of Python libraries, including the Hugging Face Transformers library for model training and FastAPI for creating an interactive user interface.

The evaluation results demonstrated the effectiveness of our approach, with high ROUGE scores indicating the model's ability to generate accurate summaries. The application was further enhanced by deploying it using Docker, making it easy to distribute and run in various environments.

Overall, this project showcases the potential of modern NLP techniques in automating the summarization process, making it a valuable tool for individuals and organizations dealing with large volumes of text data.

One of the key strengths of this project is its flexibility and potential for future enhancements. The modular design allows for easy updates and the integration of additional features, ensuring that the application can evolve with advancements in NLP research and user needs.

FUTURE SCOPE

While the current implementation is robust and performs well, there are several avenues for future improvement and expansion:

1. Model Fine-Tuning and Customization:
    - Fine-tune the models on more diverse datasets to improve their adaptability to different types of text, such as scientific papers, legal documents, and news articles.
    - Experiment with other pre-trained models and architectures to compare performance and potentially achieve better results.
2. Multilingual Summarization:
    - Extend the application to support multiple languages, allowing for summarization of text in languages other than English. This would involve training and fine-tuning models on multilingual datasets.
3. Integration of Additional Features:
    - Implement additional NLP features such as sentiment analysis, keyword extraction, and topic modeling to provide a more comprehensive text analysis toolkit.
    - Add support for user-defined custom summarization models, allowing users to upload and utilize their own trained models.
4. Enhanced User Interface:
    - Improve the user interface to include more interactive elements, such as adjustable parameters for summarization length and beam search settings.
    - Provide real-time feedback on summary quality, enabling users to make adjustments and see the impact immediately.

5. **Performance Optimization:**
    - **Optimize the application for faster response times and lower computational resource usage, especially for deployment in resource-constrained environments.**
    - **Explore the use of techniques such as model distillation and quantization to reduce the model size and improve inference speed.**
6. **API and Integration:**
    - **Develop a RESTful API to allow seamless integration of the summarization service into other applications and workflows.**
    - **Create plugins or extensions for popular content management systems (CMS) and text editors to enable easy access to summarization features directly within those platforms.**
7. **User Feedback and Continuous Improvement:**
    - **Implement mechanisms for users to provide feedback on the generated summaries, and use this feedback to iteratively improve the models and the application.**
    - **Continuously monitor and evaluate the performance of the application, incorporating new research and advancements in the field of NLP to keep the system up-to-date.**

By exploring these future directions, we can further enhance the capabilities and applicability of the text summarization application, making it an even more powerful tool for managing and extracting insights from text data.

THANK YOU.