TEXT SUMMARIZATION

Mentor: Mr. Narendra Kumar

ACKNOWLEDGEMENT

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

We would also like to express our deep gratitude and we are thankful to ***Mr. Narendra Kumar***, our 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.

We would like to thank all those who supported us throughout our internship period and helped our with completing this project. Moreover, we are grateful to the members of our team in ***group 4*** and ***colleagues*** for their cooperation and support that made this experience truly enriching.

Finally, we would like to take this opportunity to sincerely thank ***our family*** for continuous support throughout the journey, irrespective of our ups-and-downs.

Table of Contents

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

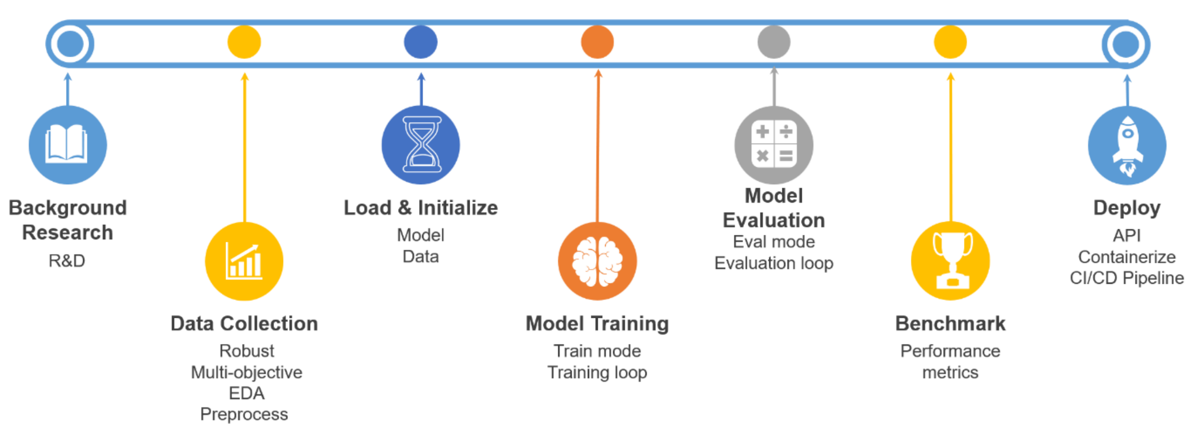


Fig.: Transformer architecture

Background Research

Literature Review

S. No	Use-Case	Paper Title	Year	Method	Dataset	Results	Limitations
1	General text summarization	Text Summarization Using Deep Learning Techniques: A Review	2023	Deep Learning (Seq2Seq, Attention, Transformers)	CNN/Daily Mail, XSum	Improved performance in capturing semantic relationships, better coherence	Computationally expensive, requires large datasets
2	Implementation of the Transformer architecture	Attention is all you need	2023	Transformer	WMT 2014 English-German, WMT 2014 English-French	Introduced the Transformer architecture, significantly improving the performance of text summarization tasks.	Requires large datasets and computational resources for training.
3	Multi-document summarization	Surveying the Landscape of Text Summarization with Deep Learning	2023	Deep learning methods. Various techniques like RBMs and fuzzy logic employed for summarization.	CNN/DailyMail	Incorporating transfer learning enhances summary quality and reduces data demand.	Complex models, high computational resources
4	Abstractive summarization	Pegasus: Pre-training with gap-sentences for abstractive summarization	2020	Transformer (Pegasus)	XSum, CNN/DailyMail, and Reddit TIFU	Significant improvements in abstractive summarization quality	Resource-intensive
5	Extractive summarization	Text Summarization with Pretrained Encoders	2019	Intersentence Transformer layers for summarization	CNN/Daily Mail, NYT, Xsum, DailyMail	BERT-based models outperformed other approaches in abstractive summarization.	High computational resources required

SOLUTION

Selected Deep Learning Architecture

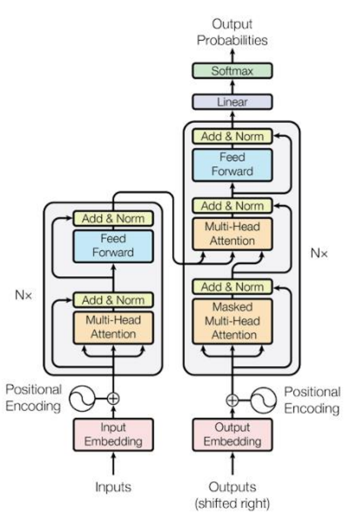


Fig.: Transformer architecture.

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

WORKFLOW

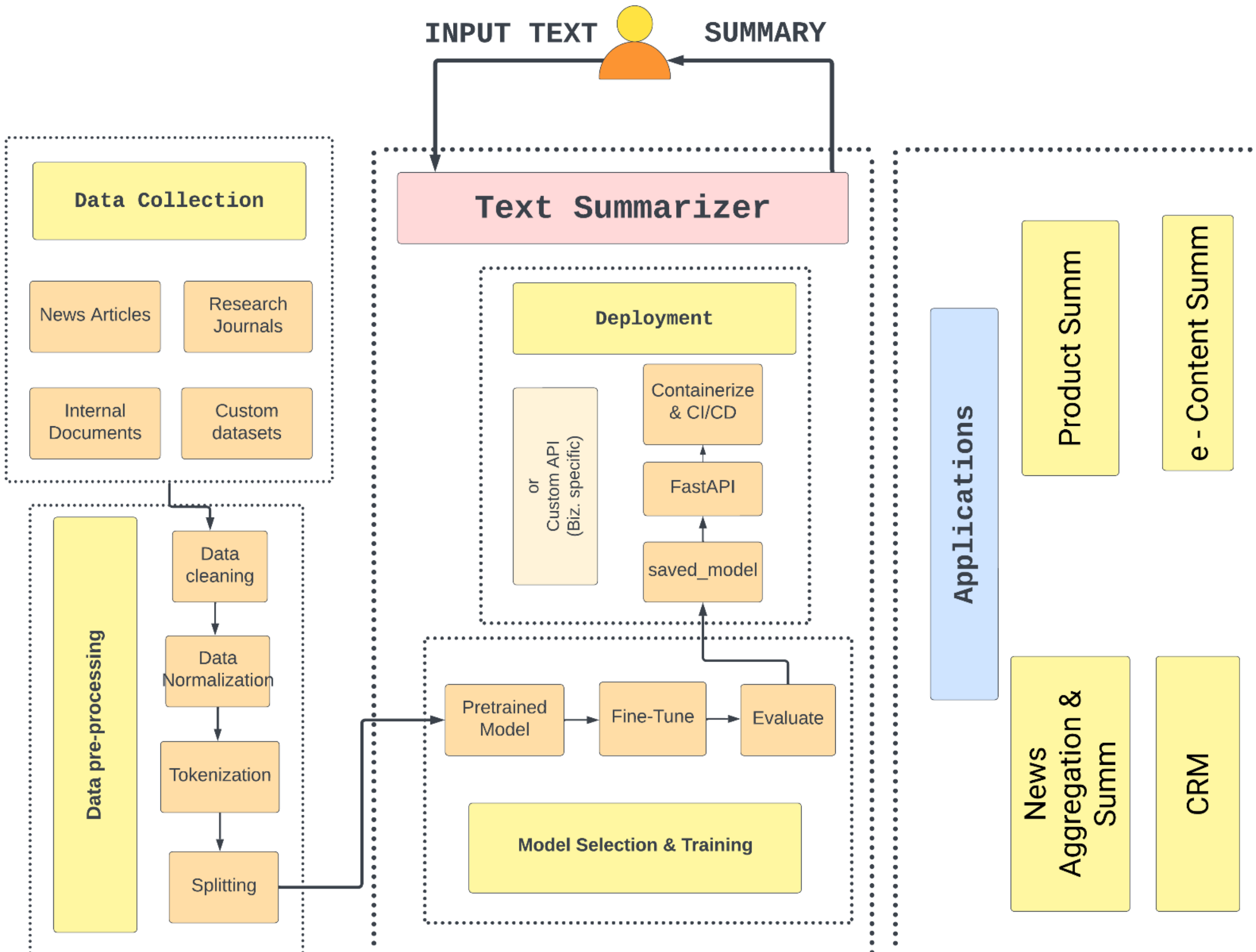


Fig.: Workflow for Text Summarizer.

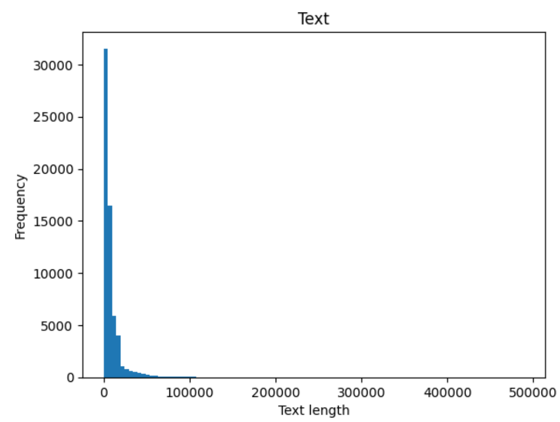
DATA COLLECTION

- Data Preprocessing & Pre-processing Implemented in [src/data_preprocessing].
- Data collection from different source
    - CNN, Daily Mail: News,
    - BillSum: Legal,
    - ArXiv : Scientific,
    - Dialoguesum : Conversations.
- Data were integrated from different sources, to ensure robust and multi-objective data. The objective of the data spans including News articles, Legal Documents – Acts, Judgements, Scientific papers, Conversations between persons.
- The quality and consistency of the raw data is very vital for the model training, to achieve benchmark performance metrics - ROUGE.
- Validated the data through Data Statistics and Exploratory Data Analysis (EDA) through Frequency Plotting, for every data source.
- Saved each data in separate files (preprocessed - 1, 2, 3, 4) in csv format.
- Data integration of selective data from each source.
    - selected from splits (train, test, val).
    - Factors considered:
        - Number of records
        - Quality of data
        - Representation of different data groups (Multi-Objective)
        - To enable the model training in domestic GPU.
            - Limits the number of records.

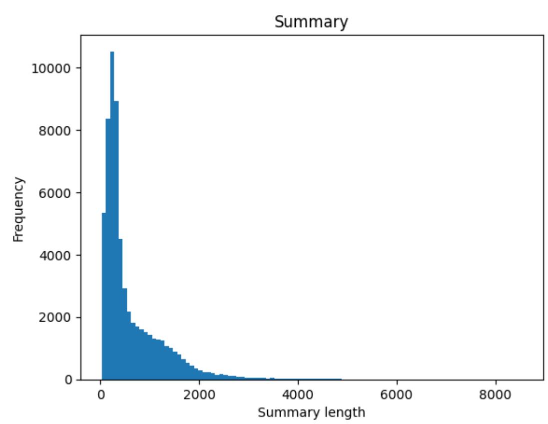
- Selected data:
  - CNN, Daily Mail - test, validation
  - BillSum - train
  - ArXiv - test
  - Dialoguesum - train
- renamed the attributes to general form (text, summary).
- Validated the data using statistics and frequency plot.

Dataset before cleansing

```
count     62707.000000
mean       8209.765879
std       11786.346696
min         190.000000
25%        1892.000000
50%        5051.000000
75%        9439.500000
max      489287.000000
Name: text, dtype: float64
```



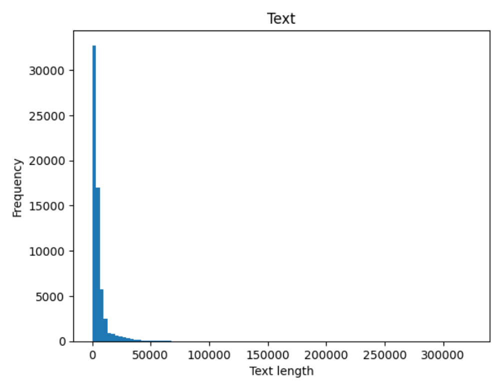

```
count    62707.000000
mean       612.585150
std        620.618153
min         31.000000
25%        213.000000
50%        353.000000
75%        839.000000
max       8541.000000
Name: summary, dtype: float64
```



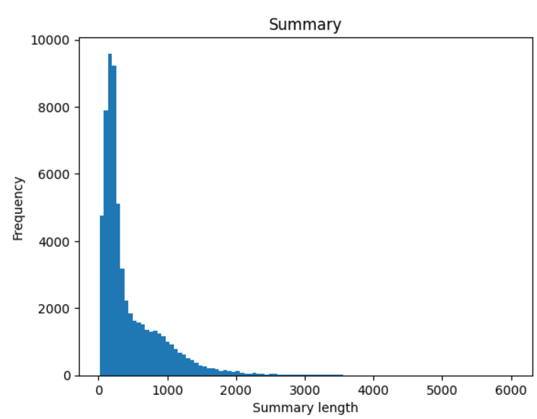
- Performed data cleansing optimized for NLP tasks.
    - Removed null records.
    - Lowercasing.
    - Punctuation removal.
    - Stop words removal
    - Lemmatization.

Dataset after cleansing

```
count     62707.000000
mean       5211.270975
std        7794.860686
min          83.000000
25%        1275.000000
50%        3176.000000
75%        5684.500000
max      323742.000000
Name: text, dtype: float64
```




```
count    62707.000000
mean       448.081937
std        459.087443
min         16.000000
25%        154.000000
50%        255.000000
75%        618.000000
max       6014.000000
Name: summary, dtype: float64
```



- Performed data splitting from the integrated data- train, test, validation - for training, testing and validating the model respectively, using sci-kit learn.
    - saved the data in csv format.
    - [Dataset](#) – for reference.

ABSTRACTIVE TEXT SUMMARIZATION

MODEL TRAINING & EVALUATION

TRAINING

- The selected transformer architecture, for ABSTRACTIVE SUMMARIZATION, can be implemented in a couple of ways. Either developing the neural network from scratch and initializing it with normalized weight and biases, then training the model with massive datasets for NLP tasks, or retraining the foundational model with custom dataset i.e. fine-tuning the pre-trained model.
- The former way, will pave the way to excessive unoptimized resource utilization, in terms of computation (GPU). Also, this would not out-perform the fine-tuning of the larger foundational model.

Fig.: Fine-Tunning Overview

- The choice of foundational model vests in considering lots of factors including its performance metrics and efficient trainable parameters in the model.
- With due analysis, *Facebook's Bart Large* – was chosen as the foundational model for abstractive text summarization.
    - `406,291,456 total parameters.`
    - `406,291,456 training parameters.`
- Model Training can be achieved in two (2) methods:
    1. Native PyTorch Implementation
    2. Trainer API implementation.

EVALUATION

- Performance metrics – ROUGE (Recall-Oriented Understudy for Gisting Evaluation)

  - ROUGE – Measures the overlap between generated summary and reference summary.
  - Best suited: evaluating ‘Text Summarization’ tasks.
  - Other options : **BLEU**.
  - Available metrics in ROUGE: ROUGE-N = 1,2,3; ROUGE LSUM.

METHOD 1 - Native PyTorch

- Native PyTorch implementation, source code: src/model.ipynb
- Implemented works as follows:
- Loaded pre-trained transformer
  - Facebook's Bart Large
- Developed OOP implementation of Dataset
  - Feature, Target Loading
  - Tokenized the dataset
  - Padding, Truncate w.r.t maximum length
  - Converted to Tensor
  - Passed on to DataLoader – with batch size
- Developed manual PyTorch training Loop
  - Set the model in 'Train mode'.
  - Utilized 'Adam' optimizer.
  - Forward pass & compute loss
  - Backward pass
  - Updated params – compute gradient
  - Updated Learning Rate
  - Zeroed the gradients
  - Updated total loss
  - [Average Training Loss: 1.3280]
- Saved the fine-tuned transformer model -> saved model
- Developed manual evaluation loop:
  - Set the model to 'Eval mode'.
  - No gradient calculation for evaluation.

  - Forward pass & compute loss.
  - Accumulate batch loss.
  - Printed batch information.
  - Calculated average validation loss.
  - Printed final evaluation results (loss and time).
  - [Validation Loss: 2.4502].

METHOD 1 - MODEL EVALUATION

- Model Evaluation, source code: src/evaluation.ipynb
- Implemented works:
- Load fine-trained transformer
  - From saved model
- OOP implementation of Dataset
  - Feature, Target
  - Tokenize
  - Padding, Truncate
  - Convert to Tensor
  - Pass to: DataLoader – with batch size
- Evaluate Model
  - Set model to evaluation mode
  - Load ROUGE metric
  - Loop through batches in DataLoader
  - Move data to device
  - Generate summaries
  - Decode predictions and labels
  - Add to ROUGE metric

- Compute ROUGE scores

- Evaluate model on validation dataset
    - Print results (ROUGE mid).
- Saved fine-tuned BART model and tokenizer. (Saved model)

Results:

- Performance metrics – After fine-tuning.

ROUGE-1

- **Precision:** 0.000307
- **Recall:** 0.000163
- **F-Measure:** 0.000182

ROUGE-2

- **Precision:** 0.000000
- **Recall:** 0.000000
- **F-Measure:** 0.000000

ROUGE-L

- **Precision:** 0.000295
- **Recall:** 0.000138
- **F-Measure:** 0.000161

ROUGE-Lsum

- **Precision:** 0.000293
- **Recall:** 0.000141
- **F-Measure:** 0.000164

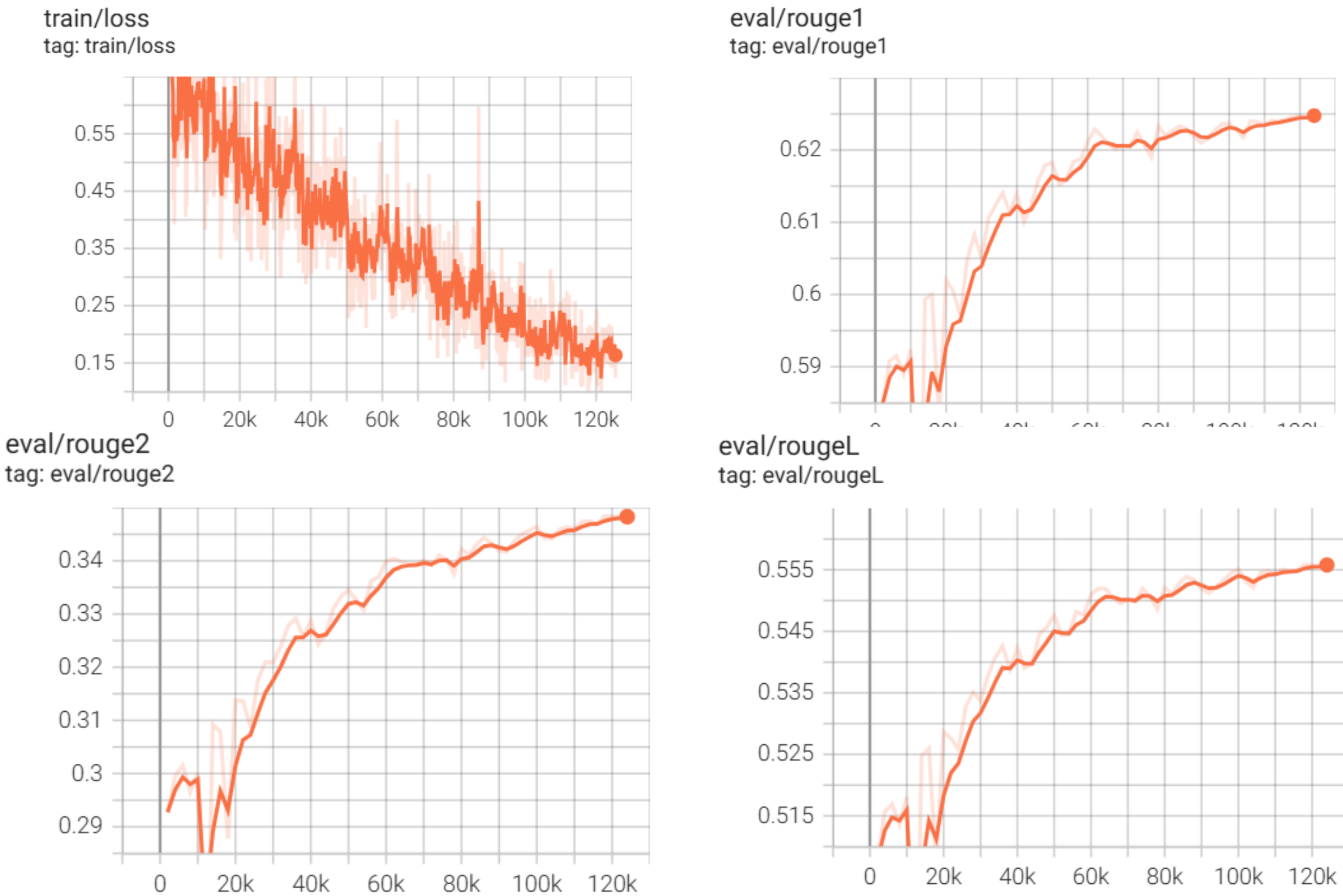
Observations:

- The trained model from method 1 was not used for deployment.
    - (Trained model from method 2 was used for deployment)
- Reason:
    - Even though the model has very minimal training loss but, the model performed inconsistently in validation & testing phase.
    - There's a suspected tensor error while training using method 1, which could be attributed to the inconsistency of the model's output.
- *Dire need to implement alternative approach* optimized for transformer model, to produce benchmark performance.

METHOD 2 – Trainer Class Implementation

- Trainer class implementation, source code: src/bart.ipynb.
- Utilized Trainer API from Hugging face, Trainer class is optimized to train the transformer models and Pre-Trained models.
- Enables to have feature-complete training and eval loop for PyTorch, optimized for transformer models.
- Implemented works in this module, as follows:
- Data Preparation:
    - Loaded and converted datasets to Hugging Face Dataset format -PyArrow
- Data Preprocessing:
    - text-to-input conversion.
    - mapped preprocessing to datasets with parallel processing.
- Model Training:
    - Load pre-trained model
        - Facebook - Bart large
    - defined evaluation metrics
    - Configured training arguments
    - Initialized Trainer
    - Trained model and obtained training history.
- Model Saving:
    - Saved fine-tuned BART model and tokenizer (Saved model)

TensorBoard Training Monitor



Results

- The model was trained with whole dataset for 10 epochs for 26:24:22 (HH:MM:SS) in 125420 steps.
    - Train loss = 17.28 (final)
    - ROUGE1 score = 62.52 (Last checkpoint)
    - Transformer model for abstractive text summarization was successfully trained with the integrated custom data.

Observations

- On testing the model, from method 2, consistent performance was observed.
- Considered the performance metrics of the models trained by the forementioned methods.
- After the due analysis, the model trained using 'Method 2' was selected for deployment.

METHOD 2 – MODEL EVALUATION

- Evaluation for the model trained using Trainer class implementation, source code: src/rouge.ipynb.
- Performance metrics – ROUGE (Recall-Oriented Understudy for Gisting Evaluation).
- Implemented works:
  - 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:

ROUGE-1

- **Precision:** 0.3097
- **Recall:** 0.4397
- **F-Measure:** 0.2905

ROUGE-2

- **Precision:** 0.1254
- **Recall:** 0.1778
- **F-Measure:** 0.1161

ROUGE-L

- **Precision:** 0.2009
- **Recall:** 0.2908
- **F-Measure:** 0.1879

ROUGE-Lsum

- **Precision:** 0.2409
- **Recall:** 0.3450
- **F-Measure:** 0.2271

- Performance Metrics - After fine-tuning:

ROUGE-1

- **Precision:** 0.6592
- **Recall:** 0.6325
- **F-Measure:** 0.6132

ROUGE-2

- **Precision:** 0.5080
- **Recall:** 0.4969
- **F-Measure:** 0.4787

ROUGE-L

- **Precision:** 0.5735
- **Recall:** 0.5600
- **F-Measure:** 0.5397

ROUGE-Lsum

- **Precision:** 0.6167
- **Recall:** 0.5946
- **F-Measure:** 0.5762

Observations

- The trained model from method 1 was not used for deployment:
- Trained model from method 2 was used for deployment
- Reason:
    - Even though the model has very minimal training loss but, the model performed inconsistently in validation & testing phase.
    - There's a suspected tensor error while training using method 1, which could be attributed to the inconsistency of the model's output.
    - Model 2 - results outperformed that of method 1.
        - ROUGE1 (F-Measure) = 61.32 -> Benchmark grade
        - GPT4 performance for text summarization - ROUGE1 (F-Measure) is 63.22

COMPARATIVE ANALYSIS

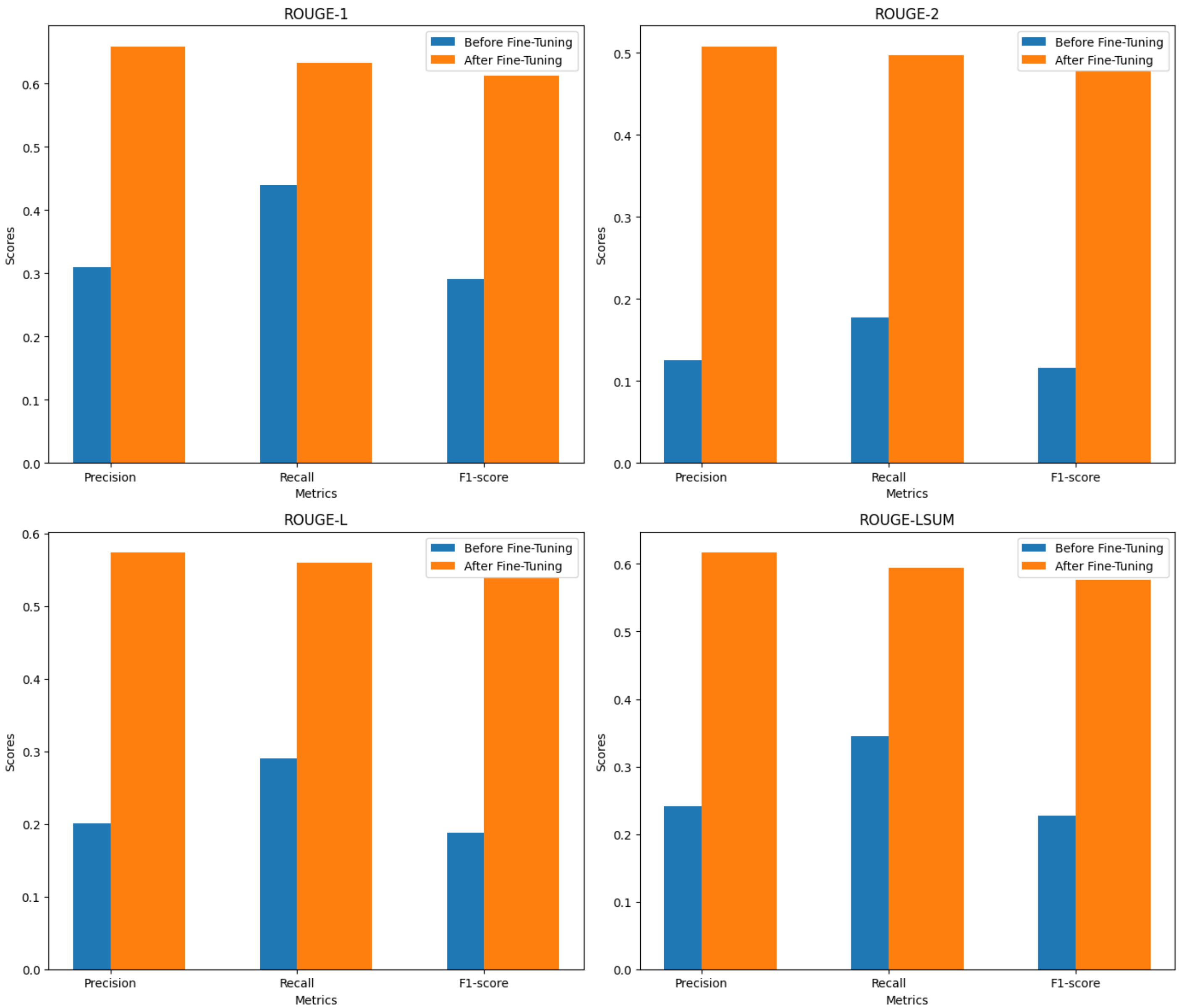


Fig.: Comparison of model's performance metrics.

- Performance comparison of the fine-tuned transformer model, source code: src/compare.ipynb.
- Implemented works:
    - Before fine tuning vs After fine tuning
        - Basis: ROUGE scores - mid values

Observations

- The fine-tuning process significantly improved the transformer's performance across all ROUGE metrics:
    - ROUGE-1 and ROUGE-LSUM show over 100% improvement in F1-score.
    - ROUGE-2 demonstrates the highest improvement with over 300% increase in F1-score
    - ROUGE-L scores also saw substantial improvements, particularly in Precision and F1-score.
- These results indicate that fine-tuning has greatly enhanced the model's ability to generate accurate and relevant summaries.
- Hence, the *fine-tuning of the transformer model is successful.*

EXTRACTIVE TEXT SUMMARIZATION

MODEL

- Utilized a rule based approach, source code: [src/Extractive_Summarization.ipynb]().

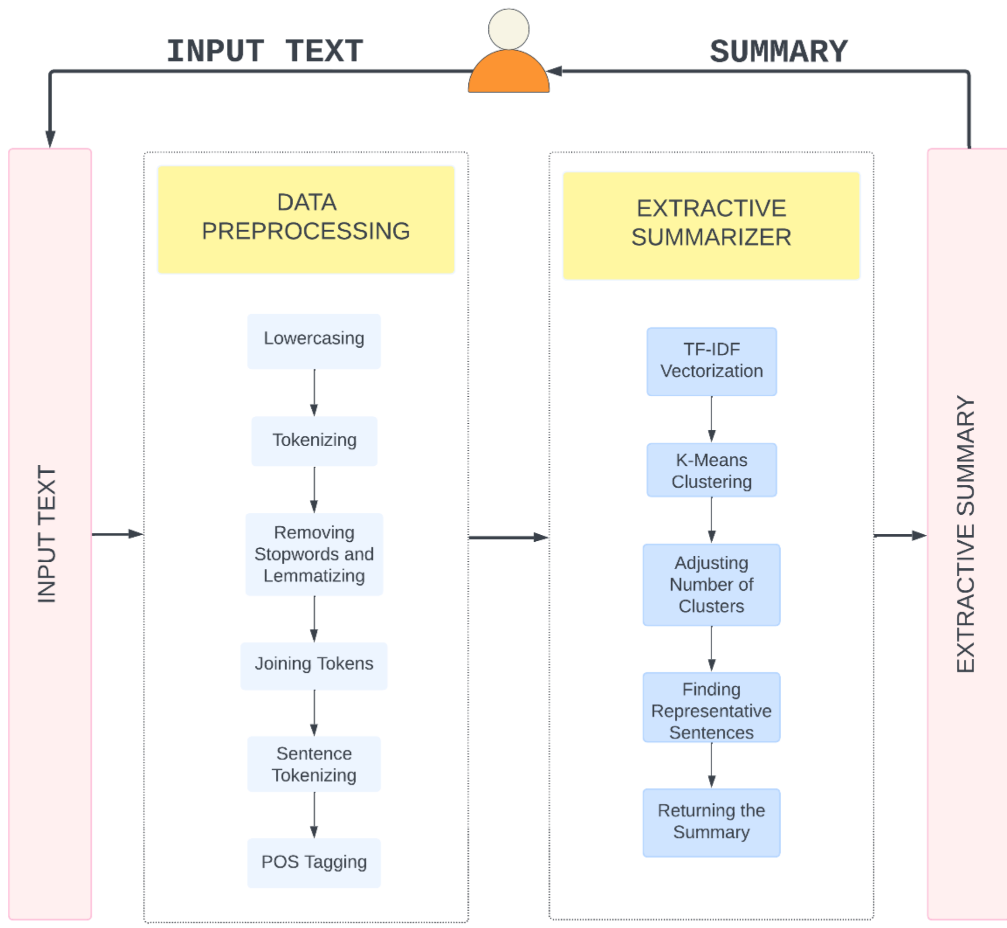


Fig.: Workflow for Extractive Text Summarizer.

- 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.
- It is the expanded topic modeling specifically to be applied to multiple lower-level specialized entities (i.e., groups) embedded in a single document. It operates at the individual document and cluster level.
- The sentence closest to the centroid (based on Euclidean distance) is selected as the representative sentence for that cluster.

- Implemented works:
    - Preprocesses the input text to get POS-tagged sentences.
    - Data Preprocessing:
        - Lowercasing
        - Stop Words Removal.
        - Lemmatization.
        - Tokenization.
        - POS Tagging.
    - Adjusts the number of clusters if there are fewer sentences than clusters.
    - Initializes a TF-IDF vectorizer with stop words removed.
    - Transforms the sentences into a TF-IDF matrix.
        - Matrix where each sentence is represented as a vector of TF-IDF scores
    - TF-IDF matrix is fed into the KMeans clustering algorithm.
        - Each sentence is assigned a cluster label.
    - Identifies the cluster labels for each sentence.
    - For each cluster:
        - Finds the indices of sentences in the cluster.
        - Calculates the centroid of the cluster.
        - Identifies the sentence closest to the centroid (representative sentence).
    - Collects the representative sentences from each cluster.
    - Joins and returns the representative sentences as a extractive summary.
    - Evaluating Summaries
        - ROUGE scores as performance metrics

Results

- Rule-based approach for extractive summarization was implemented and evaluated successfully.
- ROUGE1 (F-Measure) = 24.72.

ROUGE-1

- **Precision:** 0.3356
- **Recall:** 0.2516
- **F-Measure:** 0.2472

ROUGE-2

- **Precision:** 0.1298
- **Recall:** 0.0929
- **F-Measure:** 0.0915

ROUGE-L

- **Precision:** 0.2489
- **Recall:** 0.1786
- **F-Measure:** 0.1769

ROUGE-Lsum

- **Precision:** 0.2489
- **Recall:** 0.1784
- **F-Measure:** 0.1768

TESTING

- Implemented works, as follows, source code: src/interface.ipynb.
  - text summarization application (Abstractive text summarization).
    - using a fine-tuned transformer model (from method 2)
    - Gradio library - for web-based interface
  - Utilized for testing abstractive model's inference capabilities in real-time .

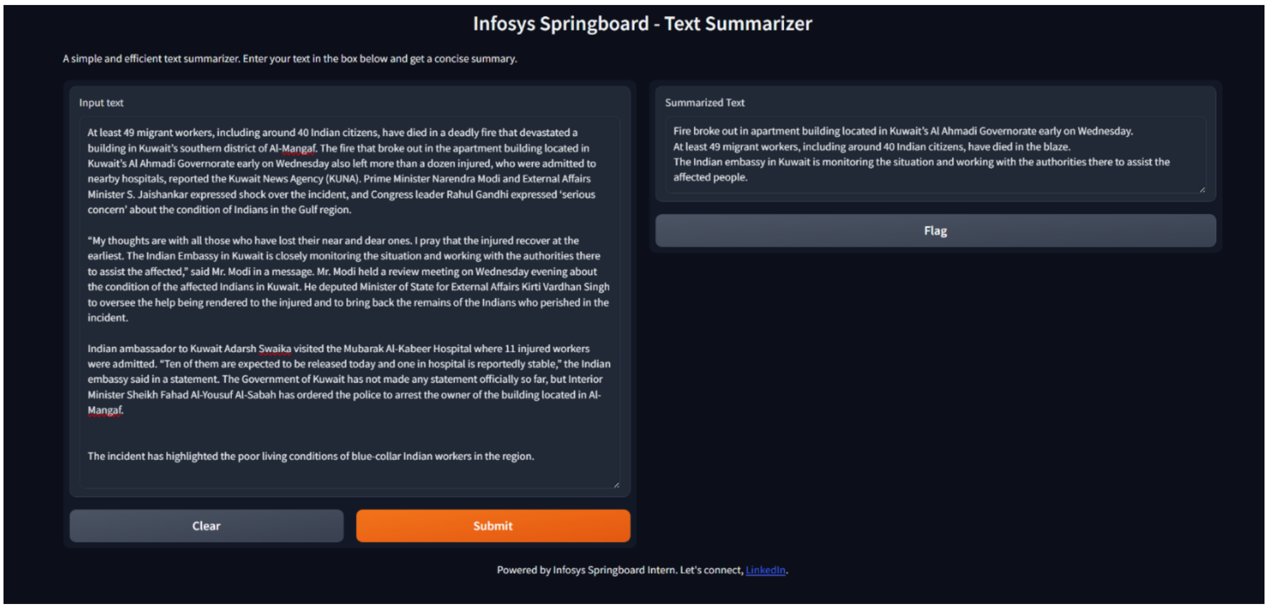


Fig.: Testing Interface – Using Gradio

DEPLOYMENT

Fig.: Roadmap for deployment.

APPLICATION

- File structure:

```
(infosys) mohan@LAPTOP-INPO8147:~$ tree infy/summarizer/
infy/summarizer/
├── Dockerfile
├── app.py
├── extractive_summary.py
├── extractors.py
├── requirements.txt
├── saved_model
│   ├── config.json
│   ├── generation_config.json
│   ├── merges.txt
│   ├── model.safetensors
│   ├── special_tokens_map.json
│   ├── tokenizer_config.json
│   └── vocab.json
└── templates
    └── index.html

2 directories, 13 files
```

Fig.: File structure for the application.

API ENDPOINTS

- Utilized the FastAPI framework to create a web application for text summarization.
    - source code: summarizer/app.py
- Developed to handles three main types of input: URLs, files, and direct 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:
    - Root Endpoint:
        - Serves an HTML template for the root URL.
    - Summarize URL:
        - Accepts a URL, extracts text from the URL.
        - generates both abstractive and extractive summaries.
        - Returns them as a JSON response.
    - Summarize File:
        - Accepts file uploads (PDF or DOCX), extracts text from the file,
        - generates summaries.
        - Returns them as a JSON response.
        - Unsupported file types return an error response.
    - 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.

EXTRACTOR MODULES

- Developed to extract text from various sources. Source code: summarizer/extractors.py
    - URLs, PDF files, and DOCX files.
- Text from URL:
    - Utilized BeatifulSoup for web crawling the web pages.
        - Extracts text from all paragraph with ‘<p>’ tags.
        - Joins the text of these paragraphs into a single string.
- Text from PDF:
    - Utilized fitz to open to open and extract text from pdf.
        - Iterates through each page of the PDF.
        - Extracts text from each page and concatenates it into a single string.
- Extract Text from DOCX:
    - Opens the DOCX file using Document.
        - Iterates through all paragraphs in the document.
        - Extracts text from each paragraph and joins them into a single string.
- These modular functions can be used independently or integrated into a larger application for processing different types of text sources.

EXTRACTIVE SUMMARY SCRIPT

- Implemented an extractive summarizer module as python script for application.
    - Source code: summarizer/extractive_summary.py .
- Developed to utilize the same programmatic approach of src/Extractive_Summarization.ipynb.
- This script is designed to:
    - Preprocess text.
    - Extract important features using TF-IDF.
    - Summarize the text by selecting representative sentences from different clusters.

USER INTERFACE

- Developed to provide a user-friendly interface for text summarization application.
    - Source code: summarizer/templates/index.html
- Designed forms for different input types (URL, file, and text).
- Utilized jQuery for event handling and AJAX.
    - Utilized AJAX to communicate with a backend server to perform the summarization.
    - Fetch API to POST server with request.
    - Implemented a loading spinner - during the request processing.
    - Displays results or errors based on the response.
- The results are displayed on the same page without reloading, providing a dynamic webpage.
- The design is enhanced with CSS for aesthetics and JavaScript for functionality.
- Added a footer with a short note.

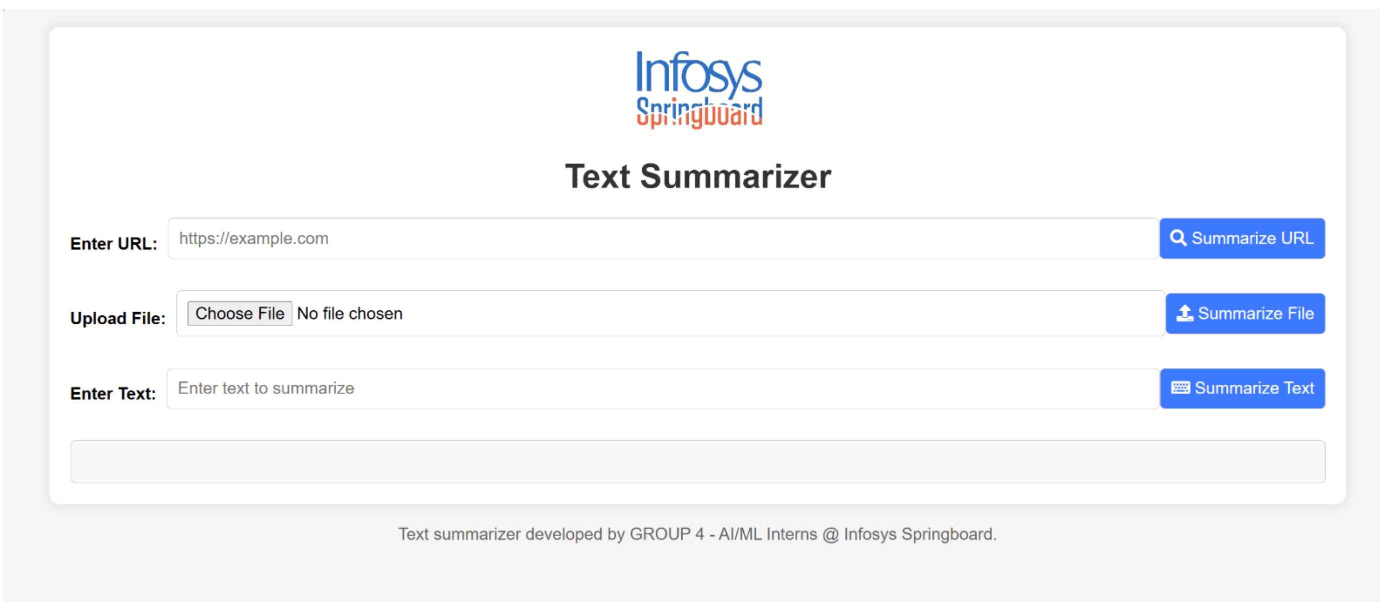


Fig.: Developed User Interface

CONTAINERIZATION

- Developed a Dockerfile to build the docker image for the FastAPI application.
  - Source code: summarizer/Dockerfile .
- 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.
  - Docker image

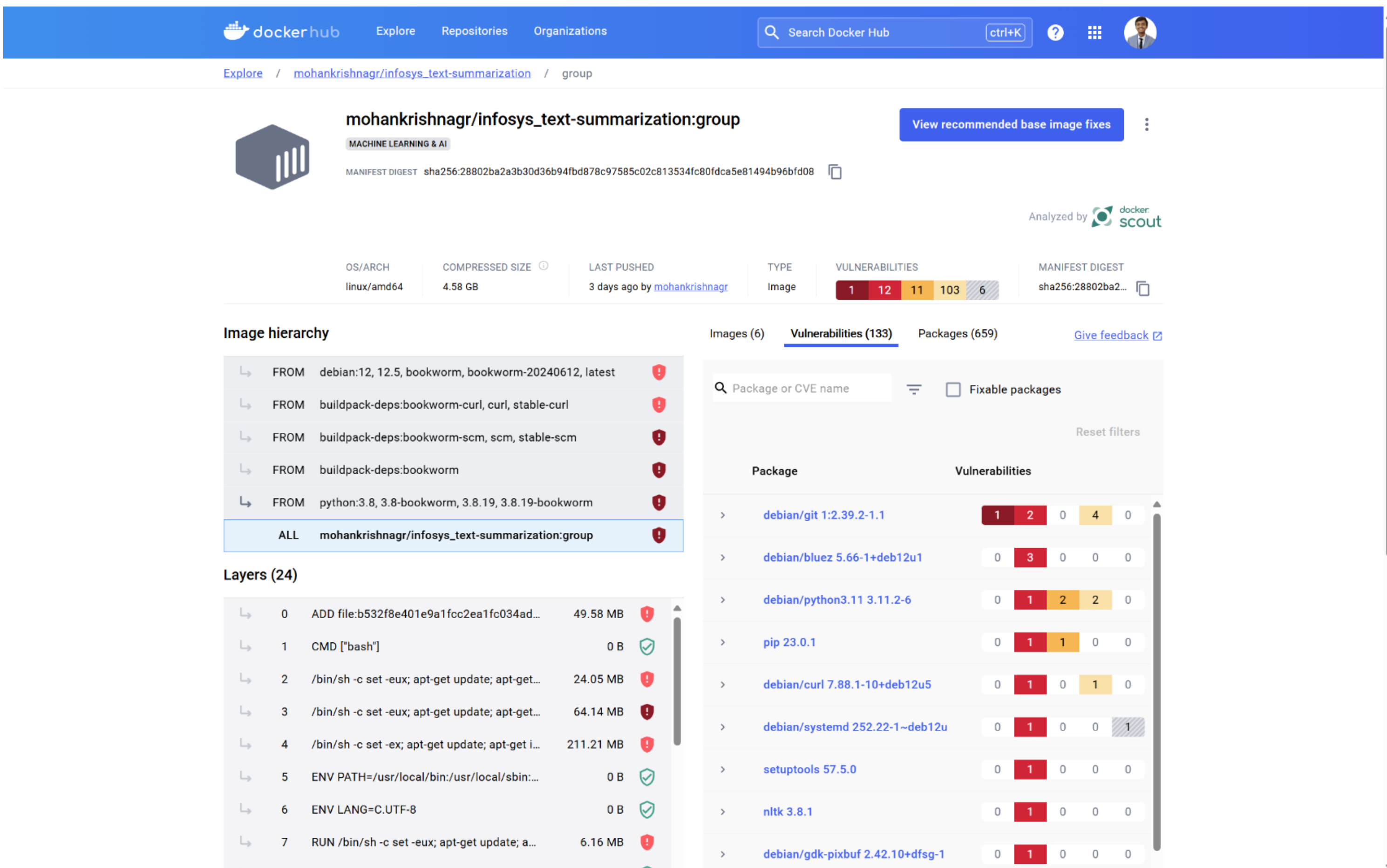


Fig.: Docker Hub Screenshot.

AZURE CONTAINER INSTANCE

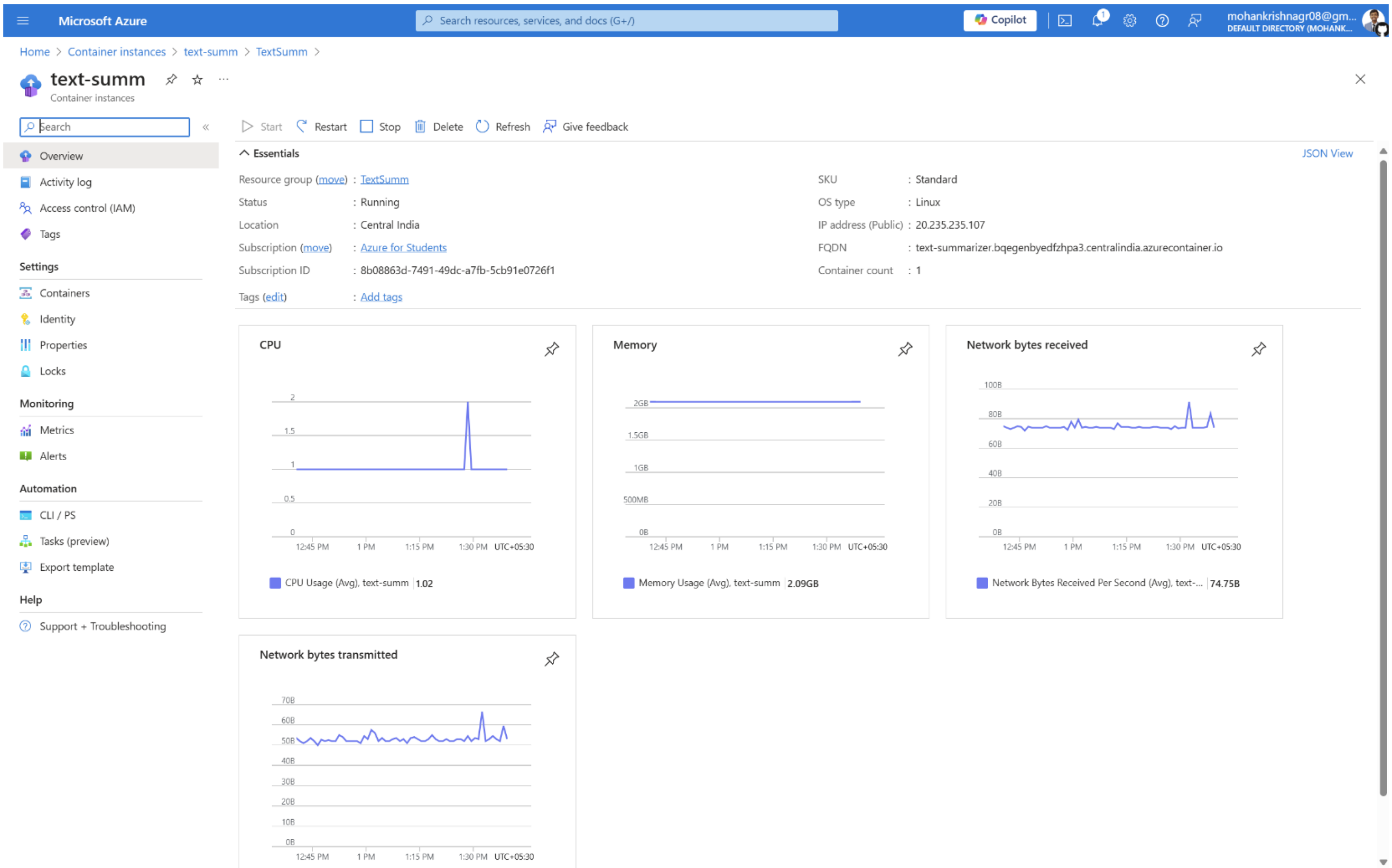


Fig.: Screenshot of Azure Container Instance Portal.

- Azure Container Instances (**ACI**) is a managed service that allows you to run containers directly on the Microsoft Azure public cloud, without requiring the use of virtual machines (VMs).
- In Azure, created a new public instance named 'TextSumm' with container name 'text-summ', under 'Free Trail' subscription. With free trail, a student user will have INR.16,700 as free credit.
- Hence, a comparatively large compute instance can be created in Azure. Created instance with 4 vCPU and 16 GiB Memory.
- Selected 'Other registry' for Image source (Docker Hub) and specified the image as: mohankrishnagr/Infosys_text-summarization:group
- Modified the networking so as to allow port 8000 as TCP connection, along with DNS label to avail a Fully Qualified Domain Name (FQDN).
- Created Role Based Access Control (RBAC), in order to integrate this with the GitHub Actions Workflow.

- Utilized Azure CLI to access the container instance – RBAC details & logs.

```
PS C:\Users\MOHAN> az container logs --name text-summ --resource-group TextSumm
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger.zip.
The argument `trust_remote_code` is to be used with Auto classes. It has no effect here and is ignored.
INFO:     Started server process [14589]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     Uvicorn running on http://0.0.0.0:8000 (Press CTRL+C to quit)
INFO:     10.92.0.5:65207 - "GET / HTTP/1.1" 200 OK
INFO:     10.92.0.5:65220 - "GET / HTTP/1.1" 200 OK
WARNING:  Invalid HTTP request received.
INFO:     10.92.0.6:59186 - "GET /favicon.ico HTTP/1.1" 404 Not Found
INFO:     10.92.0.6:55486 - "GET /summarize-text HTTP/1.1" 405 Method Not Allowed
INFO:     10.92.0.5:54436 - "HEAD / HTTP/1.1" 405 Method Not Allowed
INFO:     10.92.0.5:54459 - "HEAD / HTTP/1.1" 405 Method Not Allowed
INFO:     10.92.0.5:56189 - "GET /summarize-file HTTP/1.1" 405 Method Not Allowed
```

Fig.: Screenshot of ACI logs using Azure CLI.

- Attached a screenshot for the Azure Container Instance logs.

- Public IPv4 address & FQDN:

    - Text Summarizer (http://20.235.235.107:8000/)

    - Text Summarizer (http://text-summarizer.bqegenbyedfzhpa3.centralindia.azurecontainer.io:8000/)

CI/CD PIPELINE – AZURE

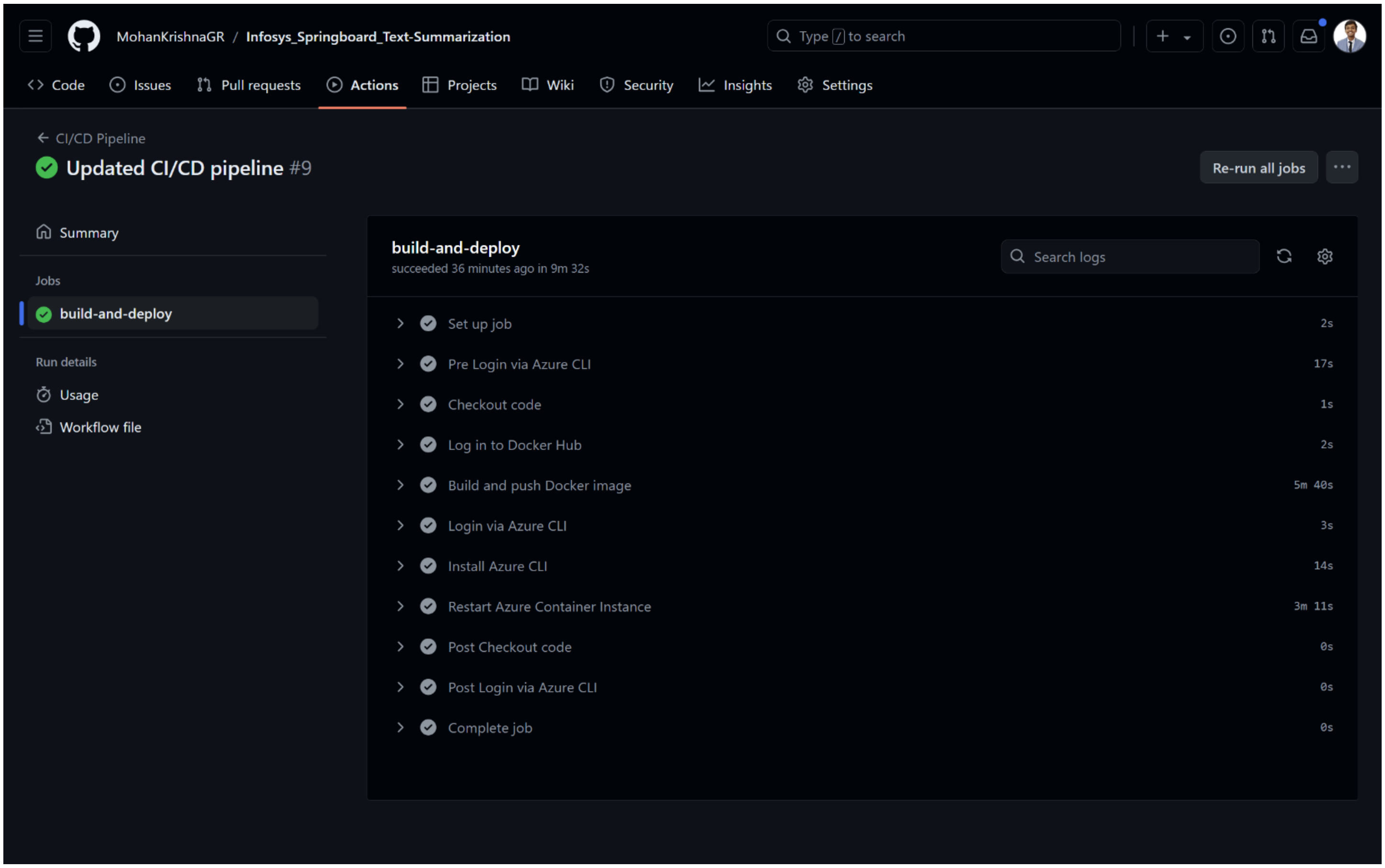


Fig.: Screenshot of Jobs in GitHub Actions.

- Created a GitHub Action Workflow to automate the process of deploying the containerized application using Azure Container Instance (ACI).
    - Source code: .github/workflows/azure.yml
- Even push event on the main branch triggers this workflow.
    - Checks out the repository code to the runner using GitHub Actions
    - Logs in to the Docker Hub.
    - Builds the new Docker Image and pushes it into Docker Hub.
    - Logins in to the Azure CLI using RBAC credentials, generated using Azure PowerShell commands.
    - Restarts the container instances with the new docker image.
- If model.safetensors file could be pushed into repo, then the building of image can also be automated in GitHub actions, for individual pro users in GitHub, it limits the maximum size of a file that can be uploaded.

    - Git LFS also has the maximum limit of 1 GiB of storage only.
    - Whereas, model.safetensors file size is 1.51 GB.
    - ISSSUE SOLVED:
        - Automated the entire CI/CD pipeline
        - Utilized ‘gdown’ to download the model.safetensors from Google Drive.
            - implemented in summarizer/download_model.py
        - Downloads the file only in the Dockerfile’s working directory.
            - Can bypass the GitHub file limits.

OBSERVATIONS

- The application deployed in Azure has proven to be the optimal choice comparatively to AWS.
    - The inferencing speed in Azure ACI is 10x higher that of AWS, given its computation instance, for *free trail*.
    - Thus, resulted in reduced response time by the server to the clients’ requests in production.
    - ACI has also a feature to get FQDN.
- Further enhancement in CI/CD pipeline could be considered in order to be more robust.
    - Need update to enhance security: http -> https

RESULTS

- All the approved modules have been implemented successfully.

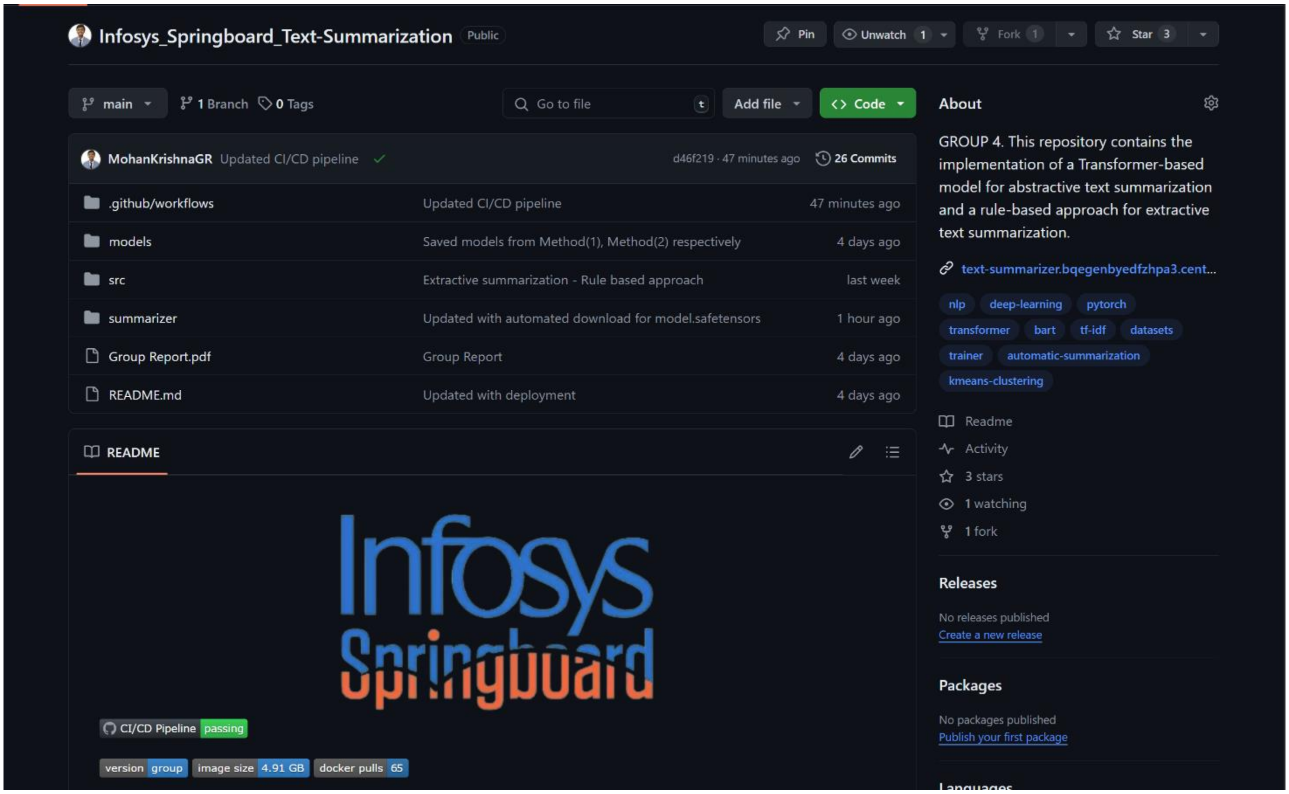


Fig.: Screenshot of GitHub repository.

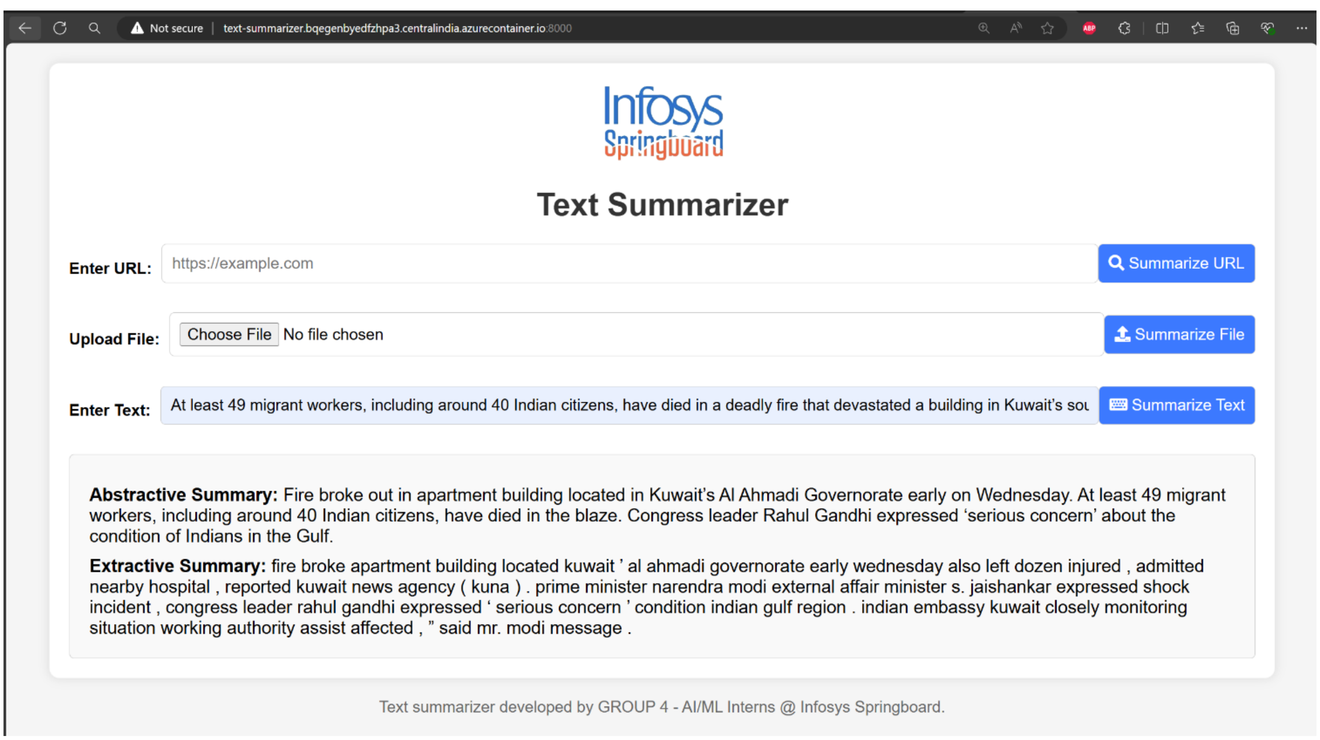


Fig.: Using Deployed Text Summarizer Application. (Direct Text)

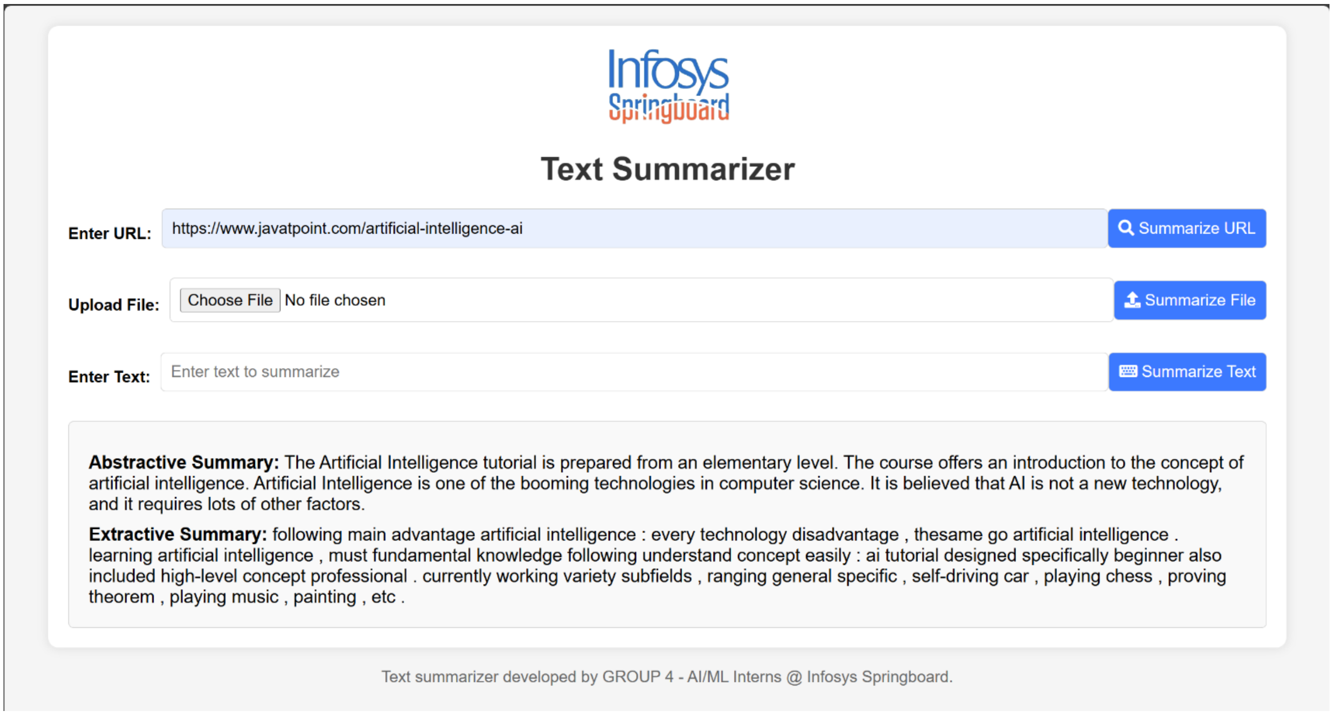


Fig.: Using Deployed Text Summarizer Application. (URL)

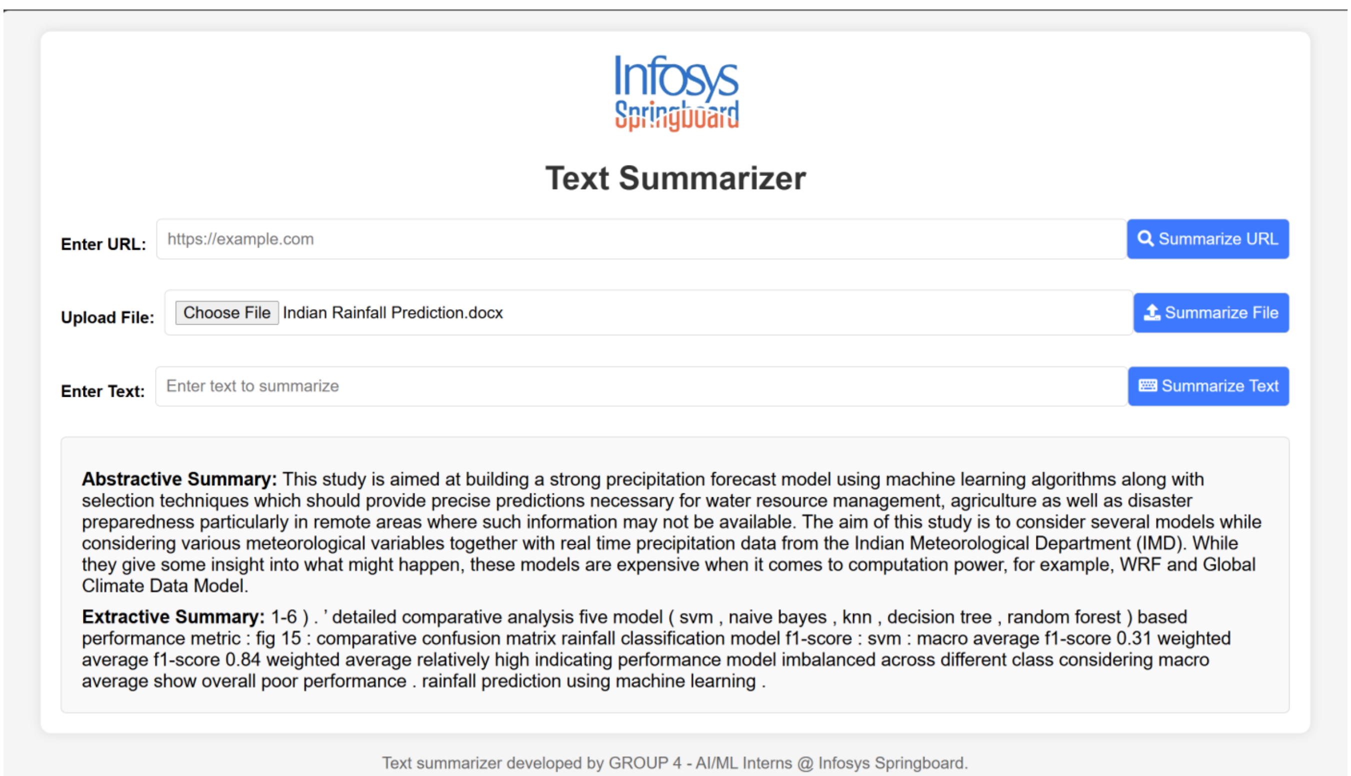


Fig.: Using Deployed Text Summarizer Application. (DOCX)

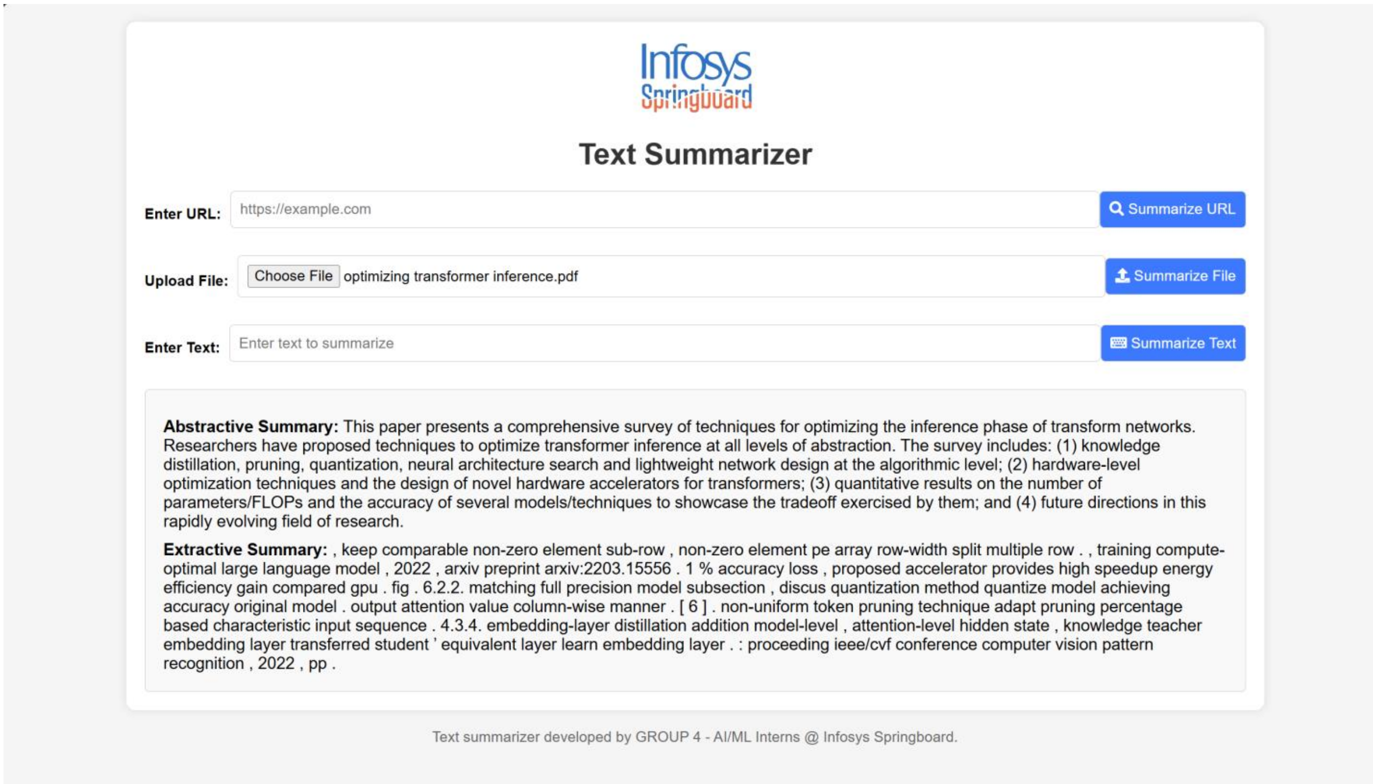


Fig.: Using Deployed Text Summarizer Application. (PDF)

CONCLUSION

- 'Text Summarization' Project achieved its goal by developing & deploying a robust 'Text Summarizer'
    - Processed and analyzed a combined dataset of over 5,64,522 text samples from various sources.
    - Implemented both abstractive and extractive methods, & took deep research on the different ways to optimally solve the sub-tasks.
    - Trained the model on the most robust and comprehensive – custom dataset.
    - Achieved Benchmark result in abstractive summarization model. (ROUGE1 = 61.32)
    - Showed a new paradigm in extractive summarization. (using TF-IDF and KMeans Clustering)
    - Developed the solution with regress check on performance metrics, to ensure quality & standard.
    - Created diversified API endpoints, to enable summarization for different text sources.
    - By deploying on Azure using Docker and GitHub Actions with a CI/CD pipeline.
        - Got 99.9% uptime, ensuring high availability and reliability.
        - Accessible via user-friendly interfaces and API endpoints.

FUTURE SCOPE

- SaaS Platform Development
    - API for Screen and Browser Summarization.
    - Platform Integration
        - Plugins - web browsers and document editors.
    - Domain-Specific Models
        - Training models using specialized datasets for industry-specific summaries.
    - User Customization.
        - Customizable Summaries.
            - Adjustable summary length & detail.
                - UI to change 'max_length' in model inferencing (model.generate()).
                - UI to adjust required number of clusters, in extractive model.
        - Multi-Language Support. (too out of scope)
- Continuous improvement steps will transform this project into a scalable, user-friendly summarization service. Will play as a value booster to both individual users and businesses.