Text Summarization App

AIDI 2004

Jatin Arora, Rishabh Malhotra,
Ritvij Sharma,

Group3

Table of Contents

Introduction

Brief overview of the text summarization problem

Text summarization is the process of automatically creating a concise and coherent version of a longer document or text. This problem is critical in the field of natural language processing (NLP) due to the massive amount of textual information available and the need to extract relevant information quickly and efficiently. There are two main types of text summarization: extractive and abstractive. Extractive summarization involves selecting important phrases or sentences from the original text, while abstractive summarization involves generating a summary that conveys the same meaning but with new and different words.

Importance of text summarization in natural language processing

Text summarization plays a crucial role in NLP, as it enables users to understand and consume large volumes of textual data more efficiently. Some applications of text summarization include news summarization, research paper summarization, legal document summarization, and customer reviews summarization. Text summarization can also aid in information retrieval, allowing search engines to display relevant summaries of search results, improving user experience and satisfaction.

Objective of the project

The objective of this project is to develop a text summarization application using a deep learning model trained on the CNN Daily Mail Dataset. The application includes a Flask API and front-end application, with the model deployed on Azure Machine Learning and the web app deployed on Heroku, connected via GitHub.

Background and Literature Review

Brief history of text summarization techniques

The history of text summarization techniques can be divided into three main eras: rule-based, statistical, and neural-based approaches. Rule-based methods involve using linguistic rules and heuristics to identify important sentences or phrases in the text. Statistical methods leverage statistical techniques like term frequency-inverse document frequency (TF-IDF) and latent semantic analysis (LSA) to determine the importance of sentences. The neural-based era emerged with the advent of deep learning techniques, such as recurrent neural networks (RNNs), long short-term memory (LSTM) networks, and more recently, transformer models.

Overview of machine learning and deep learning approaches to text summarization

Machine learning approaches to text summarization involve using algorithms like support vector machines (SVMs) and decision trees for extractive summarization. Deep learning approaches, on the other hand, primarily focus on abstractive summarization, using RNNs, LSTMs, and attention mechanisms to generate summaries. The most recent and popular deep learning approach is the transformer model, which has significantly improved the state-of-the-art in NLP tasks, including text summarization.

Introduction to transformer models and their impact on natural language processing

Transformer models are a type of deep learning architecture that rely on self-attention mechanisms to process and generate sequences of tokens. These models have become the standard for various NLP tasks due to their ability to handle long-range dependencies and parallelize computation. Some popular transformer models include BERT, GPT, and T5. These models have been pre-trained on massive text corpora and fine-tuned for various downstream tasks, resulting in significant improvements in performance compared to previous approaches.

Model Selection and Implementation

Description of the T5 model

T5 (Text-to-Text Transfer Transformer) is a transformer model designed for various NLP tasks, including text summarization. T5 is pre-trained on a large corpus of text and can be fine-tuned for specific tasks using a text-to-text approach, where both input and output are treated as sequences of tokens. This enables the model to generate abstractive summaries by understanding the context and relationships between tokens.

Justification for choosing the T5 model for this application

The T5 model was chosen for this application due to its state-of-the-art performance in text summarization tasks, as well as its versatility in handling various NLP tasks. Additionally, the T5 model's text-to-text framework simplifies the fine-tuning process for the target task of summarization.

Overview of the training process and fine-tuning the T5 model

The training process involves fine-tuning the pre-trained T5 model on the CNN Daily Mail Dataset. Fine-tuning requires adjusting the model's weights to adapt to the specific task of summarization. The dataset is preprocessed, tokenized, and split into training, validation, and testing sets. The model is then trained using the Seq2SeqTrainer from the Hugging Face Transformers library, with the evaluation performed at the end of each epoch.

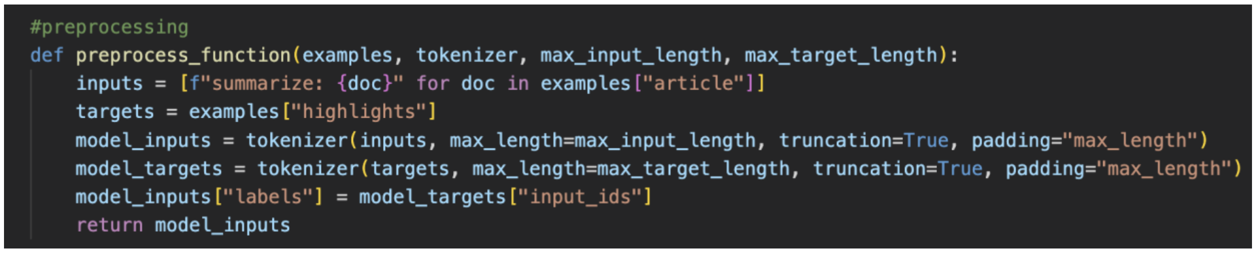
Dataset and Exploratory Data Analysis

Description of the dataset used for training and validation

The CNN Daily Mail Dataset is a large-scale dataset containing news articles and their corresponding highlights (summaries). This dataset is commonly used for training and evaluating text summarization models. It consists of over 300,000 training examples and approximately 13,000 validation examples.

Data preprocessing and tokenization steps

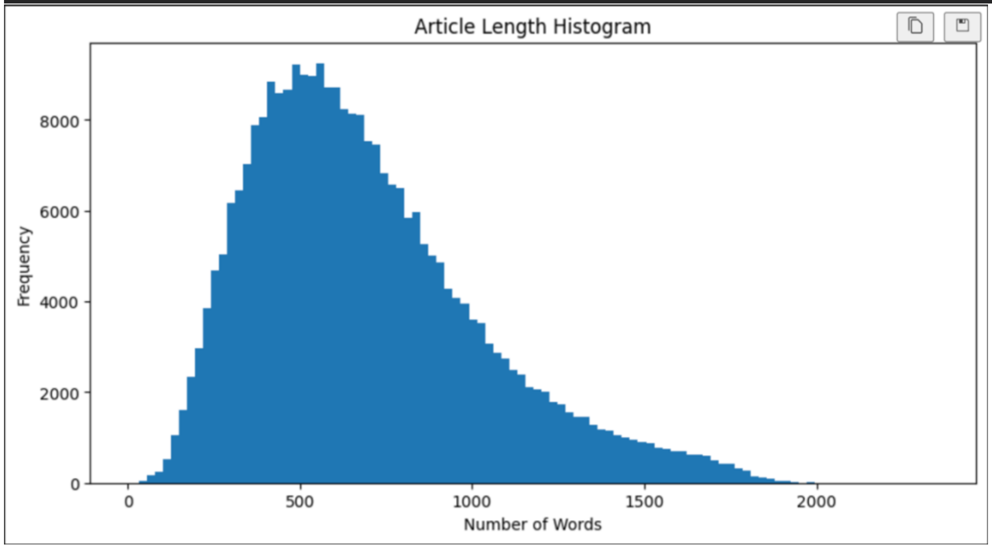
Preprocessing is a crucial step in the machine learning pipeline that involves cleaning and transforming raw data into a format suitable for model input. In this project, we preprocess the text by tokenizing the articles and highlights using the T5 tokenizer. Additionally, we truncate and pad sequences to a fixed length.



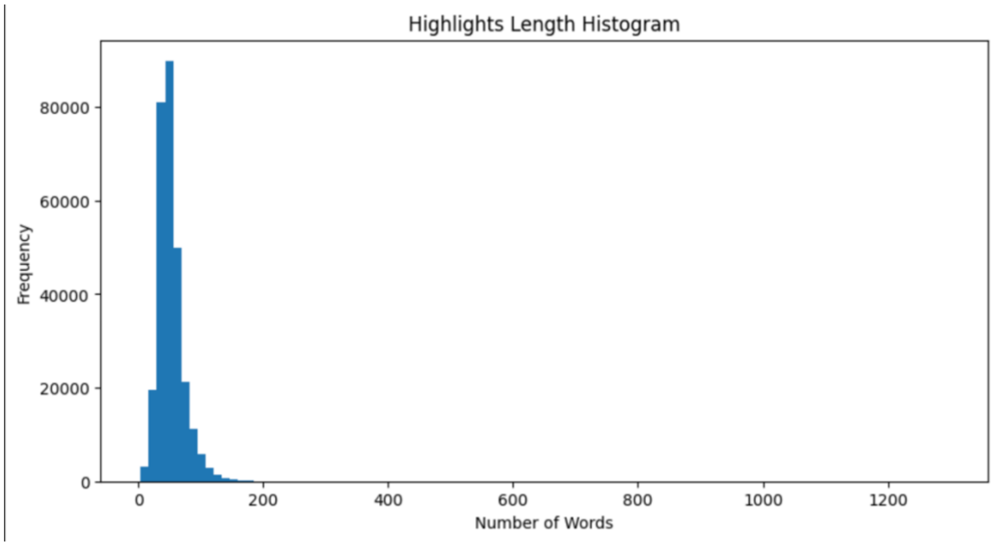
```python
#preprocessing
def preprocess_function(examples, tokenizer, max_input_length, max_target_length):
    inputs = [f"summarize: {doc}" for doc in examples["article"]]
    targets = examples["highlights"]
    model_inputs = tokenizer(inputs, max_length=max_input_length, truncation=True, padding="max_length")
    model_targets = tokenizer(targets, max_length=max_target_length, truncation=True, padding="max_length")
    model_inputs["labels"] = model_targets["input_ids"]
    return model_inputs
```

Exploratory Data Analysis

```python
def plot_article_length_histogram(dataset, bins=100):
    article_lengths = [len(doc.split()) for doc in dataset["article"]]
    plt.figure(figsize=(10, 5))
    plt.hist(article_lengths, bins=bins)
    plt.title("Article Length Histogram")
    plt.xlabel("Number of Words")
    plt.ylabel("Frequency")
    plt.show()
```

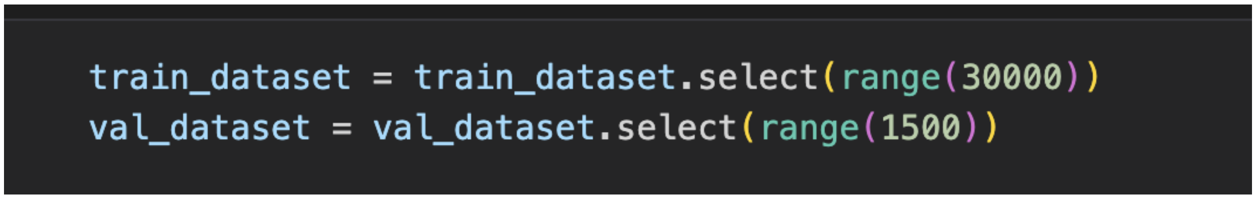


```python
def plot_highlights_length_histogram(dataset, bins=100):
    highlights_lengths = [len(doc.split()) for doc in dataset["highlights"]]
    plt.figure(figsize=(10, 5))
    plt.hist(highlights_lengths, bins=bins)
    plt.title("Highlights Length Histogram")
    plt.xlabel("Number of Words")
    plt.ylabel("Frequency")
    plt.show()
```



Dataset splitting for training, validation, and testing

The dataset is split into training, validation, and testing sets. In this project, 30,000 examples are selected for training, and 1,500 examples are selected for validation. The training and validation datasets are then tokenized and preprocessed using the T5 tokenizer.

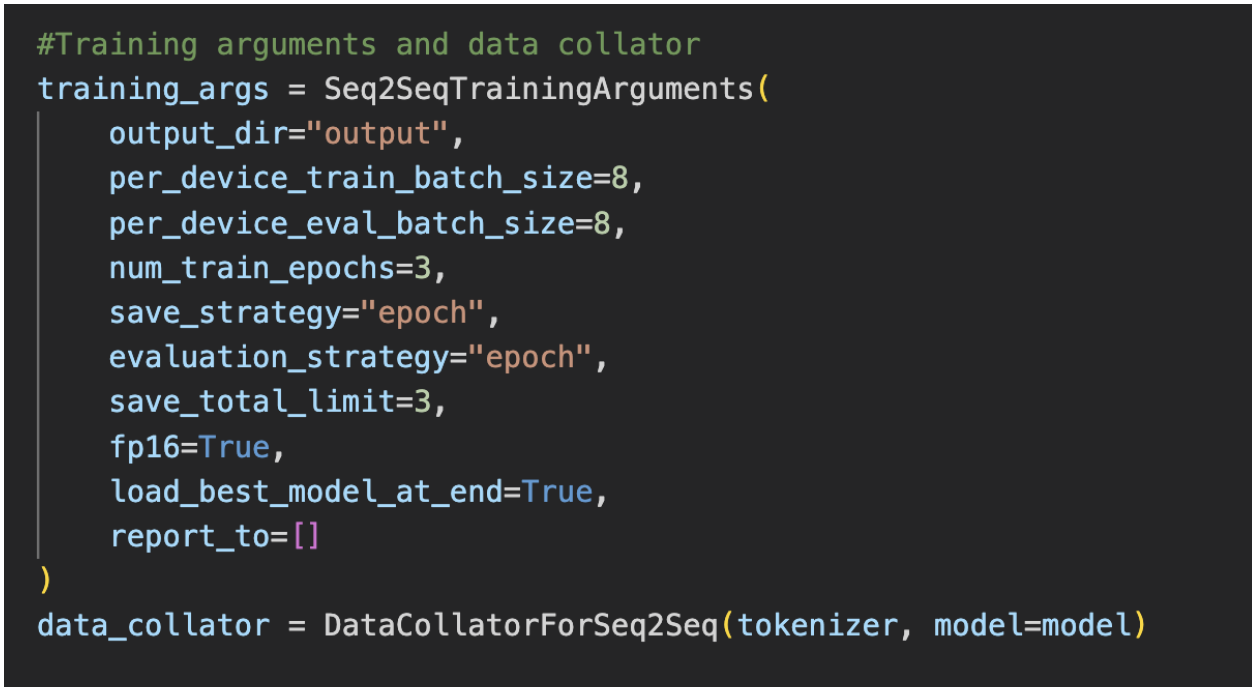


```
train_dataset = train_dataset.select(range(30000))
val_dataset = val_dataset.select(range(1500))
```

Training and Evaluation

Hyperparameter selection and tuning

Hyperparameters are adjustable parameters that determine the model's performance and training dynamics. Some important hyperparameters in this project include batch size, number of training epochs and more. These hyperparameters are tuned to achieve the best model performance on the validation dataset.



```
#Training arguments and data collator
training_args = Seq2SeqTrainingArguments(
    output_dir="output",
    per_device_train_batch_size=8,
    per_device_eval_batch_size=8,
    num_train_epochs=3,
    save_strategy="epoch",
    evaluation_strategy="epoch",
    save_total_limit=3,
    fp16=True,
    load_best_model_at_end=True,
    report_to=[]
)
data_collator = DataCollatorForSeq2Seq(tokenizer, model=model)
```

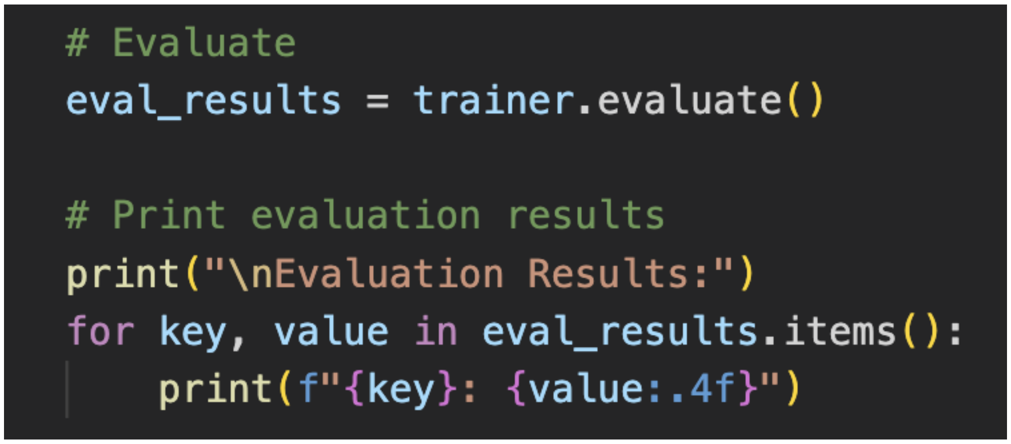
Training setup and resources (e.g., hardware, software, libraries)

The training setup includes the use of the Hugging Face Transformers library, which provides pre-built models, tokenizers, and training utilities. The training process is conducted on a GPU-enabled environment from Vast.AI to accelerate computation and reduce training time.

Evaluation metrics and results

To evaluate the performance of our text summarization model, we use the evaluation loss, which is the cross-entropy loss calculated between the predicted summaries and the reference summaries. Lower evaluation loss values indicate better performance of the model in generating summaries that closely resemble the reference summaries.

The evaluation results will display the evaluation loss, which serves as an indicator of how well the T5-small model is performing in solving the text summarization problem. By comparing this value with those of other models, we can determine the effectiveness of our T5-small model for this specific task.



```python
# Evaluate
eval_results = trainer.evaluate()

# Print evaluation results
print("\nEvaluation Results:")
for key, value in eval_results.items():
    print(f"{key}: {value:.4f}")
```

Flask Web Application

Description of the Flask framework and its advantages for web applications

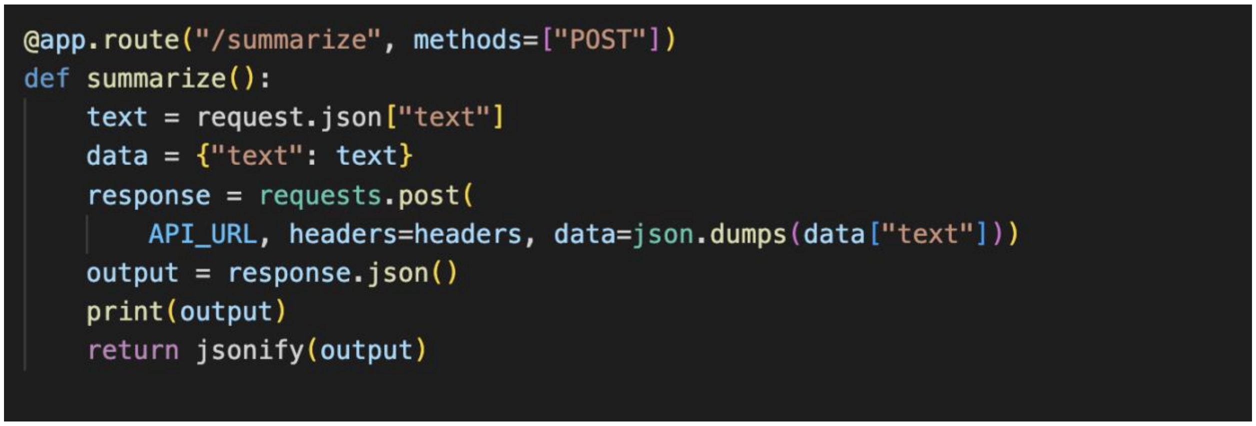
Flask is a lightweight Python web framework that allows for the rapid development and deployment of web applications. Its simplicity, flexibility, and extensive library support make it an ideal choice for creating web applications that interface with machine learning models.

Design and implementation of the Flask application

The Flask application includes a user interface for users to input text and receive generated summaries from the T5 model. The application handles user inputs, processes them using the T5 model, and returns the generated summaries. The user interface is designed to be intuitive and user-friendly, allowing users to easily interact with the text summarization system.

Handling user inputs and generating summaries using the T5 model

When a user submits a text input through the web application, the Flask API processes the request, tokenizes the input text using the T5 tokenizer, and passes it to the T5 model for summary generation. The generated summary is then returned to the user through the web interface.



```
@app.route("/summarize", methods=["POST"])
def summarize():
    text = request.json["text"]
    data = {"text": text}
    response = requests.post(
        API_URL, headers=headers, data=json.dumps(data["text"]))
    output = response.json()
    print(output)
    return jsonify(output)
```

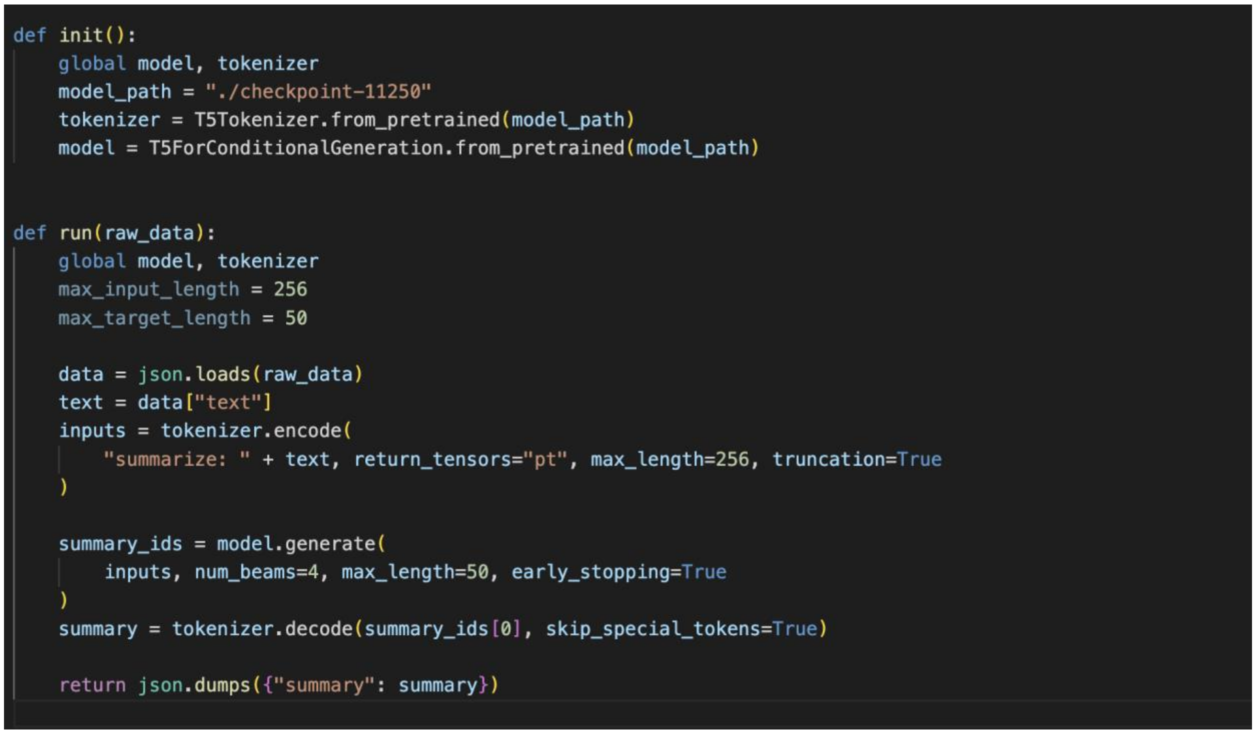
Deployment, Integration and Dockerization

Overview of deployment platforms (Heroku, Microsoft Azure)

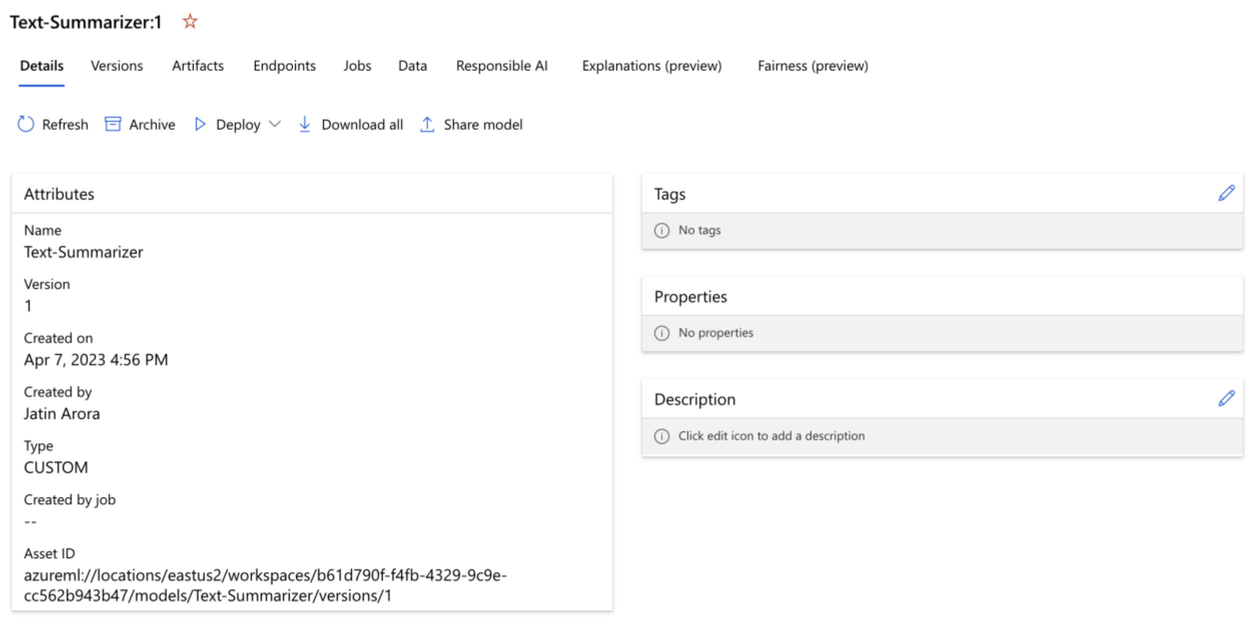
Heroku and Microsoft Azure are popular cloud-based platforms for deploying web applications and machine learning models. Heroku simplifies the deployment of web applications by providing a platform-as-a-service (PaaS) solution, while Azure Machine Learning offers a comprehensive platform for managing, deploying, and monitoring machine learning models.

Deployment of the model on Azure ML

The T5 model is deployed on Azure Machine Learning using the Azure ML SDK. The SDK provides tools to register the model, create a scoring script, define a deployment configuration, and deploy the model as a web service. Once deployed, the model can be accessed through a REST API.



```python
def init():
    global model, tokenizer
    model_path = "./checkpoint-11250"
    tokenizer = T5Tokenizer.from_pretrained(model_path)
    model = T5ForConditionalGeneration.from_pretrained(model_path)


def run(raw_data):
    global model, tokenizer
    max_input_length = 256
    max_target_length = 50

    data = json.loads(raw_data)
    text = data["text"]
    inputs = tokenizer.encode(
        "summarize: " + text, return_tensors="pt", max_length=256, truncation=True
    )

    summary_ids = model.generate(
        inputs, num_beams=4, max_length=50, early_stopping=True
    )
    summary = tokenizer.decode(summary_ids[0], skip_special_tokens=True)

    return json.dumps({"summary": summary})
```


Text-Summarizer:1

Attributes

Name
Text-Summarizer

Version
1

Created on
Apr 7, 2023 4:56 PM

Created by
Jatin Arora

Type
CUSTOM

Created by job
--

Asset ID
azureml://locations/eastus2/workspaces/b61d790f-f4fb-4329-9c9e-cc562b943b47/models/Text-Summarizer/versions/1

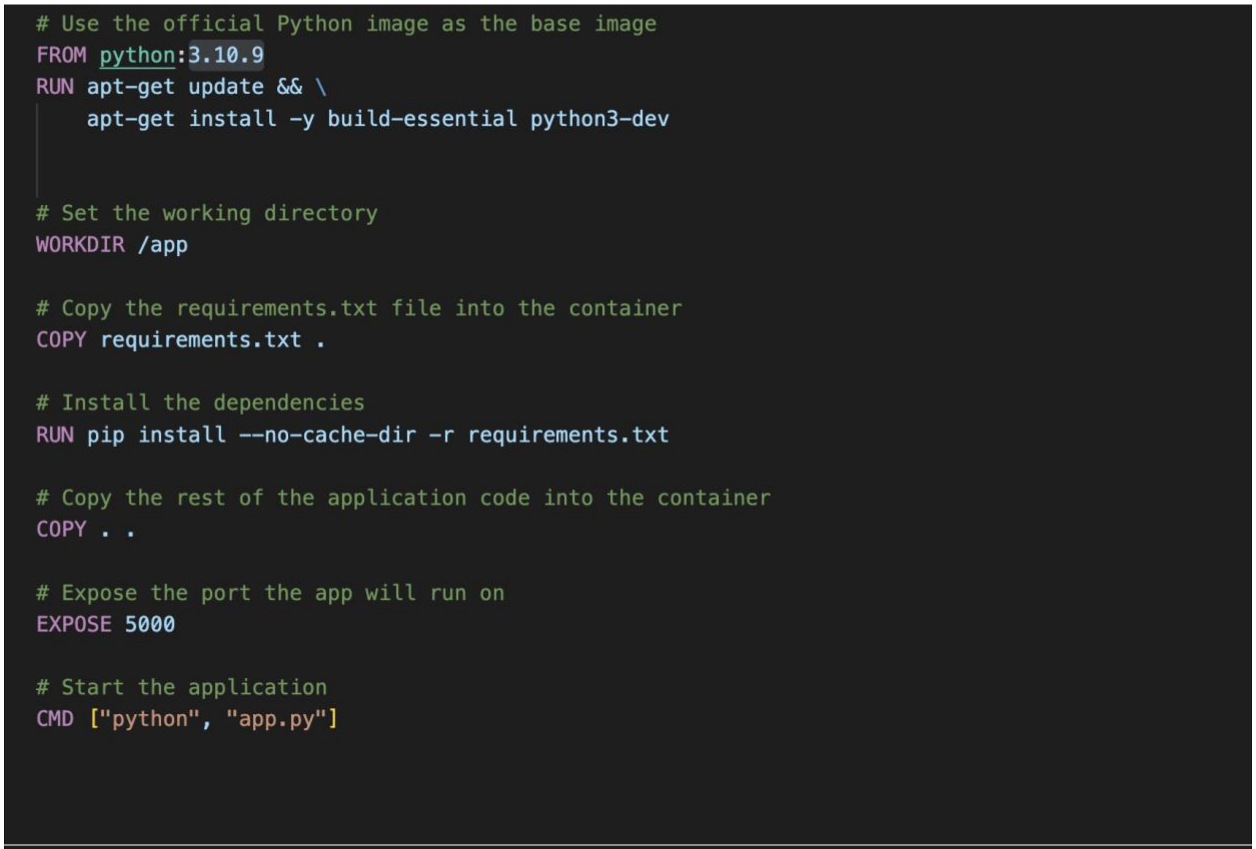
Tags
No tags

Properties
No properties

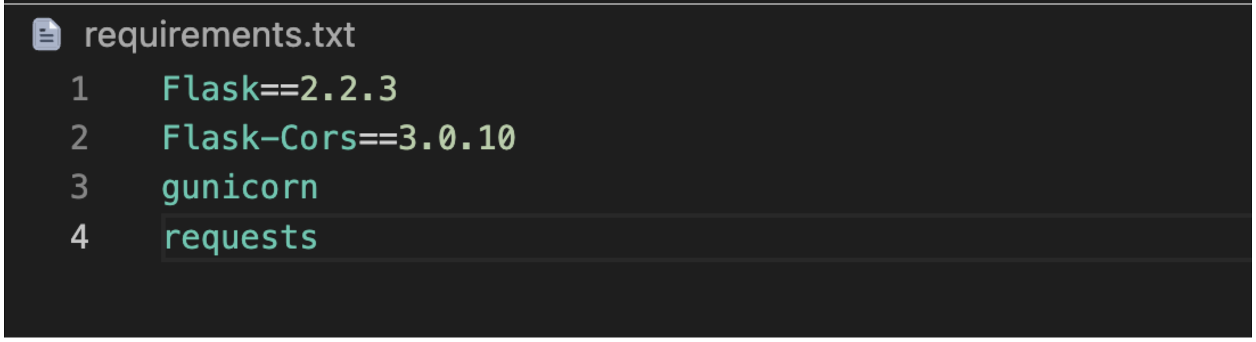
Description
Click edit icon to add a description

Dockerization of the Flask API

Docker is an open-source platform that allows developers to automate the deployment of applications inside lightweight, portable containers. Containers are standalone, executable software packages that include all the necessary components, such as libraries, dependencies, and runtime, to run an application consistently across different environments.



```
# Use the official Python image as the base image
FROM python:3.10.9
RUN apt-get update && \
    apt-get install -y build-essential python3-dev


# Set the working directory
WORKDIR /app

# Copy the requirements.txt file into the container
COPY requirements.txt .

# Install the dependencies
RUN pip install --no-cache-dir -r requirements.txt

# Copy the rest of the application code into the container
COPY . .

# Expose the port the app will run on
EXPOSE 5000

# Start the application
CMD ["python", "app.py"]
```



requirements.txt

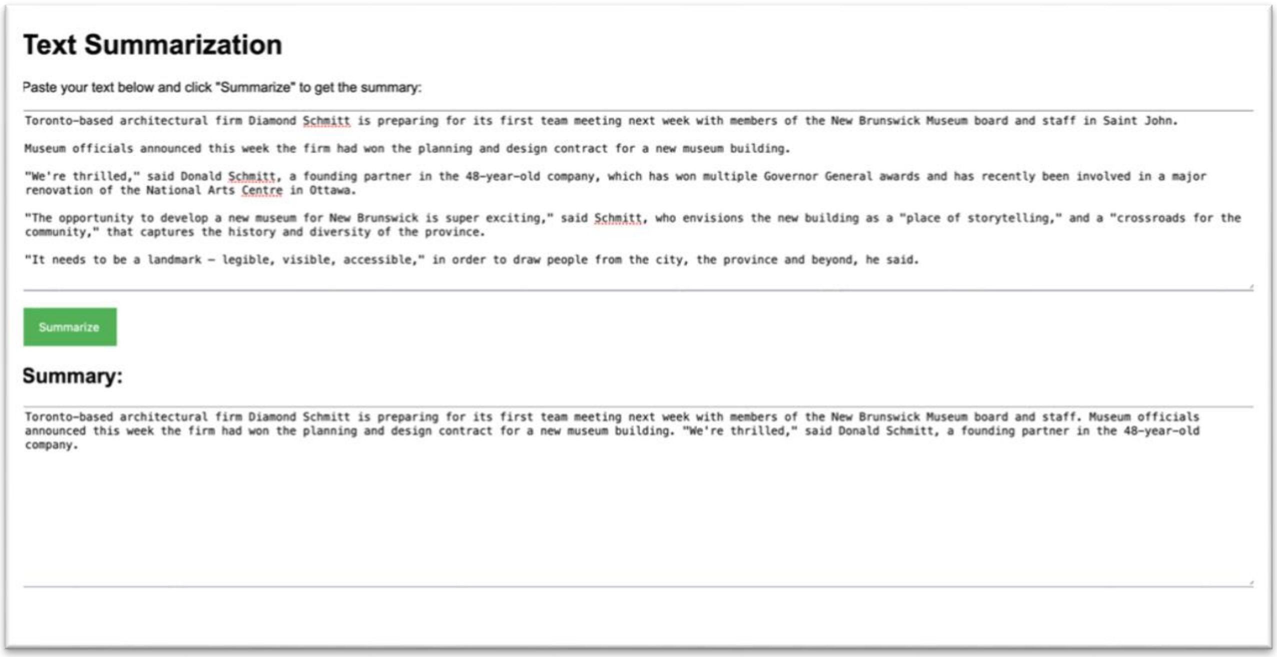
```
Flask==2.2.3
Flask-Cors==3.0.10
gunicorn
requests
```

Deployment of the Flask application on Heroku

The Flask web application is deployed on Heroku by connecting it with a GitHub repository. This allows for automatic deployment of the application whenever changes are pushed to the repository. Heroku provides a simple, streamlined process for deploying and managing web applications.

Integration of the Flask application with the model deployed on Azure ML

The Flask application is integrated with the T5 model deployed on Azure ML by making API calls to the model's endpoint. When the Flask API receives a user request, it forwards the request to the Azure ML endpoint, which processes the input text and returns the generated summary. The summary is then displayed to the user through the Flask web interface.



Text Summarization

Paste your text below and click "Summarize" to get the summary:

```
Toronto-based architectural firm Diamond Schmitt is preparing for its first team meeting next week with members of the New Brunswick Museum board and staff in Saint John.

Museum officials announced this week the firm had won the planning and design contract for a new museum building.

"We're thrilled," said Donald Schmitt, a founding partner in the 48-year-old company, which has won multiple Governor General awards and has recently been involved in a major renovation of the National Arts Centre in Ottawa.

"The opportunity to develop a new museum for New Brunswick is super exciting," said Schmitt, who envisions the new building as a "place of storytelling," and a "crossroads for the community," that captures the history and diversity of the province.

"It needs to be a landmark – legible, visible, accessible," in order to draw people from the city, the province and beyond, he said.
```

Summary:

```
Toronto-based architectural firm Diamond Schmitt is preparing for its first team meeting next week with members of the New Brunswick Museum board and staff. Museum officials announced this week the firm had won the planning and design contract for a new museum building. "We're thrilled," said Donald Schmitt, a founding partner in the 48-year-old company.
```

Conclusion

Summary of the project's achievements

This project successfully developed a text summarization application using the T5 model trained on the CNN Daily Mail Dataset. The application includes a Flask API and front-end interface, with the model deployed on Azure Machine Learning and the web app deployed on Heroku. The application demonstrates good performance in generating abstractive summaries and offers a user-friendly interface for interacting with the summarization system.

Reflection on the project's impact and potential use cases

The project showcases the power of deep learning techniques, specifically transformer models, in solving complex NLP tasks such as text summarization. The developed application can be utilized in various contexts, including news summarization, research paper summarization, and legal document summarization, helping users efficiently consume and understand large volumes of textual information.

Resources

Github Link
https://github.com/jarora-dev/Text-Summarizer-Web-Application

Heroku Link
https://text-summarizer-app.herokuapp.com/

Reference

Text Summarization Techniques: A Brief Survey: https://arxiv.org/pdf/1707.02268.pdf

Attention Is All You Need: https://arxiv.org/pdf/1706.03762.pdf

T5 paper: https://arxiv.org/pdf/1910.10683.pdf

Hugging Face T5 model documentation: https://huggingface.co/transformers/model_doc/t5.html

CNN/Daily Mail Dataset: https://huggingface.co/datasets/cnn_dailymail

Hugging Face's Seq2SeqTrainer documentation:
https://huggingface.co/transformers/main_classes/trainer.html#transformers.Seq2SeqTrainer

Flask official documentation: https://flask.palletsprojects.com/en/2.1.x/

Azure Machine Learning documentation: https://docs.microsoft.com/en-us/azure/machine-learning/

Heroku documentation: https://devcenter.heroku.com/

BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension: https://arxiv.org/pdf/1910.13461.pdf

PEGASUS: Pre-training with Extracted Gap-sentences for Abstractive Summarization:
https://arxiv.org/pdf/1912.08777.pdf