CSE 676 Deep Learning – Final Project Report

Optimizing Text Summarization with Deep Learning Models and NLP Techniques

Vavila S S V S Siri Sudheeksha (svavila2-50560556)
Divija Ankam (divijaan- 50542158)
Roshini Konjeti (rkonjeti- 50560111)

1. Overview/Problem statement

Problem Statement:

The goal of the project is to achieve automatic text summarizing for an overloaded information of text/document, the process involves in reducing large amounts of text into brief summaries without sacrificing any important information. In this digital age there is no time to take overloaded information. Our application helps in summarizing and saving time for users to understand the context. This will help in support for Decision Making in many field ex: Employee performance summarization, food reviews etc.

Objectives:

The main objectives are to improve the text summarizing model performance on particular datasets, increase the correctness and coherence of the generated summary for a textual document, and enable the models to be customized for other kinds of text data.

2. Background and real world deep learning application

In Natural Language processing with many real world applications in this domain. Our project aims to formulate and solve the problem of finding the best summary for any use case using state of the art deep learning models including Bert2Bert, Facebook Bart base, Google T5 base.

Real World Problem:

In this modern digital era, everything is data and one of the main data streams is text data that is generated on a daily basis. This poses a significant challenge in information processing. Manual summarization of heavy documentations, articles, scripts etc are impractical. Automatic text summarization solves this hurdle providing a platform to provide critical information in large text format and get the efficient summaries out of them with no loss in important information. Various domains where this is highly necessary are:

- **New Industry**: Daily news articles are summarized to provide user with quick insights
- **Healthcare**: Patient records for quick review by medical staff and professionals

- **Research**: All the lengthy research papers need a summary for the developers to work effectively.

3. Dataset

This dataset contains the data on daily news. It's a .tgz file which contains story files. Each story file is having news data and having its summary at the end of the file. The summary is identified by a delimiter as "@highlight". This data can be given to a model to train, understand the news and learn summary text.

Dataset reference: https://github.com/abisee/cnn-dailymail?tab=readme-ov-file

4. Libraries used

1. Torch.nn and torch.optim: for neural networks using pytorch
2. Random: generate random numbers
3. torchtext.data.utils.get_tokenizer: create tokens
4. torchtext.vocab.build_vocab_from_iterator: build vocabulary from an interator.
5. Pandas: for data manipulation and analysis.
6. os: to interact with os
7. re: regular expressions
8. beautiful soap: to parse html and xml documents
9. nltk: Natural Language Toolkit, for natural language processing
10. nltk.corpus.stopwords:list of stopwords
11. tarfile: to access .tgz file
12. sklearn.model_selection import train_test_split: to split data into train, test and validation
13. matplotlib, seaborn for visualization
14. PegasusTokenizer, PegasusForConditionalGeneration:

    used to tokenize specifically for Pegasus model.
15. BartTokenizer, BartForConditionalGeneration: used to tokenize specifically for Bart model
16. T5Tokenizer, T5ForConditionalGeneration: used to tokenize specifically for T5 model.

17. Trainer, TrainingArguments:

    Class from hugging face transformers library for training model & defining parameters.

18. Dataset, DatasetDict: class from hugging face for creating and manipulating datasets

19. from collections import Counter: to count the frequency of an element

20. ROUGE and BERT_score for evaluating scores

5. Data preprocessing/ Data visualization

Following are the steps to load the data which needs to be done before pre processing

(a) Open the .tgz file and extract all files in “dailymail_stories” directory

```
import tarfile
import os

tgz_file_path = '/kaggle/input/dailymail/dailymail_stories.tgz'
extract_dir = 'dailymail_stories'

os.makedirs(extract_dir, exist_ok=True)

with tarfile.open(tgz_file_path, "r:gz") as tar:
    tar.extractall(path=extract_dir)

print("Extraction completed!")
```

```
Extraction completed!
```

(b) Check if cuda is available or not. If available use cuda using “.to(device)” method

```
import torch

device = 'cuda' if torch.cuda.is_available() else 'cpu'
print(device)
```

```
cuda
```

(c) Read each story file and store all the lines in a list called “lines”. Iterate through each line. If line==”highlight”, the text below “@highlight” should be stored in highlight_lines, else store the lines in “story_lines”.

```
def read_story_file(file_path):
    with open(file_path, 'r', encoding='utf-8') as f:
        lines = f.readlines()

    story_lines = []
    highlight_lines = []
    reading_highlights = False
    for line in lines:
        if line.strip() == "@highlight":
            reading_highlights = True
            continue
        if reading_highlights:
            highlight_lines.append(line.strip())
        else:
            story_lines.append(line.strip())

    story = ' '.join(story_lines)
    highlights = ' '.join(highlight_lines)
    return story, highlights
```

(d) Loads all story files from the data path. read_story_file() method is called on each story file.

```
def load_dataset(data_path):
    articles = []
    highlights = []
    for filename in os.listdir(data_path):
        if filename.endswith('.story'):
            story, highlight = read_story_file(os.path.join(data_path, filename))
            articles.append(story)
            highlights.append(highlight)
    return articles, highlights
```

(e) Data pre processing:

[1] If the length of “highlight” is less than 4 words, then we can ignore the corresponding story file. Zip method is used to make sure that each article is matched with each highlight.

```
filtered_articles = []
filtered_highlights = []
for article, highlight in zip(data_articles, data_highlights):
    if len(highlight.split()) >= 4:
        filtered_articles.append(article)
        filtered_highlights.append(highlight)

data_articles = filtered_articles
data_highlights = filtered_highlights
```

[2] Since the data size is huge, consider only 5000 rows

```
data_articles = data_articles[:5000]
len(data_articles)
```

```
5000
```

+ Code + Markdown

```
data_highlights = data_highlights[:5000]
len(data_highlights)
```

```
5000
```

[3] Split the data into train, validation and test datasets: 80:10:10 ratio

```
from sklearn.model_selection import train_test_split

train_articles, temp_articles, train_highlights, temp_highlights = train_test_split(
    data_articles, data_highlights, test_size=0.2, random_state=42)

val_articles, test_articles, val_highlights, test_highlights = train_test_split(
    temp_articles, temp_highlights, test_size=0.5, random_state=42)

print(f"Train set size: {len(train_articles)}")
print(f"Validation set size: {len(val_articles)}")
print(f"Test set size: {len(test_articles)}")
```

```
Train set size: 4000
Validation set size: 500
Test set size: 500
```

[4] Cleaning data

```
nltk.download('stopwords')

stop_words = set(stopwords.words('english'))

def text_cleaner(text):
    text_lowercased = text.lower()
    text_cleaned = BeautifulSoup(text_lowercased, "lxml").text
    text_cleaned = re.sub(r'\([^)]*\)', '', text_cleaned)
    text_cleaned = re.sub('"','', text_cleaned)
    text_cleaned = ' '.join([contraction_mapping[t] if t in contraction_mapping else t for t in text_cleaned.sp
    text_cleaned = re.sub(r"'s\b","",text_cleaned)
    text_cleaned = re.sub("[^a-zA-Z]", " ", text_cleaned)
    text_cleaned = re.sub('[m]{2,}', 'mm', text_cleaned)
    return text_cleaned
```

->Store all the stopwords downloaded from nltk in a set called “stop_words”.

->convert all texts into lower case.

->use beautiful soap to parse and remove any “lxml” tags

->use regular expressions to remove \([^)]*\)”””’

->use contraction mapping dictionary to convert words like “ain't": "is not"

->removes possessive suffixes (ex: “s” from the word suffix)

->any char apart from letters, will be replaced into spaces

->collapses consecutive m’s into “mm”

[5] Removing stop words:stopwords are removed from article but not from highlights/summary. Because every word in highlights is important for the model to get trained.

```
def articles_clean(article) :
    clean_article = text_cleaner(article)
    word_tokens = [w for w in clean_article.split() if not w in stop_words]
    article_words = [i for i in word_tokens if len(i) > 1]
    return (" ".join(article_words)).strip()

def summary_clean(summary) :
    clean_summary = text_cleaner(summary)
    word_tokens = clean_summary.split()
    summary_words = [i for i in word_tokens if len(i) > 1]
    return (" ".join(summary_words)).strip()
```

[6] Vocabulary: to find the most frequently used words and print their count of occurrence.

```
from collections import Counter
tokenized_highlights = [word_tokenize(highlight) for highlight in cleaned_train_highlights]
vocab = Counter([word for tokens in tokenized_highlights for word in tokens])

print("Top 10 vocabulary words:", vocab.most_common(10))
```

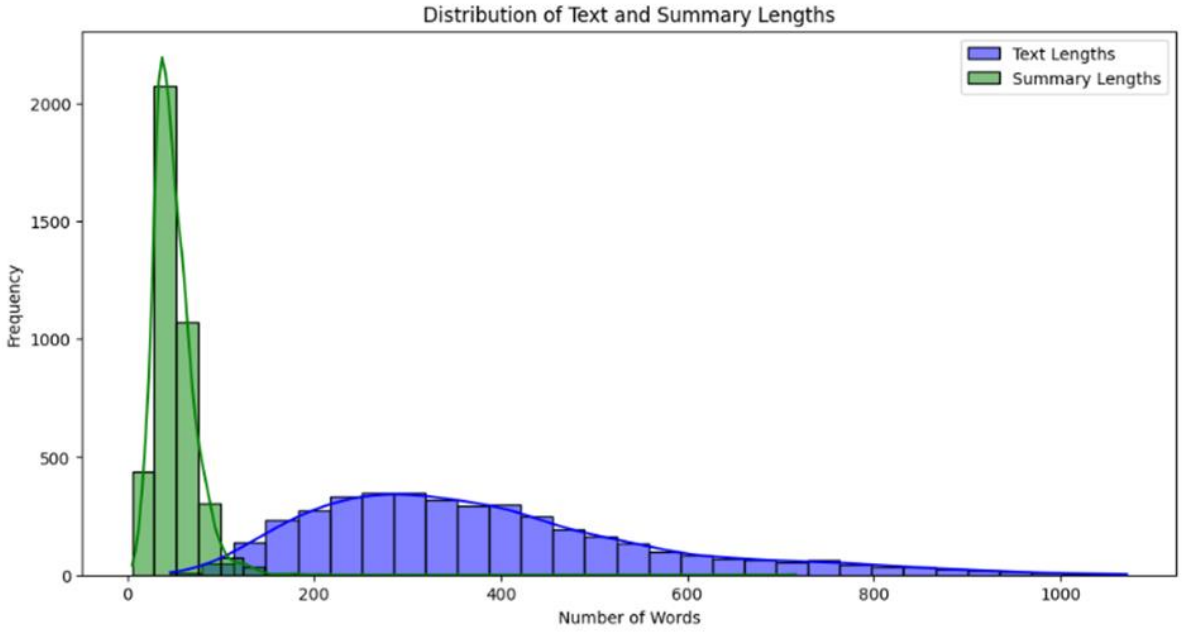
```
Top 10 vocabulary words: [('the', 8343), ('to', 5577), ('in', 5040), ('and', 4348), ('of', 4276), ('was', 2519), ('on', 223
7), ('for', 2187), ('he', 1695), ('is', 1694)]
```

(f) Data Visualization:

A dataset needs data visualization to help find patterns, trends, and anomalies in the data, which leads to improved comprehension and insights. In order to make sure the dataset is appropriate for training reliable text summarization models, it helps evaluate the caliber and distribution of articles and summaries. Furthermore, biases and gaps in the dataset can be found via visualizations, which can help direct preprocessing and augmentation efforts.

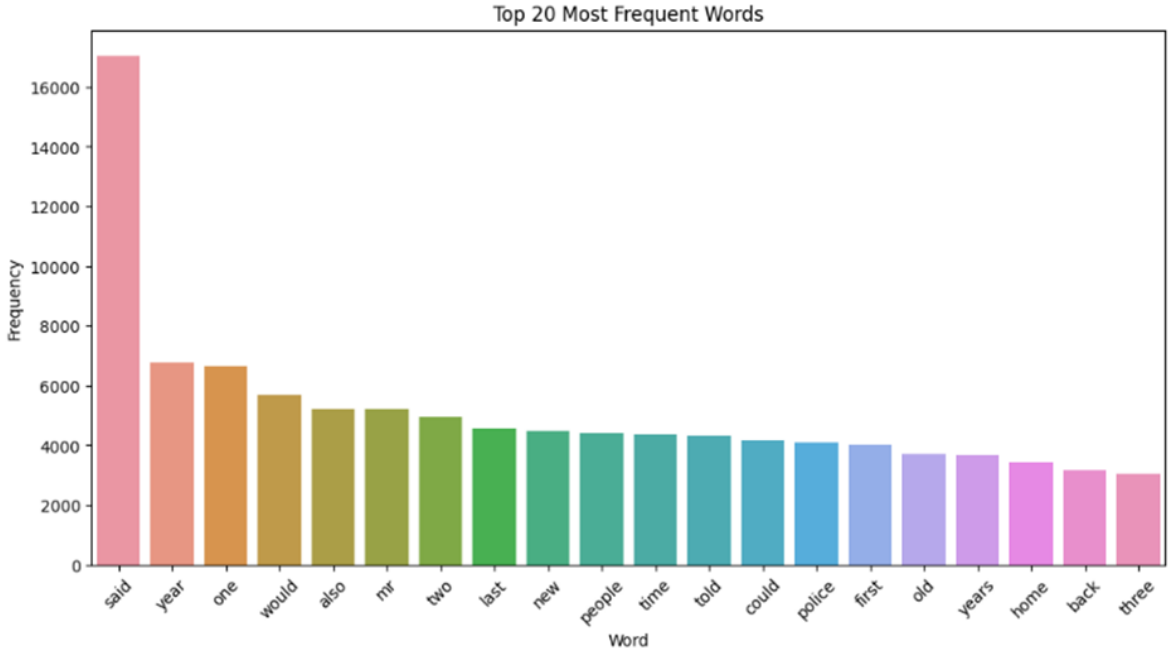
Visualization1:

The below histogram displays the frequency of the number of words in a text & summary distribution.

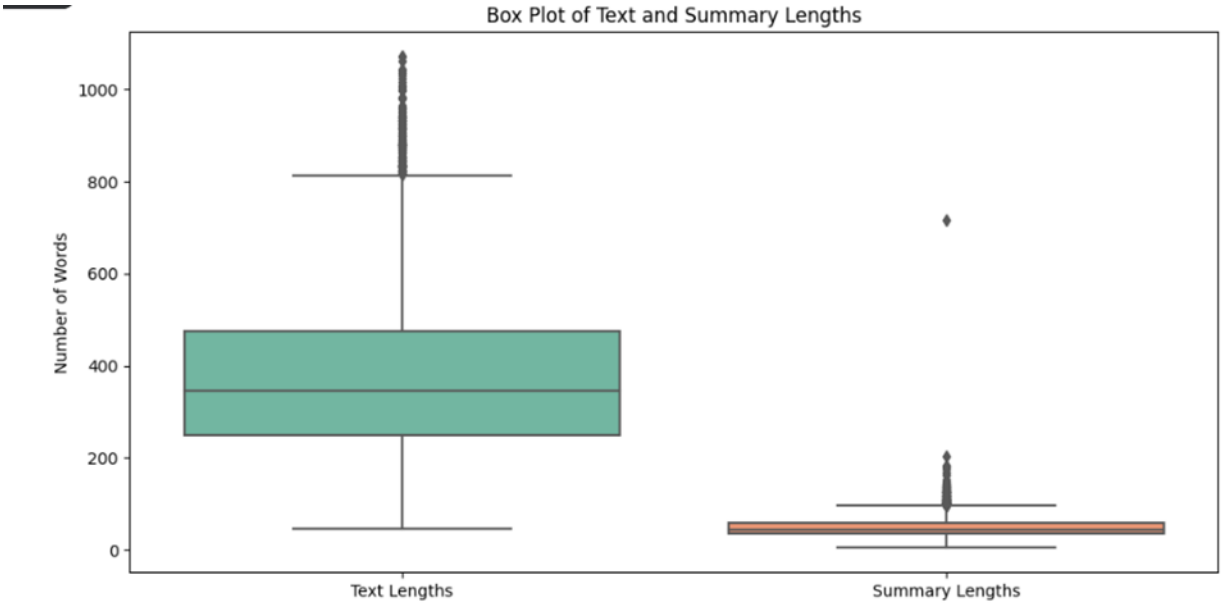


,

Visualization 2: the following histogram graph plots most common 20 words used in articles from train dataset



Visualization 3: This box plot indicates the number of words in cleaned articles and highlights



[g] Create datasets of train, validation and test dataset of cleaned articles and highlights

```
from transformers import PegasusTokenizer, PegasusForConditionalGeneration, BartTokenizer, BartForConditionalGe
from datasets import Dataset, DatasetDict

train_data = Dataset.from_dict({'article': cleaned_train_articles, 'highlights': cleaned_train_highlights})
val_data = Dataset.from_dict({'article': cleaned_val_articles, 'highlights': cleaned_val_highlights})
test_data = Dataset.from_dict({'article': cleaned_test_articles, 'highlights': cleaned_test_highlights})

dataset = DatasetDict({
    'train': train_data,
    'validation': val_data,
    'test': test_data
})
```

- train_data: creates a dataset object of training data using cleaned training articles and highlights.
- val_data: creates a dataset object for validation data using cleaned validation articles and highlights.
- test_data: creates a dataset object for test data using cleaned test articles and highlights.

6. Models

a. checkpoint 1

Analyzed the dataset and it's structure. Performed extracting files, reading story files, initial pre-processing and data cleaning, tokenization and build vocabulary, manage data loading and token indexing, DataLoader objects to batch and shuffle training and validation datasets, defined a GRU

model for encoder and decoder. Seq2Seq class was created that combines the encoder and decoder, handles the forward pass, and uses teacher forcing during training. Since, this was checkpoint1, the ideas were still not decided.

b. checkpoint 2

Previously in checkpoint 1 we attempted to build a single model for summarization. Now we are trying to run pre-trained models with different hyperparameters and compare the outcomes because using a single model is complex and does not work well for text summarization. We reviewed each highlight to confirm it contains at least four words. If a highlight has fewer than four words, we filtered it out and removed it from further processing. We obtained the top ten most commonly used terms from this vocabulary collection and also created a list of tokenized words for each highlight, which is the breakdown of the highlight into individual words. We then Tokenize and preprocess the datasets for model input

c. later improvements

Models implemented in our project

Model1: Bert2Bert

By using the BERT architecture for both the encoder and the decoder, the BERT2BERT model, which was refined using the CNN/DailyMail summarization dataset, produces an encoder-decoder model that is transformer-based. Weights from pretrained BERT models are used to initialize this model, which is then adjusted especially for summarization tasks. In order to enhance summarization performance, the CNN/DailyMail dataset is used to fine-tune the weights.

Model 2: Facebook/bart-base (Bidirectional Auto Regressive from Transformer)

Base model has 6 layers in the encoder and decoder. But bart large has 12 layers

With a bidirectional (BERT-like) encoder and an autoregressive (GPT-like) decoder, BART is a transformer encoder-decoder (seq2seq) model. First, text is corrupted using an arbitrary noising function, and then BART is pre-trained by learning a model to reconstruct the original text. BART performs well on comprehension tasks such as text classification and question answering, but it is especially useful when tuned for text generation tasks like summarization and translation.

Encoder: It reads the complete input sequence in both directions, just like BERT, so it can comprehend the context from both perspectives.

Decoder: It produces output text autoregressive, predicting the next word based on the preceding ones, just like GPT does.

Model 3: Google-t5 (Text-To-Text Transfer Transformer)

The google-t5 model is a flexible architecture based on a transformer that can handle a range of natural language processing (NLP) tasks by transforming them into a text-to-text format. T5 summarizes text by processing the input text and producing a brief summary as the output.

Encoder-Decoder Structure: T5 employs a transformer architecture, which is akin to conventional sequence-to-sequence models, in which the encoder processes the input text and the decoder produces the output text.

Pre-training: Using a masked language modeling objective, T5 is trained to predict text spans by substituting a mask token for each text segment and learning to do so from the surrounding context.

7. Fine tuning

Fine tuning is done by adjusting the parameters in the training arguments method. The following are the parameters used for our project:

→ output_dir: “ouput_path”

this is the directory location where the training output and model checkpoints are saved.

→ eval_strategy: “epoch”

this determines the frequency of our evaluation. It can have: “no”, “epoch”, “steps”, etc as the values. Here, we use evaluate per epoch.

→ learning_rate: used 5e -5

This value is used by optimizer to update the weights. We used 5e -5 learning rate value. Smaller the learning rate, better the model can be stabilized for every epoch.

→ per_device_train_batch_size: 2

Defines the batch size for training the model per device.

→ per_device_eval_batch_size: 2

Defines the batch size for evaluating the model per device.

→ weight_decay: 0.01

To avoid overfitting by penalizing large weights in model. Here, during training, applying a weight decay of 1% on model weights.

→ save_total_limit: 1

To save the number of checkpoints. 1 means the latest checkpoint is saved.

→ num_train_epochs: 5

Total number of times the model gets trained for the given dataset.

→logging_dir: “./logs”

saves the training logs as “logs” in current working directory

→logging_steps: 10

This parameter is used to save how often the logging should happen (in steps). Here, for every 10 steps, training metrics are logged.

→gradient_checkpointing: True

Saves memory by not storing few intermediate results. Set to “true” to save memory

→ fp16_opt_level: “O2”

Specifies the optimization level for floating-point 16 (FP16) precision.”O2” specifies an optimization level for memory efficiency and speed control related to NVIDIA Apex Library.

→ report_to:”wandb”

This parameters specifies the location of training metrics to be reported. “Wandb” is for storing weights and biases

8. Results

For facebook/bart-base model:

```
ROUGE Scores:
rouge-1:
  r: 0.4634
  p: 0.5938
  f: 0.5205
rouge-2:
  r: 0.2273
  p: 0.2857
  f: 0.2532
rouge-l:
  r: 0.2927
  p: 0.3750
  f: 0.3288
for facebook-bart model
BERT Precision: 0.8584
BERT Recall: 0.8532
BERT F1 Score: 0.8558
```

ROUGE Scores: The ROUGE-1 precision (0.5938) is higher than the recall (0.4634). ROUGE-2 scores are moderate, indicating some overlap for bigrams.

BERT Scores: High BERT precision (0.8584) and recall (0.8532), indicating the generated summaries are close in meaning to the reference texts.

For Google-T5 Model

```
ROUGE Scores:
rouge-1:
  r: 0.3659
  p: 0.6522
  f: 0.4687
rouge-2:
  r: 0.0909
  p: 0.1538
  f: 0.1143
rouge-l:
  r: 0.1951
  p: 0.3478
  f: 0.2500
for google-t5 model
BERT Precision: 0.8605
BERT Recall: 0.8468
BERT F1 Score: 0.8536
```

ROUGE Scores: The ROUGE-1 precision is high (0.6522), but the recall is a bit moderate (0.3659). ROUGE-2 scores are low, indicating less overlap for bigrams.

BERT Scores: High BERT precision (0.8605) and good recall (0.8468), results that the model generates summaries that are very close in meaning to the reference text.

For Bert2Bert model:

```
ROUGE Scores:
rouge-1:
  r: 1.0000
  p: 0.6721
  f: 0.8039
rouge-2:
  r: 1.0000
  p: 0.4681
  f: 0.6377
rouge-l:
  r: 1.0000
  p: 0.6721
  f: 0.8039
for bert2bert model
BERT Precision: 0.8077
BERT Recall: 0.9751
BERT F1 Score: 0.8836
```

ROUGE Scores: Perfect recall and high precision for ROUGE-1 and ROUGE-L, but lower precision for ROUGE-2.

BERT Scores: High precision (0.8077) and very high recall (0.9751), indicating excellent content overlap with reference summaries.

Evaluation Metrics

For topics like text summarization, we can evaluate the accuracy of generated text with reference text using ROUGE score, BERT score, etc.

1. ROUGE score

   Recall-Oriented Understudy for gist Evaluation (ROUGE). ROUGE-n compares n-grams of generated text with n-grams of reference text.

   What is N-Gram?

   It’s a chunk of n-words

   Ex: Deep Learning is awesome

      1-Gram: “Deep”, “Learning”, “is”, “awesome”.

2-Gram: “Deep Learning”, “Learning is”, “is awesome”

Let’s take an example on how to calculate the ROUGE score.

Generated text: I really loved reading the hunger games

Reference text: I loved reading the hunger games

Unigram (1-gram) for generated text: “ I “,”really” ,”loved”,” reading”,” the”,” hunger”,” games”

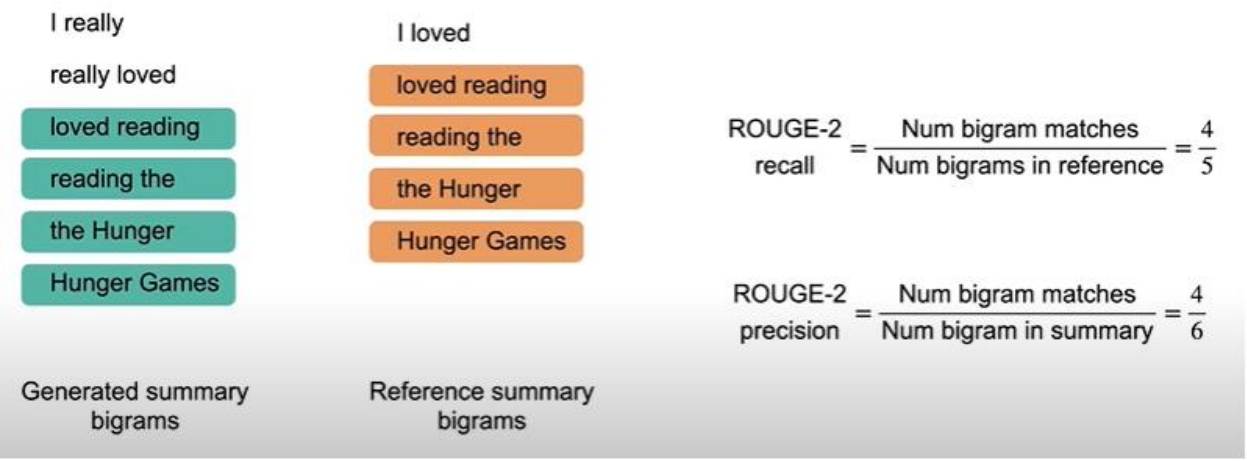
Unigram (1-gram) for reference text: “I”,”loved”,”reading”,”the”,”hunger”,”games”

$$\underset{\text{recall}}{\text{ROUGE-1}} = \frac{\text{Num word matches}}{\text{Num words in reference}} = \frac{6}{6}$$

$$\underset{\text{precision}}{\text{ROUGE-1}} = \frac{\text{Num word matches}}{\text{Num words in summary}} = \frac{6}{7}$$

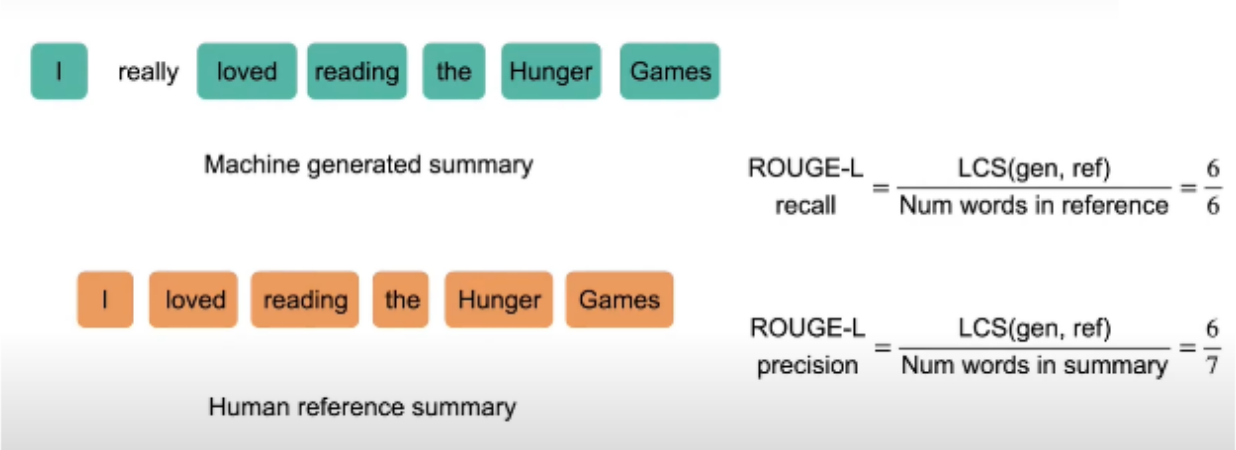
$$\underset{\text{F1-score}}{\text{ROUGE-1}} = 2\left(\frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}\right) = 0.92$$

Similarly, we calculate ROUGE-2



And for ROUGE-L:

“L” stands for longest subsequence.



The main advantage of ROUGE-L over ROUGE-1 and ROUGE-2 is that it doesn't depend on the consecutive matches of n-gram, which means it tends to capture the sentence structure more accurately.

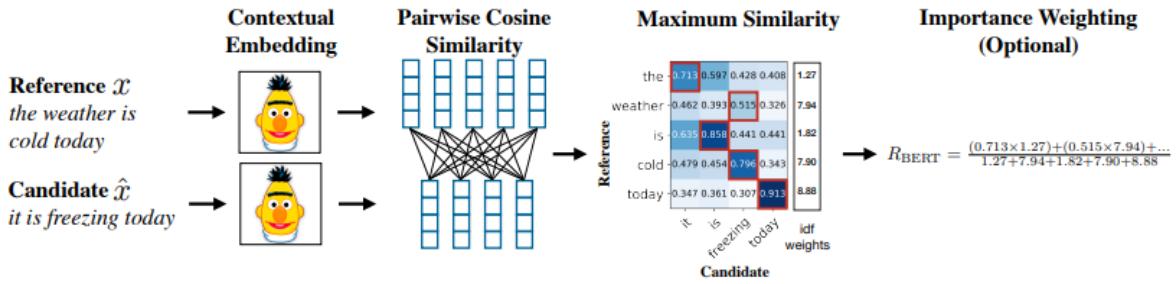
2. BERT score

Bi-directional Encoder Representation from Transformers is a context based model.

Step 1: Contextual Embedding: Models such as BERT, Roberta, XLNET, and XLM create contextual embedding based on surrounding words to represent reference and candidate phrases.

Step 2: Cosine Similarity: Cosine similarity is used to quantify how similar the contextual embedding of the reference and candidate phrases are to one other.

Step3: Token matching for precision and recall is done in step three. Recall and precision are computed by matching each token in the candidate sentence to the most comparable token in the reference sentence and vice versa. The results are pooled to determine the F1 score.



$$R_{\text{BERT}} = \frac{1}{|x|} \sum_{x_i \in x} \max_{\hat{x}_j \in \hat{x}} \mathbf{x}_i^\top \hat{\mathbf{x}}_j \quad , \quad P_{\text{BERT}} = \frac{1}{|\hat{x}|} \sum_{\hat{x}_j \in \hat{x}} \max_{x_i \in x} \mathbf{x}_i^\top \hat{\mathbf{x}}_j \quad , \quad F_{\text{BERT}} = 2\frac{P_{\text{BERT}} \cdot R_{\text{BERT}}}{P_{\text{BERT}} + R_{\text{BERT}}}$$

Step 4: Importance Weighting: Inverse Document Frequency (IDF), which can be optionally and domain-dependently added to BERTScore equations, is used to weigh the importance of rare words.

Example:

Reference Text: "the weather is cold today"

Candidate Text: "it is freezing today"

step1:Contextual Embedding

Every token in the two texts is transformed into high-dimensional vectors via BERT.

Step 2: In pairs Similarity of Cosines

Compute the reference and candidate text tokens' cosine similarity matrix.

Step 3: Maximum similarity

Determine the highest cosine similarity between any token in the candidate text and any token in the reference text for each token.

Let’s consider an example values (not exact calculated values)

Candidate \ Reference	the	weather	is	cold	today
it	0.713	0.369	0.428	0.404	0.347
is	0.360	0.369	0.515	0.441	0.316
freezing	0.428	0.515	0.342	0.433	0.347
today	0.360	0.316	0.428	0.404	0.878

Recall calculation: How well the reference text tokens are represented in the candidate text (generated text). For each token from reference text, find maximum similarity with all the tokens from the candidate text.

- the: max(0.713, 0.360, 0.428, 0.360) = 0.713
- weather: max(0.369, 0.369, 0.515, 0.316) = 0.515
- is: max(0.428, 0.515, 0.342, 0.428) = 0.515

- cold: max(0.404, 0.441, 0.433, 0.404) = 0.441
- today: max(0.347, 0.316, 0.347, 0.878) = 0.878

For simpler calculations, let's keep the weighted scores as optional.

Average of Maximum Similarities: (0.713+0.515+0.515+0.441+0.878)/5=0.6124

Precision calculation: How well the candidate text tokens are represented in the reference text

For each token from candidate text, find maximum similarity with all the tokens from reference text.

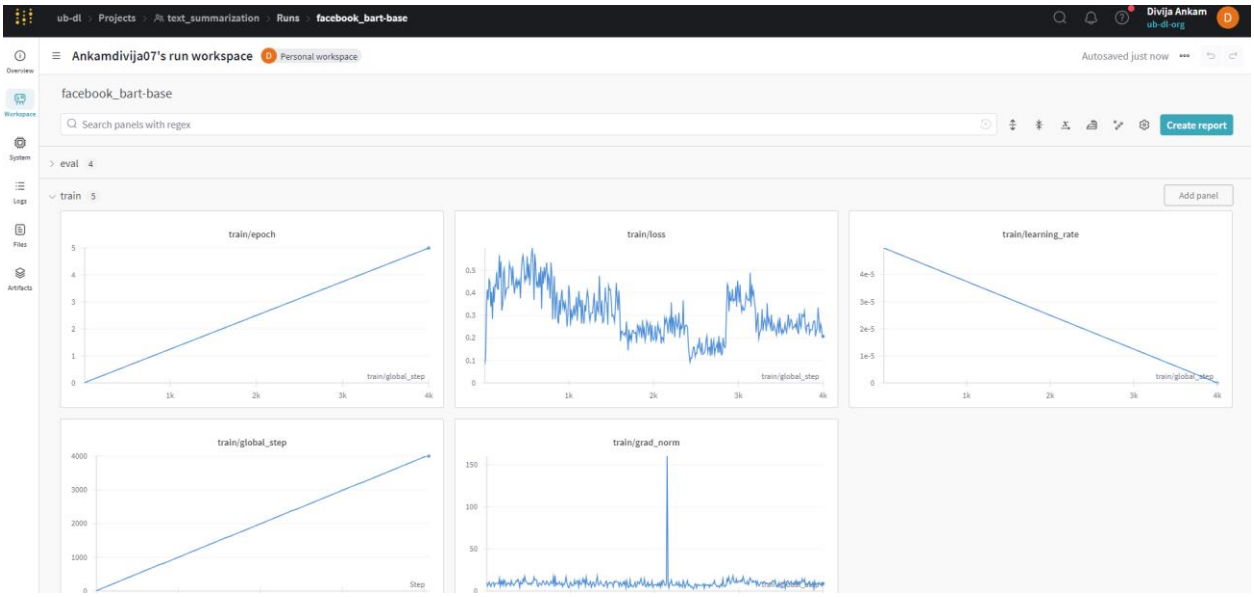
- it: max(0.713, 0.369, 0.428, 0.404, 0.347) = 0.713
- is: max(0.360, 0.369, 0.515, 0.441, 0.316) = 0.515
- freezing: max(0.428, 0.515, 0.342, 0.433, 0.347) = 0.515
- today: max(0.360, 0.316, 0.428, 0.404, 0.878) = 0.878

Average of Maximum Similarities: (0.713+0.515+0.515+0.878)/4=0.65525

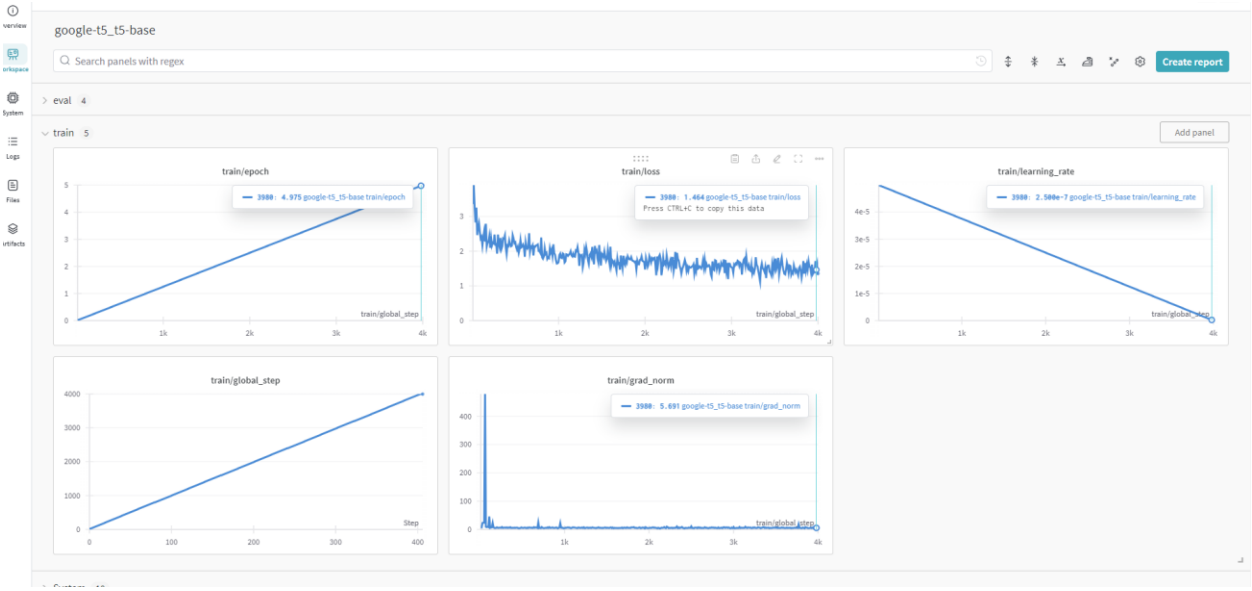
9. AI tools and other Observations

AI tool integration : https://wandb.ai/ub-dl/text_summarization?nw=nwuserankamdivija07

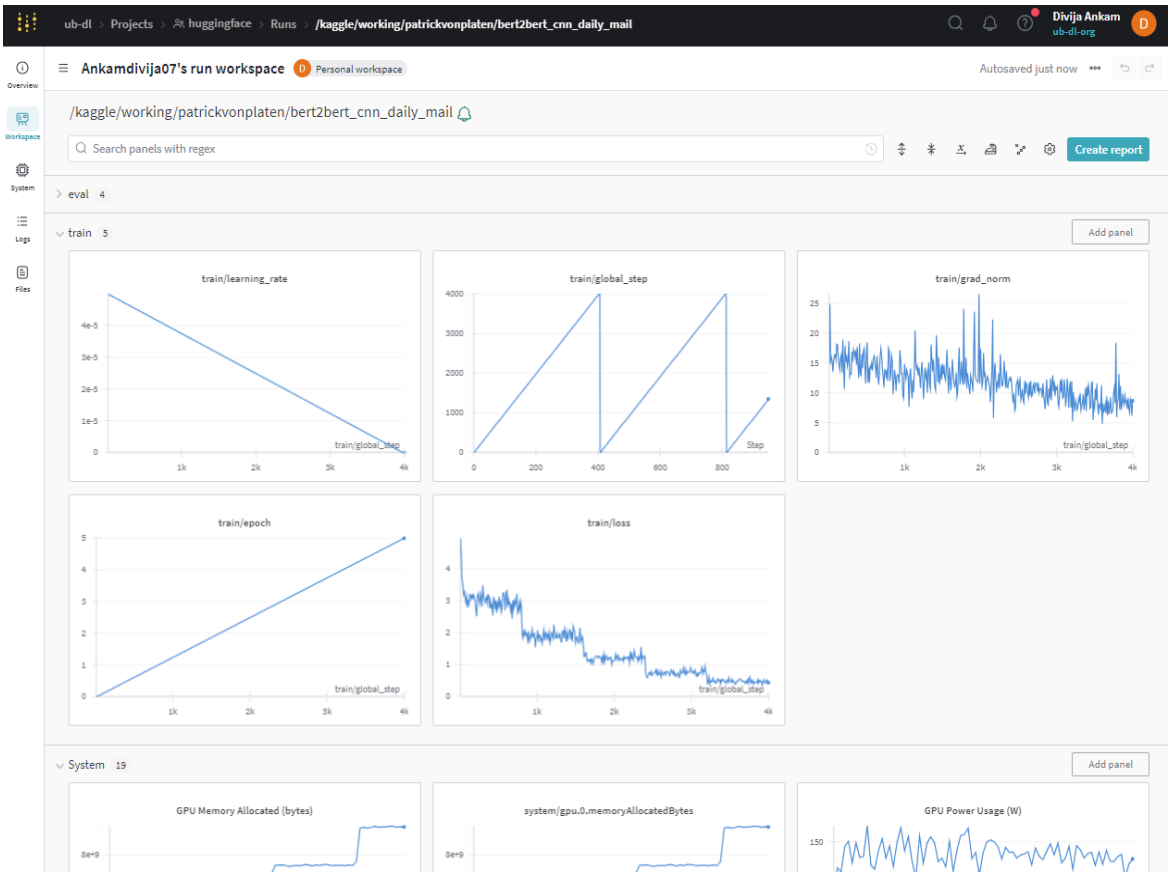
Facebook Bart Base model run:



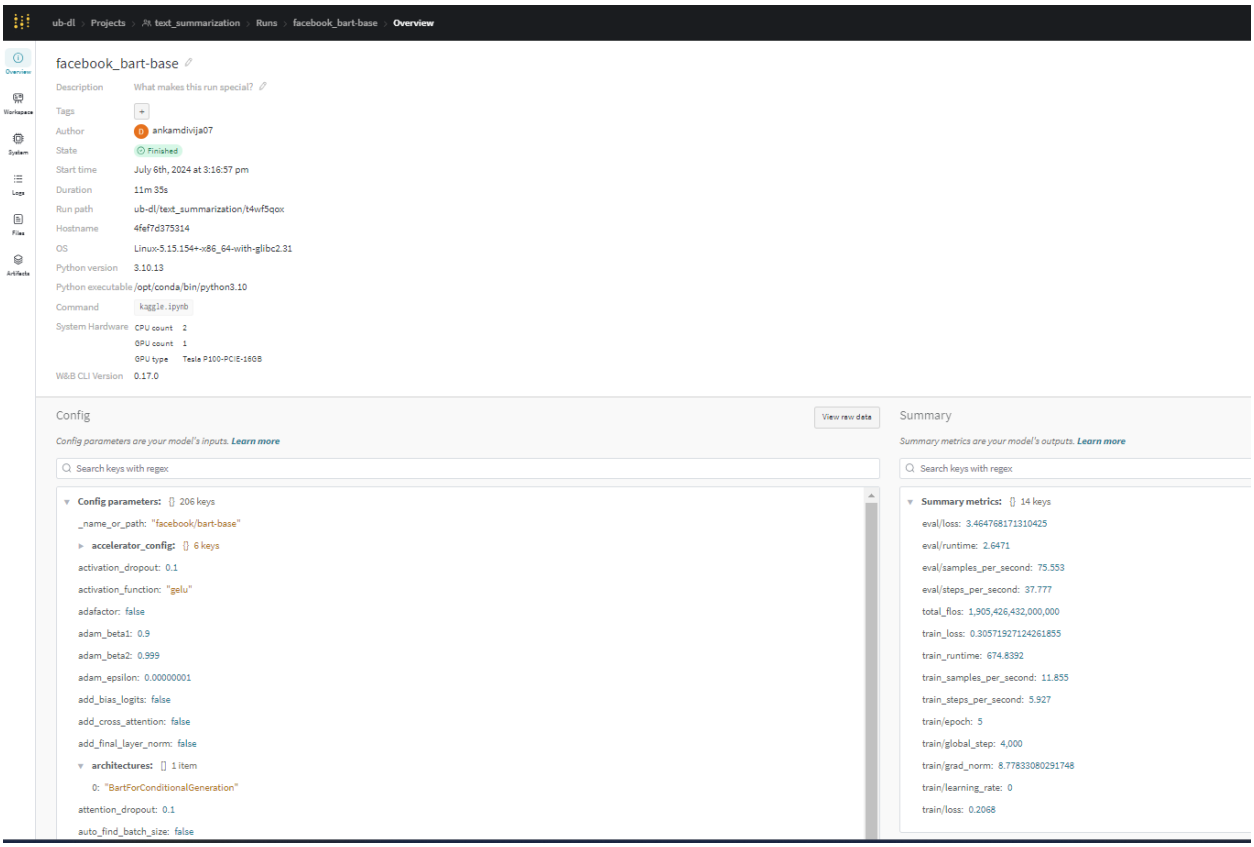
Google T5 model run :



Bert2Bert model run :

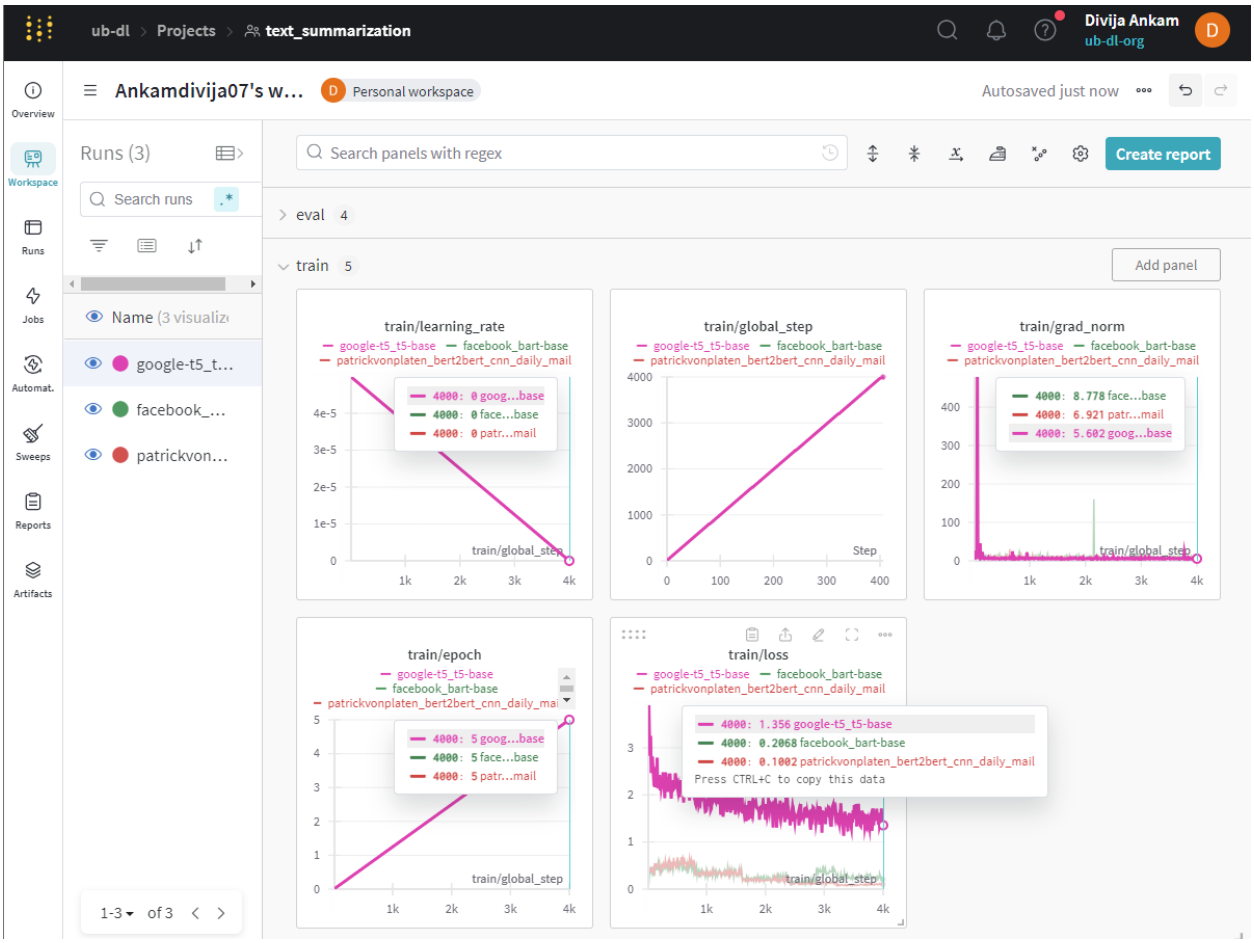


Models configuration on wandb platform :



Run history snippet on jupyter notebook :

Model comparison on wandb :



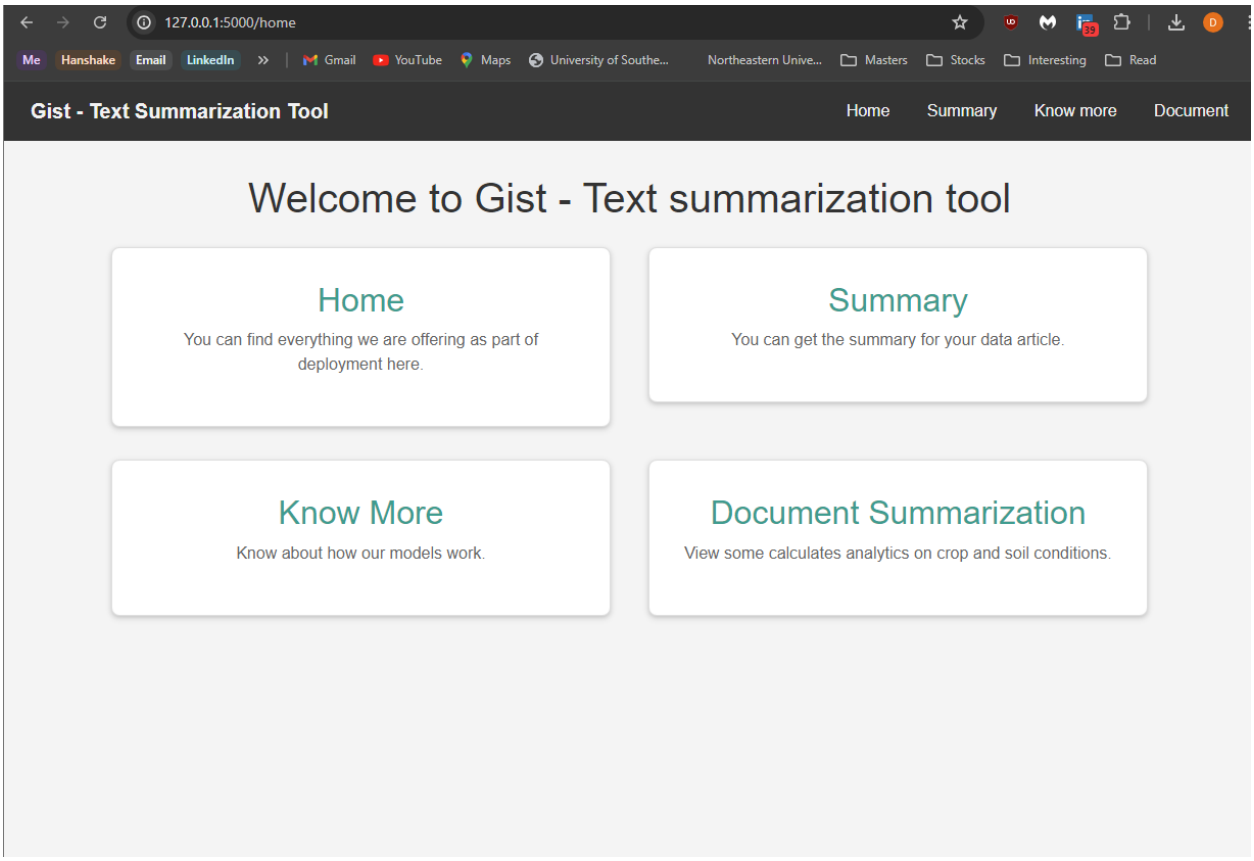
Observations:

1. Bert2Bert model and facebook bart model are trained well and quicker than compared to Google t5 model
2. Training loss are decreased over the epochs
3. Google t5 model is slowly trained but shows significant improvement over the epochs

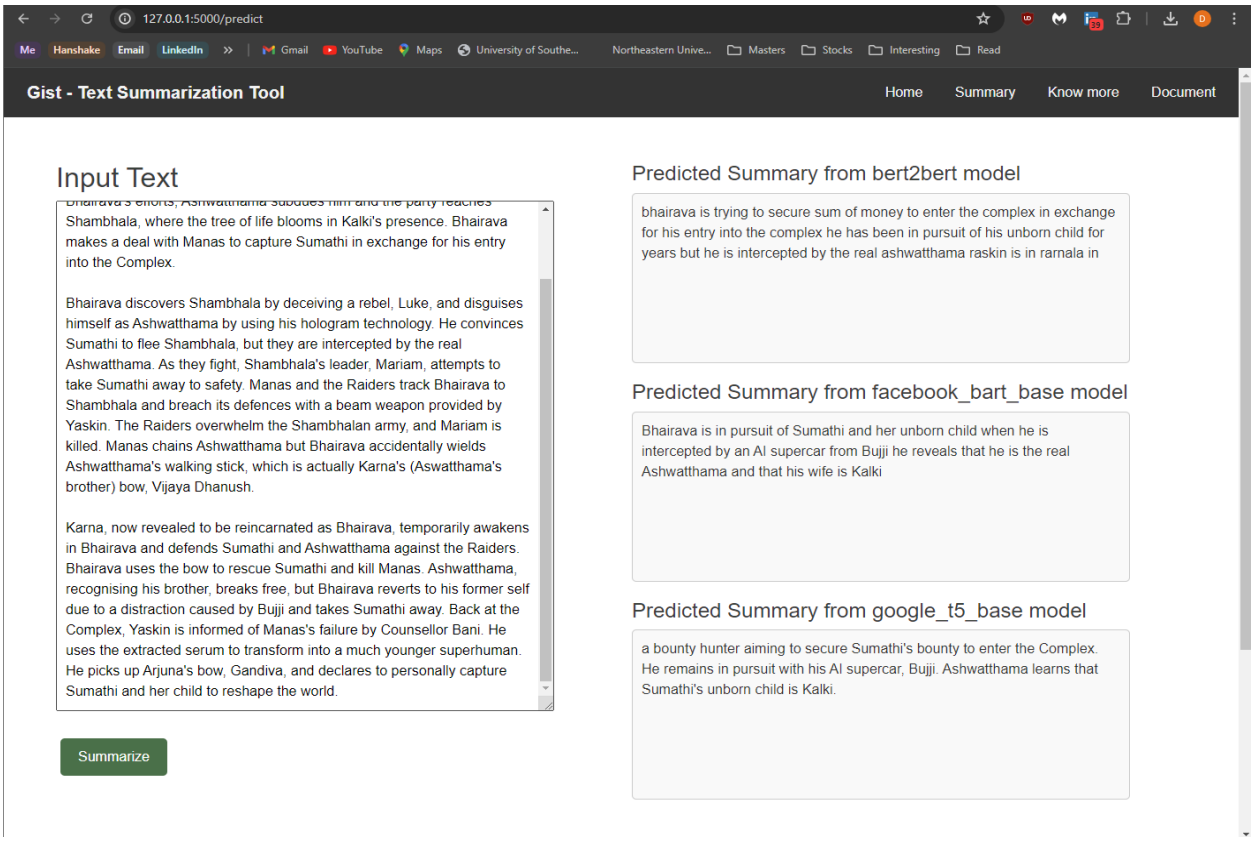
10. Deployment

We deployed our model with a user interface as below

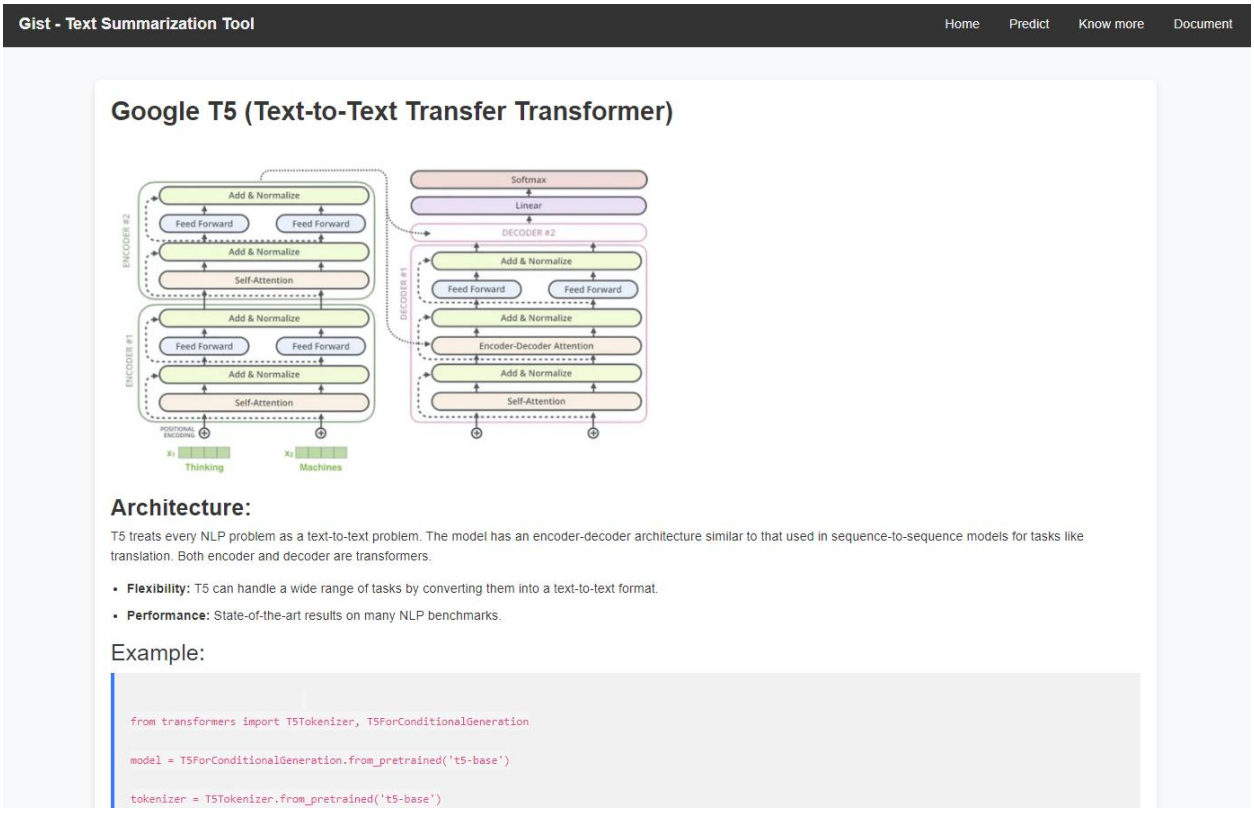
Home Screen:



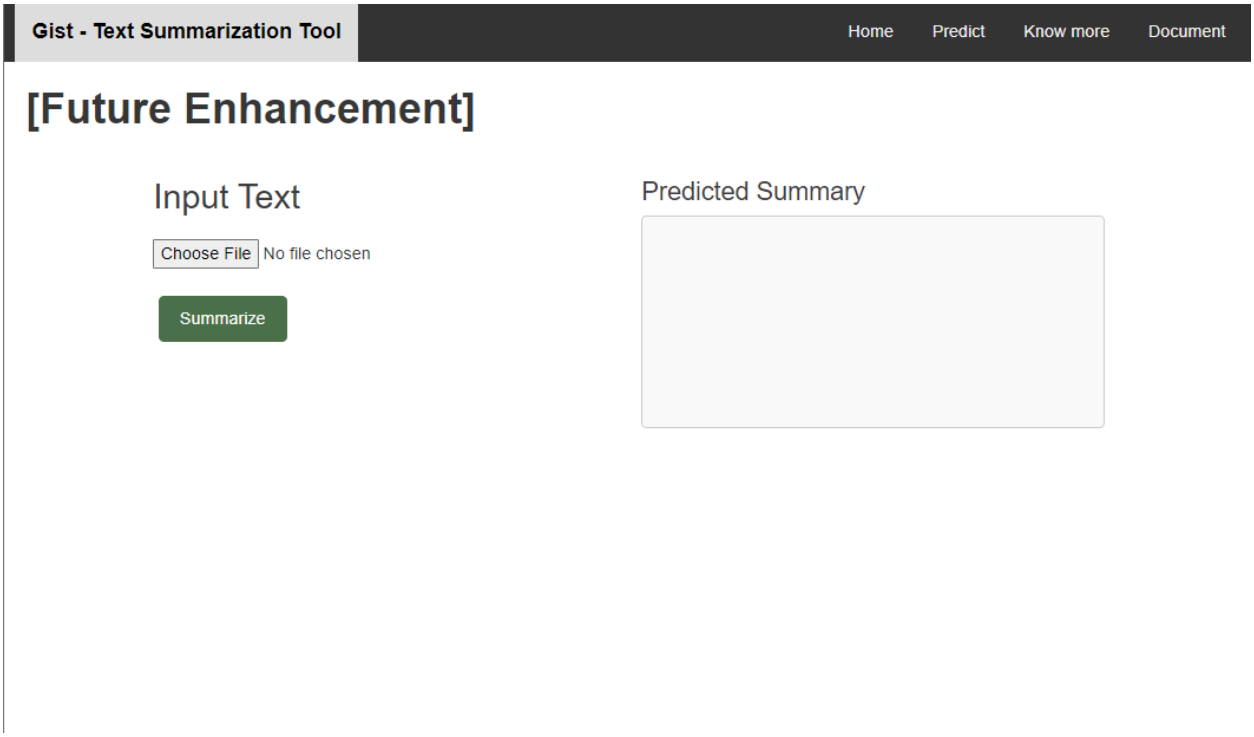
Summarization screen:



Model descriptions screen:



Future enhancements screen:



REFERENCES:

[1] ROUGE scores:https://www.youtube.com/watch?v=TMshhnrEXlg

[2] BERT scores: https://arxiv.org/pdf/1904.09675

[3] Deployment: https://medium.com/@chenyumei8866/deploying-pytorch-models-quickly-with-flask-5f8464e1c98

[4] T5 Model : https://medium.com/analytics-vidhya/t5-a-detailed-explanation-a0ac9bc53e51

[5] BART Model : https://medium.com/@nadirapovey/bart-model-architecture-8ac1cea0e877

[6] Metrics : https://fabianofalcao.medium.com/metrics-for-evaluating-summarization-of-texts-performed-by-transformers-how-to-evaluate-the-b3ce68a309c3

[7] NLP : https://huggingface.co/learn/nlp-course/en/chapter7/5

[8] NLP : https://huggingface.co/learn/nlp-course/en/chapter7/5#preparing-a-multilingual-corpus

[9] https://huggingface.co/learn/nlp-course/en/chapter7/5

[10]https://fabianofalcao.medium.com/metrics-for-evaluating-summarization-of-texts-performed-by-transformers-how-to-evaluate-the-b3ce68a309c3

[11] https://huggingface.co/learn/nlp-course/en/chapter7/5#preparing-a-multilingual-corpus

[12] Bert2Bert: https://huggingface.co/patrickvonplaten/bert2bert_cnn_daily_mail

[13] Facebook/bart-base: https://huggingface.co/facebook/bart-base

[14] Google/t5: https://huggingface.co/google-t5/t5-base