NEWS SUMMARIZER PROJECT REPORT

J004 Siddhant Sindhkar
J019 M. Saqib Siddiqui
J026 Sameer Mulani

INTRODUCTION

In today's digital age, access to news and information is more abundant than ever before. However, this wealth of data can sometimes become overwhelming, especially when trying to keep up with the latest developments and stories. Our project addresses this issue head-on, with the primary aim of simplifying the consumption of news articles. Our objective is clear: to automatically generate concise and coherent abstractive summaries of news articles, enabling readers to grasp the core insights without delving into lengthy narratives.

Leveraging the advanced capabilities of the state-of-the-art Pegasus transformer model and fine-tuning the model for our use case, we've developed a sophisticated system that not only extracts vital information but also crafts human-like summaries that encapsulate the essence of the original articles. This project represents a crucial step toward enhancing the efficiency of news consumption, making it easier for individuals to stay informed and make well-informed decisions in an increasingly data-driven and fast-paced world.

OBJECTIVE

The aim of this project is to fine-tune a Pegasus transformer model for the task of abstractive summarization of news articles. The primary objective is to create a tool that simplifies news consumption by generating concise and coherent summaries, allowing users to quickly grasp the core insights from news articles. This project seeks to streamline the information retrieval process, enhance comprehension, and empower individuals to make well-informed decisions in an information-rich environment.

INPUT DATA

To facilitate fine-tuning of our model to generate short summaries of news articles, we have used news data from Inshort webapp. Inshorts is a news app that selects the latest and best news from multiple national and international sources and summarises them to present in a short format.

The data was retrieved by web scraping using the BeautifulSoup library. The dataset consists of two columns, news headlines which gives a very short 60 word summary of the news and second column is news article which contains the description of the news.Total of **55,000 rows** were obtained.

MODEL BUILDING

We have fine-tuned on a pre-trained Pegasus transformer model for the task of abstractive summarization. Pegasus pre-training task is intentionally similar to summarization, it works as important sentences are removed/masked from an input document and are generated together as one output sequence from the remaining sentences, similar to an extractive summary. Pegasus achieves SOTA performance on 12 downstream summarization tasks spanning news, science, stories, instructions, emails, patents, and legislative bills.

Experiments demonstrate it achieves state-of-the-art performance on all 12 downstream datasets measured by ROUGE scores and is also trained on 568M parameters. For downstream summarisation, public abstractive summarization datasets were used, one of them was the CNN/DailyMail dataset which consisted of 93k articles from the CNN, and 220k articles from the Daily Mail newspapers. Both publishers supplemented their articles with bullet point summaries.

So we used the CNN/DailyMail checkpoint in our code i.e the saved verison when it was pre-trained on to capture the model's parameters, architecture, and knowledge acquired during training.

.

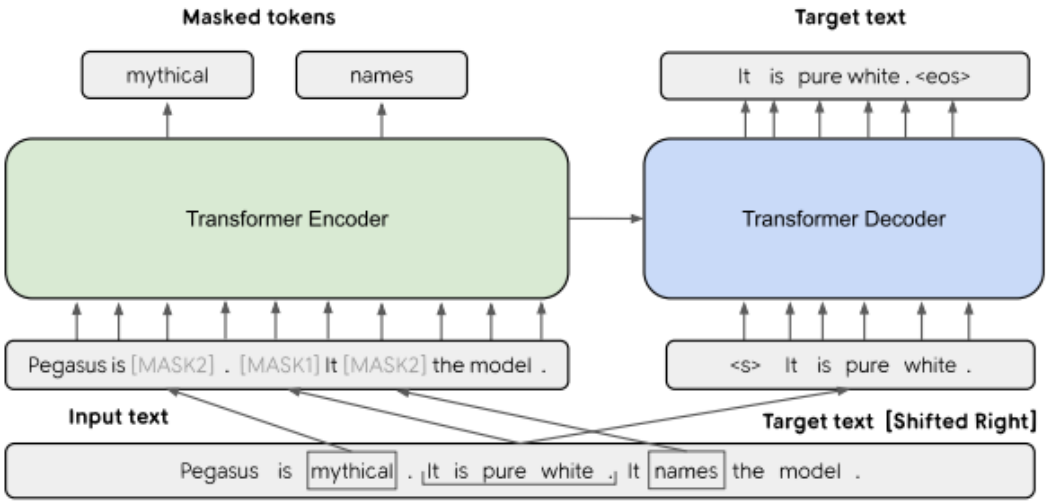


Figure: : The base architecture of PEGASUS is a standard Transformer encoder-decoder. Both GSG and MLM are applied simultaneously to this example as pre-training objectives. Originally there were three sentences. One sentence is masked with [MASK1] and used as target generation text (GSG). The other two sentences remain in the input, but some tokens are randomly masked by [MASK2] (MLM)

Pegasus is a Sequence-to-sequence model with the same encoder-decoder model architecture as BART. It is pre-trained jointly on two self-supervised objective functions: Masked Language Modeling (MLM) and a novel summarization specific pretraining objective, called Gap Sentence Generation (GSG).

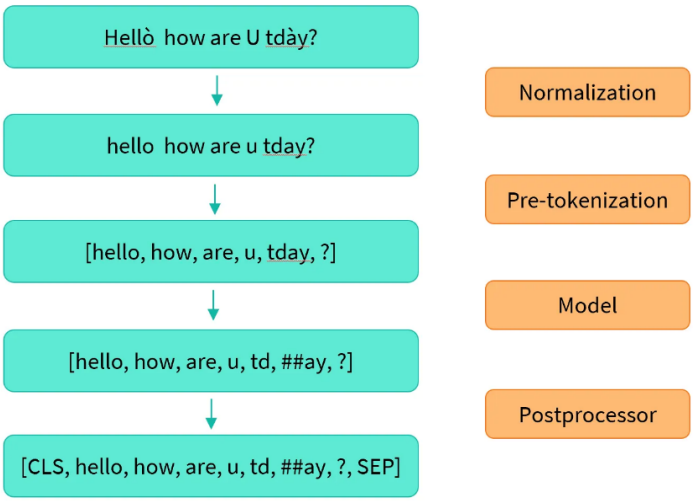
MLM: encoder input tokens are randomly replaced by a mask tokens and The goal is to predict the original tokens that were masked based on the context provided by the unmasked tokens.

GSG: In GSC, sentences are randomly removed from a text document, creating gaps in the content. Pegasus is then tasked with generating the missing sentences that would naturally fit within the document. This task encourages the model to produce sentences that are contextually relevant and maintain the document's flow.

AutoTokenizer

Once we have obtained our data on which fine-tuning is to be done, we apply the AutoTokenizer, which is an inbuilt tokenizer from Huggingface Hub. The reason to use AutoTokenizer is that it reads the text and automatically selects the appropriate tokenizer from the list of tokenizers like T5Tokenizer, BertTokenizer, GPT2Tokenizer etc.

An implementation of a tokenizer consists of the following pipeline of processes, each applying different transformations to the textual information:



The tokenization pipeline.

Normalization: The normalization step involves some general cleanup, such as removing needless whitespace, lowercasing, and/or removing accents

Pre-tokenization: A tokenizer cannot be trained on raw text alone. Instead, we first need to split the texts into small entities, like words. That's where the pre-tokenization step comes in. A word-based tokenizer can simply split a raw text into words on whitespace and punctuation. Those words will be the boundaries of the subtokens the tokenizer can learn during its training.

Modelling: After normalization and pre-processing steps, we apply a training algorithm to the text data. The output of this step is dependent on the type of training strategy we are going to use. This is where the AutoTokenizer selects which tokenizer to use based on the input text (like T5Tokenizer etc.)

Postprocessor: Similar to the modeling part, a number of post-processors are available depending on the training strategy used. They're responsible for adding the special tokens to the input sequence as needed by the model.

After Tokenizing we convert our data to 'dataset' datatype for processing of the data and directly splitting into training and testing.We then dynamically pad the input tokens and labels using **DataCollatorForSeq2Seq.**

TrainingArguments

After preprocessing our data, we fine-tune the hyperparameters of our model. This is achieved by using the TrainingArguments class from the transformers library. The

TrainingArguments class is used to define and customize the training arguments for fine-tuning or training a transformer model. It allows us to specify various parameters such as the output directory, number of training epochs, batch size, learning rate, evaluation strategy, weight decay, and more. Thus, we can fine-tune our transformer model according to our specific requirements and use case.

Trainer

The Trainer class from the transformers library is used to streamline the training process for transformer models. We initialize the Trainer function, the input to which are the training arguments defined by the TrainingArguments class, tokenizer and padding function along with our training and testing data on which we want to fine-tune.

Then we run the trainer function and let the hyperparameters get tuned according to the input data and also calculate the rogue score on the test data set.

Once our model is fine-tuned, we create a repository on the Huggingface Hub website and upload our model to it. This enables us to use our model whenever and wherever required.

```
    (decoder): PegasusDecoder(
      (embed_tokens): Embedding(96103, 1024, padding_idx=0)
      (embed_positions): PegasusSinusoidalPositionalEmbedding(1024, 1024)
      (layers): ModuleList(
        (0-15): 16 x PegasusDecoderLayer(
          (self_attn): PegasusAttention(
            (k_proj): Linear(in_features=1024, out_features=1024, bias=True)
            (v_proj): Linear(in_features=1024, out_features=1024, bias=True)
            (q_proj): Linear(in_features=1024, out_features=1024, bias=True)
            (out_proj): Linear(in_features=1024, out_features=1024, bias=True)
          )
          (activation_fn): ReLU()
          (self_attn_layer_norm): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
          (encoder_attn): PegasusAttention(
            (k_proj): Linear(in_features=1024, out_features=1024, bias=True)
            (v_proj): Linear(in_features=1024, out_features=1024, bias=True)
            (q_proj): Linear(in_features=1024, out_features=1024, bias=True)
            (out_proj): Linear(in_features=1024, out_features=1024, bias=True)
          )
          (encoder_attn_layer_norm): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
          (fc1): Linear(in_features=1024, out_features=4096, bias=True)
          (fc2): Linear(in_features=4096, out_features=1024, bias=True)
          (final_layer_norm): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
        )
      )
      (layer_norm): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
    )
  )
  (lm_head): Linear(in_features=1024, out_features=96103, bias=False)
)
```

```
    (decoder): PegasusDecoder(
      (embed_tokens): Embedding(96103, 1024, padding_idx=0)
      (embed_positions): PegasusSinusoidalPositionalEmbedding(1024, 1024)
      (layers): ModuleList(
        (0-15): 16 x PegasusDecoderLayer(
          (self_attn): PegasusAttention(
            (k_proj): Linear(in_features=1024, out_features=1024, bias=True)
            (v_proj): Linear(in_features=1024, out_features=1024, bias=True)
            (q_proj): Linear(in_features=1024, out_features=1024, bias=True)
            (out_proj): Linear(in_features=1024, out_features=1024, bias=True)
          )
          (activation_fn): ReLU()
          (self_attn_layer_norm): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
          (encoder_attn): PegasusAttention(
            (k_proj): Linear(in_features=1024, out_features=1024, bias=True)
            (v_proj): Linear(in_features=1024, out_features=1024, bias=True)
            (q_proj): Linear(in_features=1024, out_features=1024, bias=True)
            (out_proj): Linear(in_features=1024, out_features=1024, bias=True)
          )
          (encoder_attn_layer_norm): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
          (fc1): Linear(in_features=1024, out_features=4096, bias=True)
          (fc2): Linear(in_features=4096, out_features=1024, bias=True)
          (final_layer_norm): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
        )
      )
      (layer_norm): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
    )
  )
  (lm_head): Linear(in_features=1024, out_features=96103, bias=False)
)
```

TESTING

ROUGE score measures the similarity between the machine-generated summary and the reference summaries using overlapping n-grams, word sequences that appear in both the machine-generated summary and the reference summaries. The most common n-grams used are unigrams, bigrams, and trigrams. It calculates the recall of n-grams in the machine-generated summary by comparing them to the reference summaries.

The formula for ROUGE score is as follows:

ROUGE = ∑ (Recall of n-grams)

Where:

Recall of n-grams is the number of n-grams that appear in both the machine-generated summary and the reference summaries divided by the total number of n-grams in the reference summaries.

ROUGE score ranges from 0 to 1, with higher values indicating better summary quality. Like BLEU score, a perfect summary would have a ROUGE score of 1, while a completely incorrect summary would have a ROUGE score of 0.

ROUGE scores are branched into ROUGE-N,ROUGE-L, and ROUGE-S.

ROUGE-N: ROUGE-N measures the overlap of n-grams (contiguous sequences of n words) between the candidate text and the reference text. It computes the precision, recall, and F1-score based on the n-gram overlap. For example, ROUGE-1 (unigram) measures the overlap of single words, ROUGE-2 (bigram) measures the overlap of two-word sequences, and so on. ROUGE-N is often used to evaluate the grammatical correctness and fluency of generated text.

ROUGE-L: ROUGE-L measures the longest common subsequence (LCS) between the candidate text and the reference text. It computes the precision, recall, and F1-score based on the length of the LCS. ROUGE-L is often used to evaluate the semantic similarity and content coverage of generated text, as it considers the common subsequence regardless of word order.

ROUGE score is widely used in text summarization tasks as it provides a way to objectively assess the quality of machine-generated summaries compared to reference summaries. It takes into account the overlap of n-grams, which helps in capturing the important content of the summary. ROUGE score is also flexible as it allows the use of different n-gram lengths based on the task requirements.

To compare the different transformer models and select the best one for our use case, we used Rogue Score as an evaluation metric.
Pegasus achieved the highest rouge score of all models, and was hence used for deployment, after necessary fine tuning.

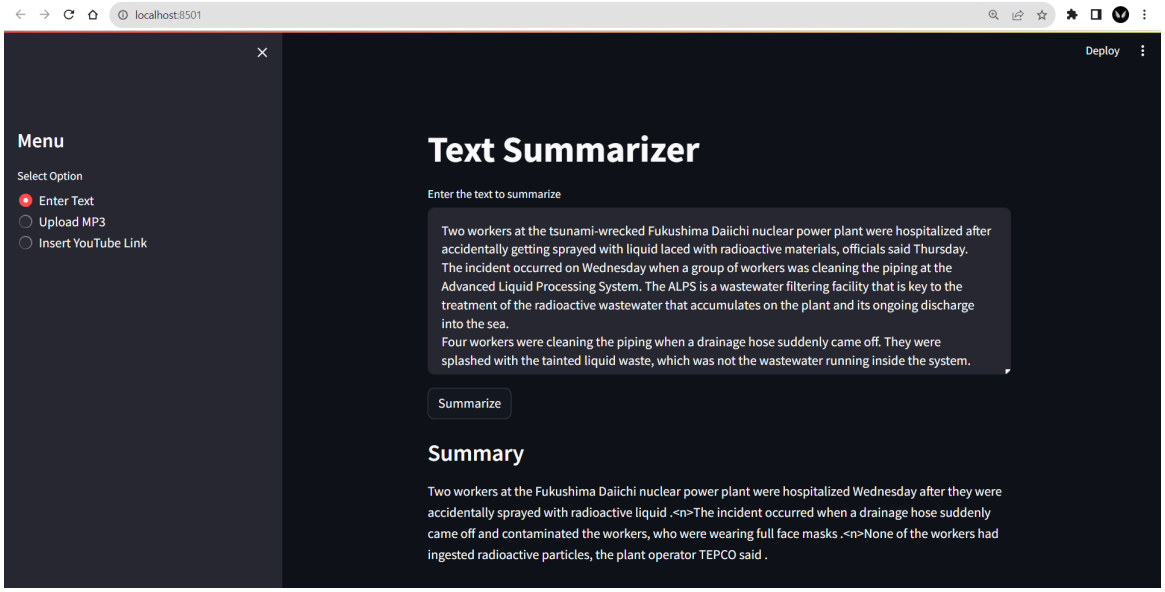
	Pegasus	T5	BART
rouge1	0.562601	0.427988	0.435645
rouge2	0.356816	0.225650	0.215857
rougeL	0.439377	0.313929	0.316608
rougeLsum	0.441730	0.313567	0.318620

DEPLOYMENT

The model has been deployed on streamlit. Three options are available for users to input data for generating summaries.The interface provides flexibility by allowing users to enter text, upload an MP3 file, or input a YouTube video link. Each option serves a unique purpose, enabling users to generate concise summaries based on their preferred input method.

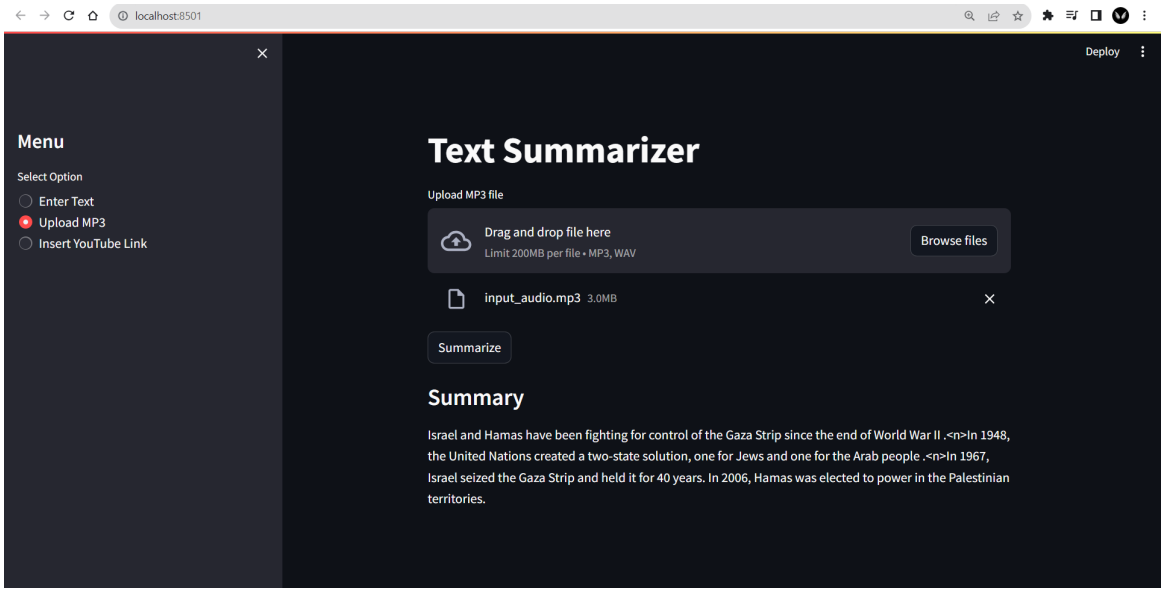
Option 1: **Enter Text**

The first option allows users to directly enter text into the system. Users can type or paste their desired content into the provided text input area. This option is ideal for users who want to summarize any news article they have come across, or want to summarize their own written text.



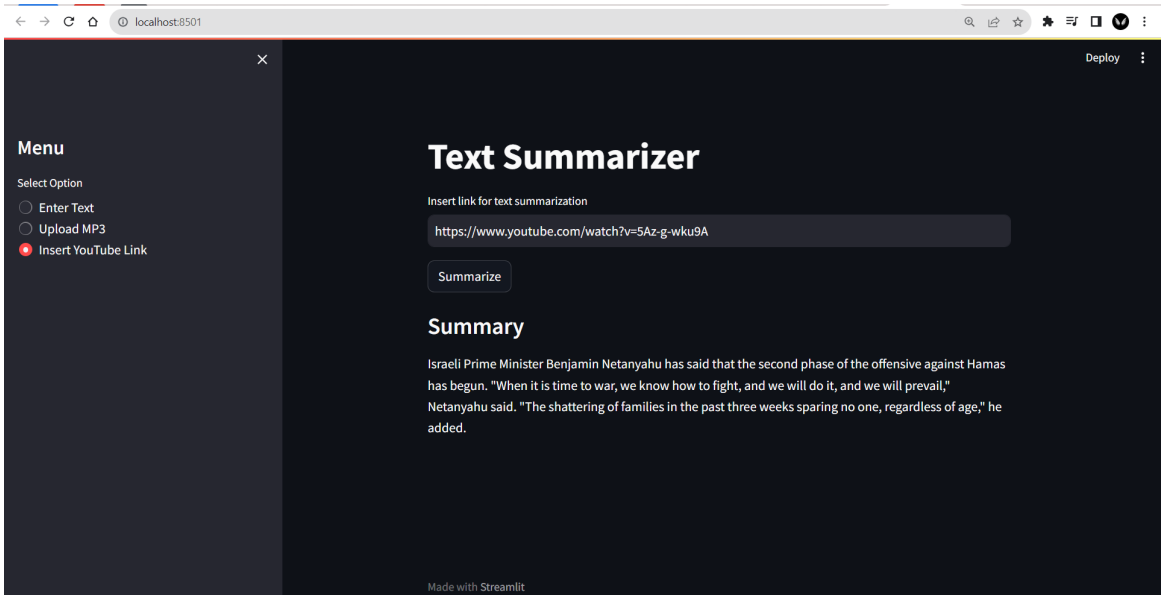
Option 2: **Upload MP3 File**

The second option enables users to upload an MP3 file containing audio content. The system will process the audio file and generate a summary based on the audio content. This option caters to users who have audio recordings or interviews that they want to summarize quickly and efficiently.



Option 3: **Enter YouTube Link**

The third option allows users to input a YouTube video link. The system will extract the audio content from the specified video and generate a summary based on the audio. This option is particularly useful for users who want to summarize any news content available on YouTube, which will save their time to watch the entire video.



CONCLUSION

The goal of our fine-tuning was to adapt the Pegasus transformer model to generate high-quality summaries specifically tailored for our news article use case. We achieved this by training the model on 55k data points web scraped from Inshot web app, allowing it to learn the requirements of generating a concise summary.

During the fine-tuning process, we adjusted and tuned various hyperparameters of the Pegasus model to optimize its performance for report summarization. To evaluate the performance of the fine-tuned model, we utilized standard evaluation metric such as ROUGE scores. These scores measure the quality of the generated summaries compared to reference summaries, providing valuable insights into the model's effectiveness in summarizing report content.

The results of our fine-tuning process revealed that the fine-tuned Pegasus model demonstrated impressive performance in generating informative summaries for news articles. The model was able to capture the essential details of the original news article while maintaining conciseness and coherence.

FUTURE SCOPE

Segmentation into Different Categories

Implement a sophisticated categorization system that can segment the entire day's news into distinct categories such as politics, sports, entertainment, technology, and more. This will enable users to access summarized content tailored to their specific interests and preferences, enhancing the overall user experience and engagement with the news.

Personalization and User Preferences

Incorporate personalized recommendation features that leverage user preferences and behavior to deliver curated news summaries tailored to individual interests. Implement a user profiling system that analyzes user interactions and feedback to fine-tune the summarization process, providing users with highly relevant and engaging news content aligned with their specific preferences and browsing history.

Multi-Modal Summary Display

Enhance the summarization tool to support multi-modal summary display, allowing users to access summarized content both as text and as voice notes. Integrate advanced text-to-speech (TTS) technologies to generate high-quality voice notes, ensuring accessibility for visually impaired individuals. This inclusive approach will enable a wider

audience to benefit from the summarized news content, promoting greater accessibility and user engagement.

Real-Time News Fetching and Summarization

Integrate robust API-based functionalities that enable real-time news fetching from various reputable sources on the internet. Develop a dynamic system that can aggregate and process live news updates efficiently, generating real-time summaries that reflect the latest developments and events. Implement an effective caching mechanism to optimize data retrieval and minimize latency, ensuring the timely delivery of up-to-date and relevant news summaries to users.

Collaboration with News Aggregation Platforms

Establish strategic partnerships and collaborations with prominent news aggregation platforms and publishers to expand the reach and impact of the summarization tool. Integrate the tool with existing news apps and platforms, offering users a seamless and integrated news consumption experience. Foster collaborations to enhance the overall quality and diversity of news sources accessible through the summarization tool, promoting comprehensive and well-rounded news coverage for users worldwide.

REFERENCES

https://medium.com/@sthanikamsanthosh1994/understanding-bleu-and-rouge-score-for-nlp-evaluation-1ab334ecadcb

https://rohanpaul.gumroad.com/l/python-core-with-under-the-hood-explanations

https://huggingface.co/docs/transformers/tasks/summarization

https://huggingface.co/facebook/bart-large-cnn

https://huggingface.co/google/pegasus-cnn_dailymail

https://huggingface.co/t5-base

https://streamlit.io/

https://www.aljazeera.com/news/2023/10/28/israeli-troops-in-gaza-amid-blackout-hamas-to-counter-with-full-force

https://medium.com/@awaldeep/hugging-face-understanding-tokenizers-1b7e4afdb154

https://github.com/kishanpython/Inshorts-News-Data-Collection/blob/main/README.md
https://arxiv.org/pdf/1912.08777v2.pdf