NEWS SUMMARIZATION USING NLP

TEAM - 3
Abhinay Kotla, Hema Sri Puppala
Department of Computer Science and Engineering
The University of Texas at Arlington
{axk5827, hxp9993}@mavs.uta.edu

Abstract:

In the digital age, the exponential growth of information has led to a pressing need for automatic text summarization (ATS) systems that can condense large amounts of text into concise, meaningful summaries. This project explores two approaches to ATS—an LSTM-based encoder-decoder model and a fine-tuned T5 Transformer. Using the Indian News Dataset, these models were trained and evaluated to generate abstractive summaries. The evaluation process involved quantitative metrics like ROUGE scores and cosine similarity, as well as a novel semantic rating generated by GPT. Through these analyses, the T5 model emerged as the superior choice, providing human-readable summaries with greater semantic coherence compared to the LSTM model. This report details the entire workflow, including preprocessing, model design, training, evaluation, and analysis of results.

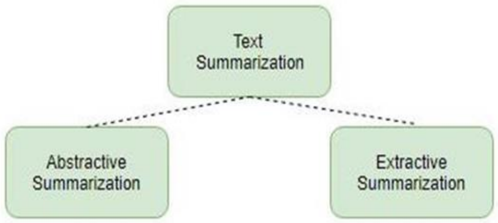
Introduction:

With the explosion of digital media, individuals and organizations struggle to process the vast amount of available information. Automatic text summarization has become an essential tool to address this challenge by extracting or generating condensed summaries from larger texts. Abstractive summarization, the focus of this study, generates new phrases and sentences that encapsulate the essence of the source material, making it more challenging yet valuable compared to extractive summarization.

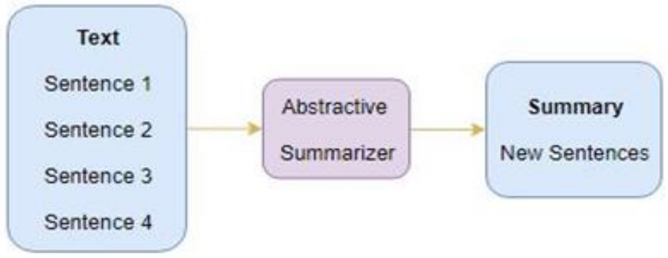
This project aims to evaluate the effectiveness of two abstractive summarization techniques:

1. A sequence-to-sequence LSTM-based encoder-decoder architecture, leveraging GloVe embeddings for textual representation.
2. The T5 (Text-to-Text Transfer Transformer), a state-of-the-art transformer-based model known for its versatility in various NLP tasks.

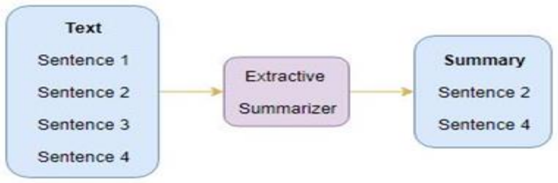
A concise and well-organized synopsis of text can be extracted using Automatic Text Summarization (ATS) from a wide range of sources, including books, emails, research papers, blogs, tweets, and news articles.
Two approaches to summarizing text:



Abstractive Summarization:
Abstractive Summarization produces more meaningful human written sentences, rather than being limited to terms from the source text.



Extractive Summarization:
Extractive summarization is a traditional way to generate summaries, such as clipping relevant chunks of the source text and combining them to make a rational summary.



Data Set Description:

We used the News Summary Indian News Dataset for this project [10]. The datasets were designated "news summary" and "news_summary_more," and they had 4515 and 98280 instances, respectively. For this project, we have only used the "news summary" data collection due to computational constraints. Here, we have just used text (the article's summary) and ctext (the full text), out of the six variables (author, date, headline, read more, text, and ctext).

Implementation:

Data Pre-processing:

Preprocessing or cleaning data is just as crucial to any machine learning project as building the model itself. One of the most unstructured types of data is text, and handling human language is not feasible. Neural Language Processing (NLP) is a background method that processes large amounts of text before providing an answer.

The various text pre-processing steps are:

1.Handling null values: Removing all the rows containing null values for news articles.

2.Lowercase Transformation: Lowering the case of a word (NLP -> nlp). This helps in standardizing the text data for further processing.

3.Removing HTML tags: Cleaning HTML tags before vectorizing the text data. This function takes a raw text string as input and returns the text without HTML tags.

4.Stop words and punctuation removal: Stop words (an, the, a, and so on) are frequently employed in papers. These terminologies have no real meaning because they do not assist in distinguishing between two publications.

5.Tokenization: Tokenization is essentially splitting a sentence, paragraph, phrase, or an entire text into smaller parts, such as individual terms or words.

6.Lemmatization: Unlike stemming, lemmatization restricts words to words that already exist in the language.

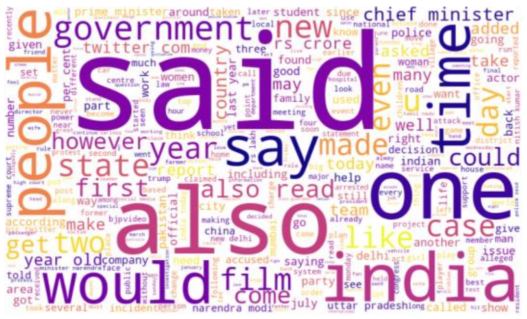
For modeling seq2seq data, special tokens are attached at the beginning and end of the summary as an extra pre-processing step.

Exploratory Data Analysis:

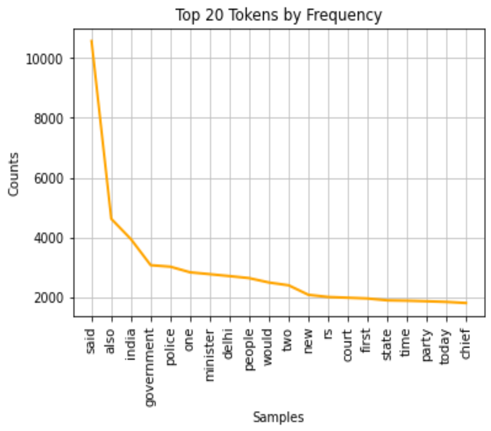
Exploratory data analysis, or EDA for short, is the process of identifying patterns, spotting irregularities, testing hypotheses, and verifying assumptions using graphical representations and summary statistics. We have carried out the following data analysis as part of EDA.

1) Type token ratio (TTR): To compute TTR, one must divide the total number of words in a text by the total number of tokens, or distinct word categories. While a low TTR indicates the opposite, a high TTR reflects a significant degree of lexical diversity. We received a 0.04 TTR score in our instance! as one may anticipate from news items.

2) Word cloud: Word clouds, sometimes referred to as tag-clouds, are graphic representations of word frequency that draw attention to words that recur regularly in a source text. The term appears more times in the text document (s) the longer it is in the image.



3) Top 20 frequent words: The terms with the highest frequency of occurrences throughout the entire corpus are called frequently occurring words. This type of analysis aids in our comprehension of the data's context. With 10570 times, the word "said" emerged as the most often occurring word. Additional frequently occurring terms in our corpus were "one," "would," "people," "also," and "new." examined the hapaxes as well. Among the outcomes are "notables," "patronize," and others.



Architecture:

Encoder-Decoder:

Sequence to Sequence (seq2seq) models are a type of Recurrent Neural Network architecture that is commonly used to handle complicated language problems such as question answering, constructing Chabot's, machine translation,text summarization, and so on.

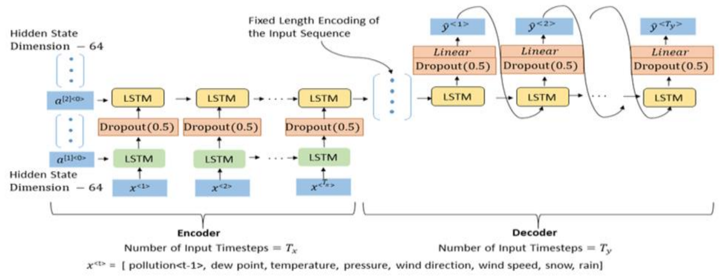
A Seq2Seq model has two primary components:

Encoder: The encoder is an LSTM model designed to process the entire input sequence, one word at a time, at each timestep. It captures contextual information from the sequence, enabling it to understand the input data's underlying patterns. The encoder's final hidden state and cell state serve as the initial states for the decoder.

Decoder: Similarly, the decoder is an LSTM network that generates the target sequence step by step. It learns to predict each subsequent word in the sequence based on the preceding ones. Before inputting the sequence into the decoder, special tokens <start> and <end> are added to mark the beginning and end of the sequence, respectively. During the testing phase, the actual target sequence is not available. Therefore, the decoding process begins by feeding the decoder the <start> token. It continues generating words until it predicts the <end> token, indicating the sequence's completion.

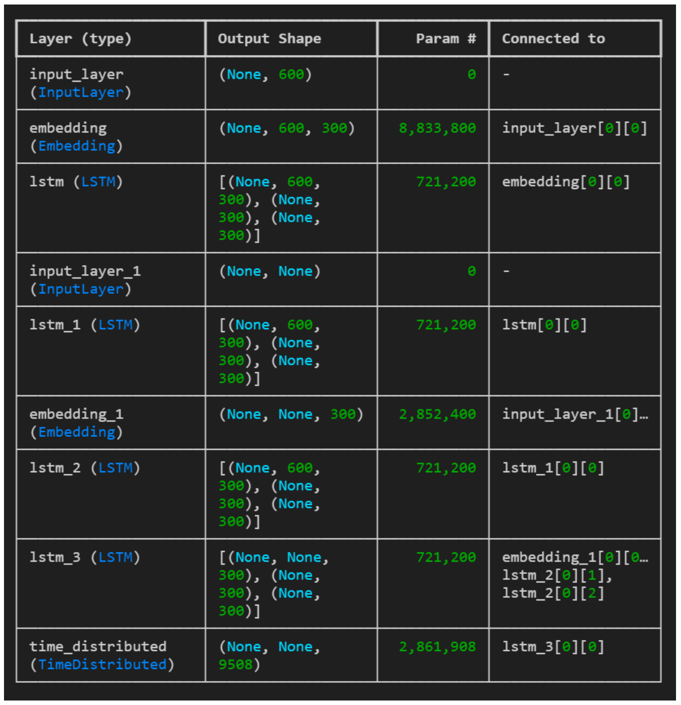
Model Building and Execution:

The model we built for the news summarization involved similar architecture of encoder and decoder as shown in the below image. Embedding dimension of 300 was used along with 3 layers of LSTM in the encoder and 1 layer of LSTM in decoder with a dropout 0.4. In the dense layer, **SoftMax** was used as an activation function.


Hidden State Dimension – 64
Fixed Length Encoding of the Input Sequence
$\hat{y}^{<1>}$
$\hat{y}^{<2>}$
$\hat{y}^{<T_y>}$
Linear
Dropout(0.5)
LSTM
$a^{[2]<0>}$
$a^{[1]<0>}$
Hidden State Dimension – 64
$x^{<1>}$
$x^{<2>}$
$x^{<T_x>}$
Encoder
Number of Input Timesteps = T_x
Decoder
Number of Input Timesteps = T_y
$x^{<t>}$ = [pollution<t-1>, dew point, temperature, pressure, wind direction, wind speed, snow, rain]

Model Summary:

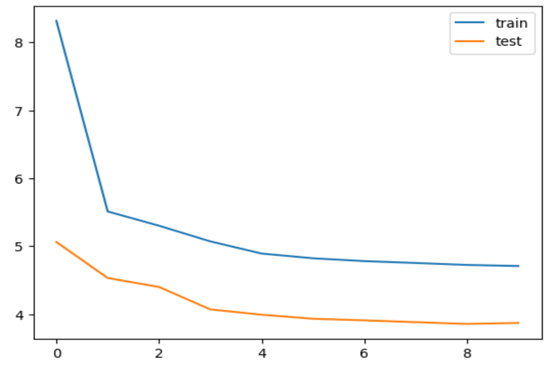
The model summary of the baseline model is as shown below. Encoder of the model contained 3 layers of LSTM and 1 layer of LSTM in the decoder part along with the input and embedding layers.



```
┌─────────────────────┬───────────────────┬────────────┬───────────────────┐
│ Layer (type)        │ Output Shape      │    Param # │ Connected to      │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ input_layer         │ (None, 600)       │          0 │ -                 │
│ (InputLayer)        │                   │            │                   │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ embedding           │ (None, 600, 300)  │  8,833,800 │ input_layer[0][0] │
│ (Embedding)         │                   │            │                   │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ lstm (LSTM)         │ [(None, 600,      │    721,200 │ embedding[0][0]   │
│                     │ 300), (None,      │            │                   │
│                     │ 300), (None,      │            │                   │
│                     │ 300)]             │            │                   │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ input_layer_1       │ (None, None)      │          0 │ -                 │
│ (InputLayer)        │                   │            │                   │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ lstm_1 (LSTM)       │ [(None, 600,      │    721,200 │ lstm[0][0]        │
│                     │ 300), (None,      │            │                   │
│                     │ 300), (None,      │            │                   │
│                     │ 300)]             │            │                   │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ embedding_1         │ (None, None, 300) │  2,852,400 │ input_layer_1[0]… │
│ (Embedding)         │                   │            │                   │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ lstm_2 (LSTM)       │ [(None, 600,      │    721,200 │ lstm_1[0][0]      │
│                     │ 300), (None,      │            │                   │
│                     │ 300), (None,      │            │                   │
│                     │ 300)]             │            │                   │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ lstm_3 (LSTM)       │ [(None, None,     │    721,200 │ embedding_1[0][0… │
│                     │ 300), (None,      │            │ lstm_2[0][1],     │
│                     │ 300), (None,      │            │ lstm_2[0][2]      │
│                     │ 300)]             │            │                   │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ time_distributed    │ (None, None,      │  2,861,908 │ lstm_3[0][0]      │
│ (TimeDistributed)   │ 9508)             │            │                   │
└─────────────────────┴───────────────────┴────────────┴───────────────────┘
```

Model Evaluation:

Our initial model results without hyper parameter tuning. The train loss and validation loss are plotted below. The validation loss is slightly higher than the train loss with an overall accuracy of 46% which is below the average model accuracy.



Limitations of Encoder-Decoder Architecture:

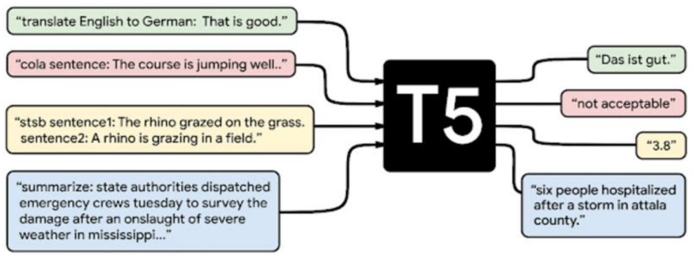
This encoder-decoder design is quite useful, although it does have some drawbacks.

1. The encoder processes the entire input sequence and compresses it into a fixed-size vector representation, which is then used to predict the output sequence. However, since the decoder relies solely on this fixed-size vector to generate predictions, it becomes less effective for longer sequences, as the single vector may not adequately capture all the detailed information needed for accurate predictions.

2. The concern with long sequences is long sequences are difficult for the encoder to remember into a fixed length vector. For LSTM to perform better, according to BLEU score report, the length of input text should be less than 30 and the length of reference should be around 15.

T5: Text-To-Text Transfer Transformer:

The T5 (Text-to-Text Transfer Transformer) model is pre-trained on the Colossal Clean Crawled Corpus (C4) dataset, a large and diverse text corpus. It delivers state-of-the-art performance across various natural language processing (NLP) tasks while maintaining versatility. This flexibility allows T5 to be fine-tuned for specific tasks, making it a powerful tool for solving diverse NLP challenges.


"translate English to German: That is good."
"cola sentence: The course is jumping well.."
"stsb sentence1: The rhino grazed on the grass. sentence2: A rhino is grazing in a field."
"summarize: state authorities dispatched emergency crews tuesday to survey the damage after an onslaught of severe weather in mississippi…"
T5
"Das ist gut."
"not acceptable"
"3.8"
"six people hospitalized after a storm in attala county."

Each task feeds text into a model that has been trained to create some target text. This enables the use of the same model, loss function, and hyperparameters for a variety of tasks as shown above - machine translation (highlighted in green), linguistic acceptability (highlighted in red), phrase similarity (highlighted in yellow), summarization (highlighted in blue).

Fine Tuning T5 transformer for News Summarization:

The HuggingFace Transformers hub has a T5 pre-trained model. For this project, we utilized the HuggingFace T5-base model. T5Tokenizer and T5-base model are included in the HuggingFace NLP library.

Before the main article content, a new string "summarize:" is prefixed to the values in the main article column. This adjustment is made to align with the formatting used in the T5 summary dataset. The first five rows of the resulting dataframe are then displayed in the console.

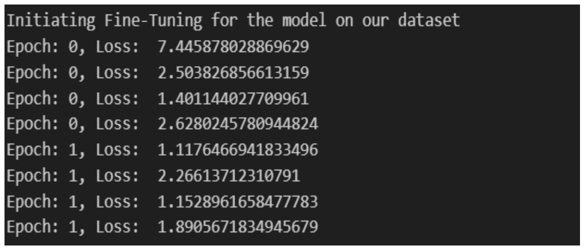
For test and validation, the updated data frame is partitioned in an 80:20 ratio.

1.To define our model, we utilized the "T5ForConditionalGeneration.from pretrained("t5-base")" command.

2. To define the tokenizer, we used the T5 Tokenizer from the pretrained("t5-base") command.
3. The Adam optimizer is being used.
4. We limited the input target to have 512 tokens and 150 for the output summary.

T5 Base Model Evaluation:

With a batch size of two, we trained the model for two epochs. Every 500th step, the training loss is displayed on the console.


```
Initiating Fine-Tuning for the model on our dataset
Epoch: 0, Loss:  7.445878028869629
Epoch: 0, Loss:  2.503826856613159
Epoch: 0, Loss:  1.401144027709961
Epoch: 0, Loss:  2.6280245780944824
Epoch: 1, Loss:  1.1176466941833496
Epoch: 1, Loss:  2.26613712310791
Epoch: 1, Loss:  1.1528961658477783
Epoch: 1, Loss:  1.8905671834945679
```

Training loss was quite high in the first iteration. However, it significantly reduced in the later iterations.

Summary Generated from the T5 Transformer:

Validation Process After Training:

1. **No Weight Updates:** During the validation phase, the model's weights remain fixed. The fine-tuned model is employed to generate new summaries based on the input article text.

2. **Progress Feedback:** The process prints progress updates to the console after every 100th step, indicating how many validation steps have been completed.

3. **Summary Conversion**: The original and generated summaries are both converted into lists and passed back to the main function.

4. **DataFrame Creation:** These lists are then used to construct a DataFrame with two columns: "Generated Summary" and "Actual Summary."

5. **Saving Results:** The resulting DataFrame is saved locally as a CSV file for future reference.

6. **Qualitative Analysis:** The saved DataFrame can later be used for qualitative analysis, allowing a detailed comparison of the model's output with the expected summaries.

Differences in Approach/Method and Accuracy/Performance

1. Difference in Approach/Method Between Our Project and the References

Approach Variations:

1. Alexander et al. [4] and Rush et al. [5] utilized traditional encoder-decoder frameworks, focusing on feed-forward models to map input words into contextual probability distributions. Their methodologies often incorporated beam search to optimize sequence generation by maximizing word probabilities.

2. Hujia et al. [6] enhanced encoder-decoder models by integrating attention mechanisms within LSTM and RNN architectures, enabling the model to concentrate on the most relevant parts of the input during the generation process.

3. Shengli et al. [8] combined convolutional neural networks (CNNs) with LSTM encoders to create abstractive summaries, emphasizing the generation of semantic phrases. In contrast, Touseef et al. [7] explored the use of gated mechanisms in neural networks to improve text summarization.

Key Differences:

1. Unlike references that predominantly used RNNs, our project employed **GloVe embeddings** alongside an LSTM encoder-decoder model, enhancing the representational capacity of the text input.

2. Our use of the **T5 Transformer** marks a significant departure from the RNN/LSTM-heavy focus in prior studies. The T5 architecture treats all NLP tasks as a text-to-text problem, allowing fine-tuning for abstractive summarization with pre-trained transformer models, a modern advancement not explored in the cited works.

3. While references like Abhijeet et al. [9] focused on extractive summarization using LSTM, our work exclusively addresses **abstractive summarization**, generating new sentences and phrases beyond direct text extraction.

2. Difference in Accuracy/Performance Between Our Project and the References

ROUGE Scores:

1.Hujia et al. [6] reported ROUGE scores of 20.52 (ROUGE-1) and 18.77 (ROUGE-2) for their attention-based models with a 3–4 layer RNN setup.

2. Abhijeet et al. [9] achieved ROUGE-1 and ROUGE-2 scores of 0.825 and 0.815, respectively, with an LSTM-based extractive summarization approach.

3. In contrast, our T5 model recorded **ROUGE-1 (0.5320)**, **ROUGE-2 (0.3507)**, and **ROUGE-L (0.4266)** scores, demonstrating competitive performance for abstractive summarization while offering more semantically coherent summaries.

Semantic Performance:

Our project introduced **cosine similarity metrics** and **GPT-based semantic evaluations**, which are novel compared to the traditional focus on ROUGE scores in the cited works. The T5 model achieved high cosine similarity peaks (~0.8), indicating strong semantic alignment with reference summaries.

Modern Architectures:

References like Shengli et al. [8] explored hybrid models (LSTM-CNN), but their performance metrics were not as thoroughly detailed. Our T5 implementation leveraged modern transformer architectures, which outperformed LSTM models in generating coherent and human-readable abstractive summaries, a limitation noted in many RNN/LSTM-based studies.

In summary, while prior works explored innovative applications of LSTM and RNN architectures, our integration of state-of-the-art transformer models (T5) offers a significant performance edge in generating abstractive summaries, with robust evaluations across both traditional (ROUGE) and modern (cosine similarity, GPT ratings) metrics.

Analysis

1.What Did I Do Well?

1. Successful implementation of two distinct summarization approaches.

2. Achieved meaningful results by combining multiple evaluation metrics.

3. Fine-tuned the T5 model effectively to generate high-quality summaries.

4. Effective data preprocessing and cleaning.

2.What Could I Have Done Better?

1. Use of larger datasets for better generalization.

2. Experimentation with advanced transformer architectures like GPT-4 for comparison.

3. Hyperparameter optimization to further improve performance.

3. Future Work:

1.Extend models to handle longer and more complex articles.

2. Integrate summarization capabilities into real-world applications via APIs.

3. Explore multimodal summarization involving text, images, and videos.

Conclusion:

This project successfully compared two abstractive summarization approaches: the LSTM-based encoder-decoder model and the T5 Transformer. The T5 model outperformed the LSTM model across all evaluation metrics, producing summaries that were semantically rich, lexically accurate, and more comprehensible. However, the LSTM model provided valuable insights into the traditional encoder-decoder architecture, particularly in understanding its strengths and limitations for text summarization tasks.

Extractive text summarization models focus on syntactic structure, while abstractive models like the ones used in this project emphasize semantics. Incorporating the strengths of both approaches could lead to even more effective summarization systems. The findings of this study highlight the superior performance of the T5 Transformer in generating coherent and human-like summaries.

Future work will address the limitations of both models, such as handling longer and more complex articles, and testing their scalability across diverse datasets and real-world applications. Additionally, exploring hybrid models that combine the syntactic accuracy of extractive approaches with the semantic depth of abstractive techniques could further enhance summarization quality.

References:

[1] Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., … Liu, P. J. (2019). Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. doi:10.48550/ARXIV.1910.10683

[2] Exploring Transfer Learning with T5: the Text-to-Text Transfer Transformer, Google AI Blog: https://ai.googleblog.com/2020/02/exploring-transfer-learning-with-t5.html.

[3] The Illustrated Transformer: Jay Alammar, Github: https://jalammar.github.io/illustrated-transformer/

[4] A. M. Rush, S. Chopra and J. Weston, “A Neural Attention Model for Abstractive Sentence Summarization”, 2015.

[5] Rush, A.M., Chopra, S. & Weston, J. (2015) A Neural Attention Model for Abstractive Sentence Summarization. 2015 Conference on Empirical Methods on Natural Language Processing (EMNLP).

[6] Yu, Hujia, Chang Yue and Chao Wang, “News Article Summarization with Attention-based Deep Recurrent Neural Networks” Stanford Natural Language Processing Group, Stanford University, pp.2746634, 2016.

[7] Iqbal, Touseef & Sambyal, Abhishek & Padha, Devanand. (2018), “A Review of Text Summarization using Gated Neural Networks”, INTERNATIONAL JOURNAL OF COMPUTER SCIENCES AND ENGINEERING, vol,6, pp. 5.

[8] Song, Shengli & Huang, Haitao & Ruan, Tongxiao. (2019). Abstractive text summarization using LSTM-CNN based deep learning. Multimedia Tools and Applications. 78. 10.1007/s11042-018-5749-3.

[9] Abhijeet Ramesh Thakare and Preeti Voditel, “Extractive Text Summarization Using LSTM-Based Encoder-Decoder Classification”, 2022 ECS Trans. Vol.107, pp.11665.

[10] News Summary, Kaggle:

https://www.kaggle.com/datasets/sunnysai12345/news-summary