Abstract

In this modern era of collection and utilization of data and information, it is important to extract concise yet comprehensive summaries from voluminous texts in order to understand it within a limited time. The Text Summarization System has been developed to generate abstractive summaries by understanding the entire document provided by the user, using a sophisticated and advanced approach, by utilizing the neural networks and the Transformer model, in order to employ the strategies and methods of deep learning to implement this summarizing system, that is able to provide the holistic understanding and core ideas of a document, which captures the essence of the document and be meaningful to the users.

This project has been intended to address the growing needs of effective and time-saving systems for proper and cohesive understanding of pieces of information. The system focuses on crafting summaries that encapsulates the crux of the original text, using the principles of deep learning and fundamentals of Natural Language Processing (NLP).

Keywords: Abstractive summary, Neural Network, Transformer model, Deep Learning, Natural Language Processing (NLP)

Table of Contents

List of Figures

List of Tables

List of abbreviations

Abbreviation	**Definition**
API	Application Programming Interface
3D	Three Dimensional
PNG	Portable Network Graphics
PDF	Portable Document Format
SVG	Scalable Vector Graphics
SGD	Stochastic Gradient Descent
RMSProp	Root Mean Squared Propagation
JWT	JSON Web Token
JSON	JavaScript Object Notation
GPU	Graphics Processing Unit
TPU	Tensor Processing Unit
SOS	Start-of-Sequence
EOS	End-of-Sequence
HTML	HyperText Markup Language
XML	Extensible Markup Language
NLP	Natural Language Processing
RNN	Recurrent Neural Network
BERT	Bidirectional Encoder Representations from Transformers
GPT-3	Generative Pre-trained Transformer
XLNet	eXtreme Learning Network
RoBERTa	Robustly optimized BERT approach

CHAPTER 1: INTRODUCTION

1.1 INTRODUCTION TO TEXT SUMMARIZATION:

The main objective of the project is to create abstractive summaries of news stories. Abstractive summarization involves studying the entire document, concentrating on its key ideas, and producing a summary that can accurately capture the ideas presented in the original text. Unlike the extraction summarization method, which identifies certain important lines and phrases in the source text and outputs them, this sort of summarization is more complicated. The abstractive summaries are usually generated after the entire document has been considered and understood, after which the core concepts underlying in the document are dissected and a proper summary is crafted that encapsulates the essence of the original text properly.

Therefore, to create abstractive summarization systems that provide a brief, coherent, and meaningful summary, more complex procedures and functions must be used. These systems are highly effective at interpreting texts in a condensed amount of words to save time and/or space. They can handle ambiguities, such as complex language styles or those that are present within the texts.

The implementation of the Text Summarization System involves training the model with a proper dataset from Kaggle. This system has been implemented by training the model using the already available dataset found in Kaggle due to a limited time constraint. The datasets containing news articles and their short highlights have been particularly used because it is more convenient to use such pairs, i.e., news and its summary, to train the model for this system. The datasets used for this project are as follows:-

1. Inshorts News Data (containing 55,104 rows)
2. CNN-DailyMail News Text Summarization (containing 287,113 training sets, 13,368 validation sets, and 11,490 test sets)

For this system, the news article related datasets have been selected, particularly because it is more convenient to get the news articles in their entirety along with their short highlights, which is just the type of dataset that needs to be trained for this text summarization system. Moreover, such datasets align seamlessly with the requirements of training a text summarization system, where the model learns to filter the essence of news stories into concise and informative summaries, for producing high-quality abstractive summaries. The link to the code in GitHub repository is as follows:-

https://github.com/SarinSthapit/Text-Summarization-System

1.2 OBJECTIVES:

The objectives of the Text Summarization System are as follows:-

1. To understand the concepts of neural networks and the Transformer model.
2. To study the workings of summarization systems using deep learning.
3. To implement the knowledge of neural networks and deep learning for the implementation of a text summarization system to generate abstractive summaries.
4. To explore the strategies that are used for optimizing the efficiency and performance of the text summarizing systems whilst taking care of the inference speed.

CHAPTER 2: DESIGN AND IMPLEMENTATION

2.1 SYSTEM REQUIREMENT SPECIFICATION

Software Specification:

The software and tools that will be used for the development of the system are as follows:-

1. **Pandas:**
   Pandas is a library that allows you to manipulate and analyze data efficiently. It offers a range of tools and functions for working with tables and time series. It provides two data structures: DataFrames, which are like tables with rows and columns, and Series, which are one-dimensional arrays with labeled indices. It is used for cleaning up data, performing in-depth analysis, and creating representations.

2. **Numpy:**
   NumPy is a tool used in computations. It offers a multidimensional array object and a variety of functions to manipulate arrays. It is used for tasks like linear algebra, statistical analysis, machine learning, and matrix operations such as matrix multiplication, determinant computation, and eigenvalue decomposition. Leading machine learning frameworks like Scikit-learn and TensorFlow rely on NumPy's handling of arrays, efficient data manipulation and computation.

3. **TensorFlow:**
   TensorFlow is an open-source software library for machine learning and artificial intelligence. It is mainly used for the training and inference of deep neural networks. It offers flexibility and scalability, making it suitable for a diverse range of machine learning applications. TensorFlow makes it easy for beginners and experts to create machine learning models for desktop, mobile, the web, and the cloud. Its cross-language compatibility widens its reach, and it provides high-level APIs, like Keras, that make creating and training deep neural networks easier.

4. **Matplotlib:**

   Matplotlib is an extensive library for creating high-quality static, animated, and interactive visualizations in Python. It supports a diverse set of plot types, including line plots, scatter plots, bar plots, histogram plots, pie charts, heat maps, 3D plots, etc. It can function in combination with other Python modules and frameworks, such as NumPy for numerical computing and Pandas for handling data. It is platform-independent and can work on various operating systems. Users can save plots in a variety of formats, including PNG, PDF, SVG, and others.

5. **Keras:**

   Keras is an open-source deep learning framework written in Python that acts as an interface to the machine learning platform TensorFlow. It is well-known for its simple and high-level API, which simplifies the development, training, and deployment of deep neural networks.

   Layers with individual parameters and activation functions can be stacked to generate models. Building complicated neural network structures is made easier by this modular design. To train neural networks, it provides several optimization algorithms, such as stochastic gradient descent (SGD), Adam, RMSprop, and others. These optimizers are simple to configure and fine-tune for users.

6. **Scikit-learn:**

   Scikit-learn is a powerful and efficient tool for tasks ranging from classification and regression to clustering and dimensionality reduction. It is open-source and easy to use. It has data preprocessing capabilities such as data scaling, categorical variable encoding, and missing value handling, which are critical when it comes to preparing data for machine learning models. It provides a variety of model evaluation methods, including cross-validation, hyperparameter tuning, and metrics for classification, regression, clustering, and more. It makes the process of evaluating model performance easier.

7. **Google Colab:**

   Google Colab, short for Google Colaboratory, is a free cloud-based platform offered by Google that allows users to run Python code in an environment similar to Jupyter Notebook. It's especially popular among data scientists and machine learning experts. It gives users free access to computational resources like GPUs and TPUs and allows them to collaborate in real-time with others. It comes pre-installed with numerous commonly used Python libraries, including NumPy, Pandas, Matplotlib, and TensorFlow, saving time and effort over installing these libraries locally. Colab notebooks may be saved straight to Google Drive, and you can also use Colab to access datasets and files stored in your Google Drive.

8. **Kaggle:**

   Kaggle is an online platform for individuals, researchers, and professionals interested in data science and machine learning. It provides a collection of publicly accessible datasets on a variety of topics. It is a significant resource for data exploration, analysis, and research because users can upload and share datasets. Kaggle Kernels are a Jupyter Notebook environment that allows users to write, run, and collaborate on code in the cloud. Kernels are useful for exploring datasets, building models, and sharing reproducible data analysis. They support a range of programming languages, including Python and R. Kaggle also provides free access to GPUs and TPUs to participants, making it easier for them to construct and train machine-learning models that require massive computational resources.

9. **Anvil:**

   Anvil is an online platform that allows its users to build beautiful and engaging web applications using Python programming language, and simplifies the entire development process by providing a framework for building web applications using simple Python codes, both on client and server sides. This platform allows easy drag-and-drop interface, where components can be added to the interface, and their properties can basically be changed directly on the properties tab. This platform was particularly chosen for web app development to learn about the newer streamlined approach to web application development which speeds up the development process, and integrates easily with the Python codes.

Hardware Specification:

For the development of this system, no high-performance hardware was required as the model was developed and implemented in Google Colab and Kaggle, and these platforms provided powerful GPUs for training the model in less time. This system can be used by any regular user who possesses a modern device that supports the developed web app.

2.2 DESIGN AND IMPLEMENTATION OF THE SYSTEM

The basic steps behind the development of this project have been explained in the following steps:-

1. **Data Preprocessing:**

   First of all, the datasets contain the news stories and their summaries, along with other relevant information regarding the news. This dataset needs to be loaded so that only the news stories and their summaries remain. Then, the tokens are added to identify the beginning and end of the summaries. For this project, the sentences were converted to tokens using Keras Tokenizer, which mapped words to tokens after filtering the punctuations and converting the text to lowercase. This data preprocessing step is explained in detail below.

   To begin, the datasets chosen for this project, Inshorts News Data and CNN-DailyMail News Text Summarization, were combined into a single Excel workbook. This dataset includes news articles and their summaries in each row, as well as other information. After reading the dataset file, the unnecessary columns were removed, leaving only the news articles and their summaries. The lambda function is then applied to each article and its summary to mark the Start of Sentence with <SOS> and the End of Sentence with <EOS>.

The texts with start and end marks are again stored back in their respective variables. Then the *preprocess* function is used to match the regular expression with HTML or XML character entities used to represent the special characters, and such characters are replaced with a space character, which means that such characters are removed and the modified text is again stored.

The texts (i.e., both news articles and their respective summaries) are tokenized using the TensorFlow library's 'Tokenizer' class to convert them into sequences of integers. The set of characters that are exclamation marks (!), Quotation mark or double quote ("), Hash symbol (#), Dollar sign ($), Percent sign (%), Ampersand (&), Parentheses, Asterisk or star (*), Plus sign (+), Comma (,), Hyphen or minus sign (-), Period or dot (.), Forward slash (/), Colon (:), Semicolon (;), Equal sign (=), Question mark (?), At symbol (@), Square brackets ([ ]), Circumflex accent (^), Underscore (_), Curly braces ({ }), Vertical bar (|), Tab characters (\t), Newline characters (\n), Tilde (~), Backtick (`), and Backslash (\\). are removed by substituting them with white space.

Furthermore, special tokens are replaced with the 'oov_token', also known as the Out of Vocabulary. Then the article and summary were tokenized. Then, 'inputs' and 'targets' were used to store the sequences of integers representing the article texts and summary texts, respectively.

2. Model Architecture:

The implementation of the system will be done using the Transformer model, which is a comparatively newer network model proposed in 2017 and has been widely used due to its extensive capability of capturing the details lying between the word sequences. A transformer is a deep learning architecture that processes text sequences using an attention mechanism. It has served as the foundation for many cutting-edge natural language processing (NLP) tasks, such as machine translation, text summarization, sentiment analysis, etc. The attention mechanism is the primary innovation of the transformer. When generating an output, a model can use attention to focus on distinct parts of an input sequence. This approach, which enables transformers to capture long-term dependencies in language, is critical to their success. Some examples of NLP transformers are BERT (Bidirectional Encoder Representations from Transformer), GPT-3 (Generative Pre-trained Transformer 3), XLNet, RoBERTa, and T5. One of the primary benefits of transformers is their capacity to enable transfer learning. With very small datasets, pre-trained transformer models can be fine-tuned for specific NLP tasks, minimizing the requirement for vast volumes of task-specific labeled data. The Transformer model is a form of semi-supervised learning which means that these models are pre-trained in an unsupervised manner, with a large unlabelled dataset, and they are later fine tuned through supervised training for better performance. The figure below shows the architecture of the Transformer model:-

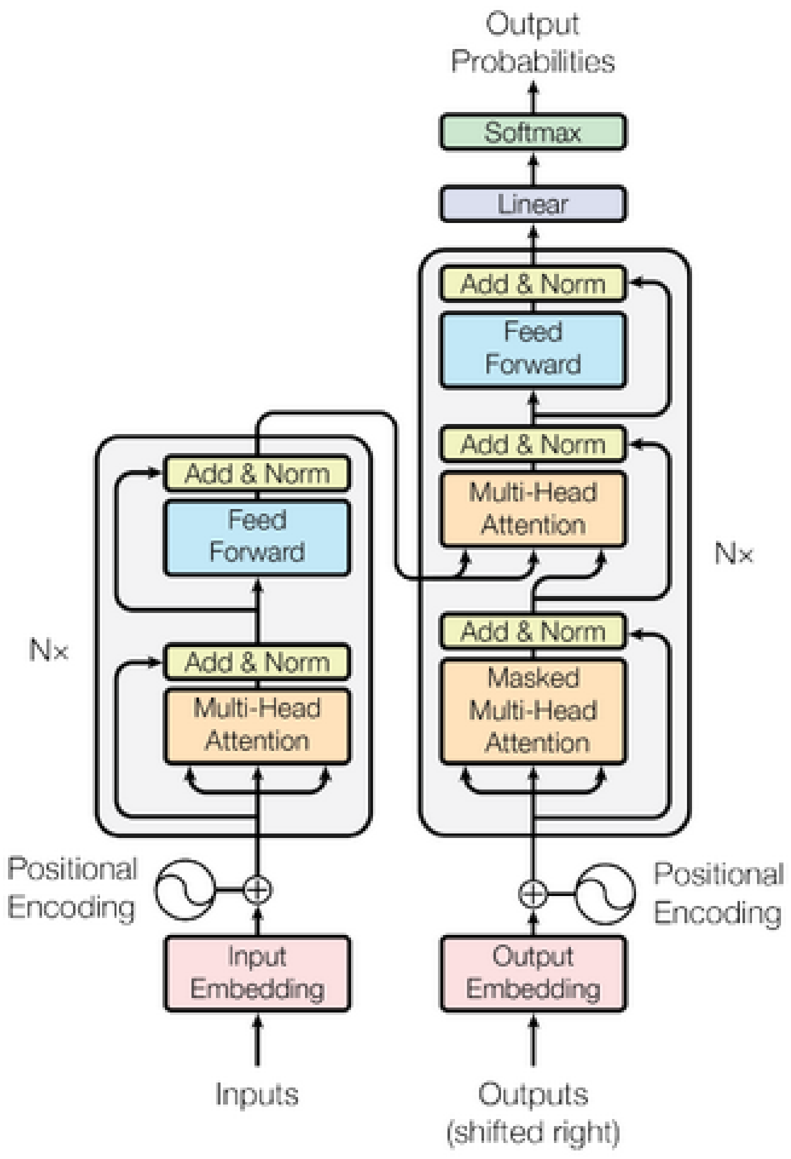


Figure 2.2.1: Architecture of Transformer model

The Transformer model uses an attention mechanism, so the data need not be processed in a sequential manner, rather this mechanism provides some context around the items in the input sequence. The use of this mechanism is the main reason that the Transformer model is preferred more than the RNNs. Moreover, this model runs multiple sequences in parallel, due to which the training time speeds up significantly. The attention mechanisms are used by the encoder and decoder to weigh different parts of the input sequence for encoder, and the arts of already generated output sequence for the decoder, to selectively concentrate on the relevant information, which makes the model capable of capturing long-range dependencies and improving the generation of coherent and contextually relevant summaries. The Transformer model uses the encoder and decoder blocks, as well as the point-wise feed-forward network. The Transformer model can more effectively comprehend the relationships between words in any sequence, its application in Natural Language Processing (NLP) tasks like question answering, machine translation, and text summarization because of which this model has been used for the text summarization system. Afterwards, the tokens from article texts were mapped to unique integer IDs using a dictionary, such that every token is assigned some unique integer ID in the vocabulary. Then, the number of unique tokens in article texts was counted, and one was added to it for the special 'oov_token' in it. This value was assigned to the 'ENCODER_VOCAB' variable to denote the size of the vocabulary for the encoder. Similarly, the tokens from summary texts were mapped to unique integer IDs using a different dictionary, so that each token is assigned some unique integer ID in the vocabulary. Then, the number of unique tokens in summary texts was counted, and one was added to it for the special 'oov_token' in it. This value was assigned to the 'DECODER_VOCAB' for the size of the vocabulary for the decoder. Finally, the 'inputs' and 'targets' variables were modified such that they have a uniform sequence length. Therefore, longer sequences are truncated, and shorter sequences are padded with zeros.

The encoder and decoder are two interdependent components that are used to process and generate sequences of data for text summarization. Each of them have been discussed below:-

A. Encoder:

The encoder is used to process any input sequence and extract meaningful information from it. It applies self-attention mechanisms to find dependencies and relationships between different input parts. The self-attention mechanism focuses on relevant information and considers context from all positions in the input sequence. When processed in feedforward neural networks, the outputs of encoders can be used to capture several complicated patterns and features within them. The final output of the encoder is fed into the decoder to generate the target sequence. The encoder block comprises several layers of self-attention mechanisms and feedforward neural networks. This self-attention mechanism allows the model to weigh different parts of the input sequence differently when the information is encoded, in such a way that the relationship and context between the adjacent or corresponding words within the source text is easily understood. The structure of an encoder is shown in the figure below:-

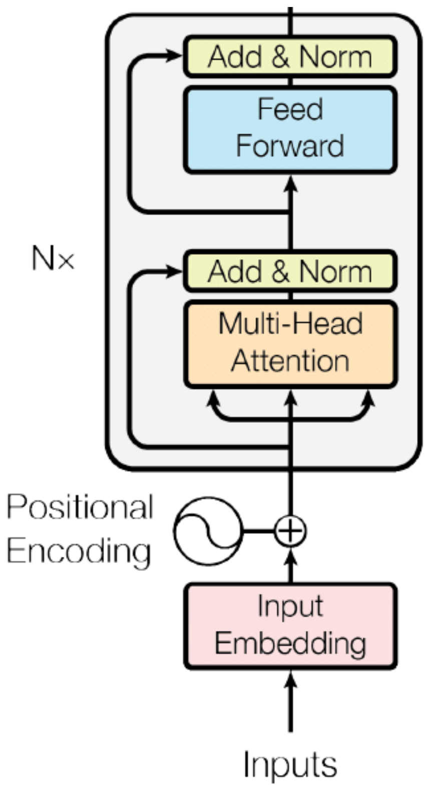


Figure 2.2.2: Structure of Encoder in Transformers model

The input sequence is initially converted to continuous vector representations called embeddings, which include positional information to maintain the order of words in sequence. This is called Input Embedding. To understand the order of sequences, positional encoding is added to Input Embedding in order to provide information about positions of words in sequence. The most important component is the Multi-Head Attention to compute the attention scores for each word based on its relationship and dependency with other words. This output along with input embeddings and layer normalization is applied. Then the output is processed by a feedforward network, and the results are again added to input and normalized. In this way, the encoder can focus on the contextual information and generate continuous representation for input sequence, for the generation of abstractive summaries.

B. Decoder:

The decoder utilizes a sequence of context-aware representations, which are output by the encoder, to generate the output sequence, which includes multiple layers of self-attention and feedforward neural networks. The decoder uses its previous output and the output of the encoder to ensure that the sequence that has been generated is relevant and coherent. Masked self-attention is used during the training to avoid looking ahead in the output sequence. The decoder's output is a predicted sequence that generates one token at a time, where probability over the vocabulary at each time step is produced to select the most likely token to be selected for the next output. The decoder block also comprises several layers of self-attention mechanisms and feedforward neural networks. Here, throughout the decoding process, the self-attention mechanism helps the model to concentrate on the parts of the input sequence, which allows it to generate the summary by attending to the relevant information in the source text. The decoder of a transformers model has a layered structure, containing several sub-components for processing the input and generation of the output sequence. First of all, the input sequence is converted to embeddings supplemented by positional encoding to preserve the word order. Then a masked multi-head attention mechanism is used to attend to different parts of the generated output sequence. Outputs from them are added to the input embeddings and then layer normalization is applied. Moreover, position-wise feedforward networks are employed to process the result after which the final layer produces a probability distribution over the vocabulary for each position in the output sequence. In this way the decoder is used to generate coherent and contextually relevant output sequences for generation of abstractive summaries from any text input. The structure of an decoder is shown in the figure below:-

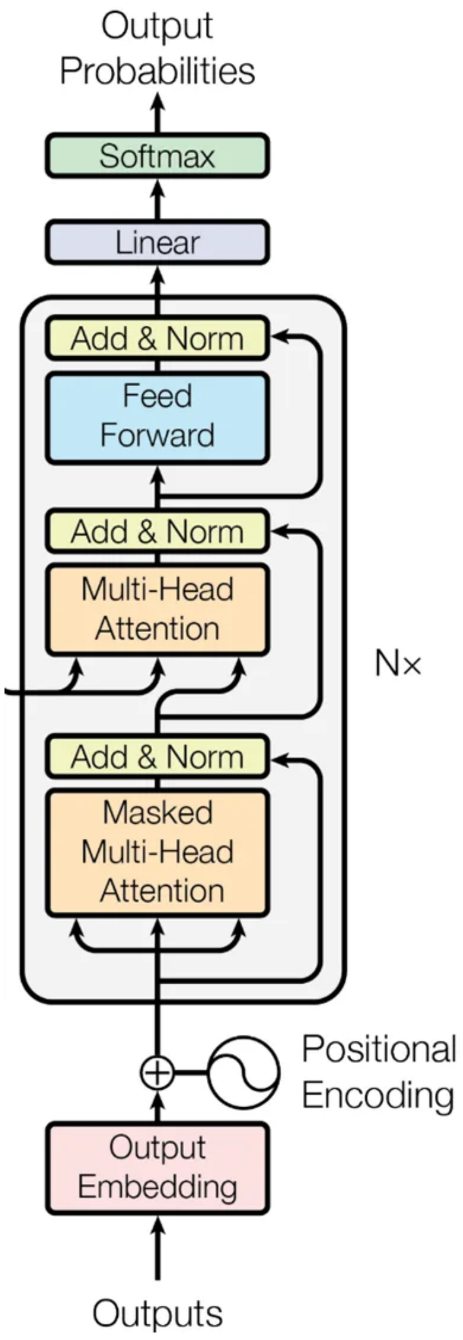


Figure 2.2.3: Structure of Decoder in Transformers model

3. **Defining the utility functions:**

The utility functions for further steps are defined. The following functions are used for this text summarization system:-

A. ***calculate_angles* function:**

This function is used to calculate the angles for positional encoding, to provide the model with information about the positions of words or tokens within a sequence, and to consider the order of the words in the input data.

B. ***generate_positional_encoding* function:**

The purpose of this function is to identify the positional encodings for a given sequence of positions in the word embeddings of input tokens to provide the model with information about the positions of tokens within a sequence. This function is used to generate the positional encodings that can be added to the word embeddings of the tokens in the input sequence so that the model can distinguish the positions of the tokens, allowing the model to capture the sequential information and understand the order of the words within the input sequence.

C. ***generate_padding_mask* function:**

This function is used for the creation of a padding mask for a sequence to indicate whether the elements in the sequence are padding or not. These padding masks are used to prevent the model from attending to padding tokens when processing the input sequences.

D. ***generate_look_ahead_mask* function:**

This function is used for the creation of a padding mask to prevent elements in a sequence from attending to future elements in that sequence when the self-attention mechanism is applied. During the self-attention calculation, these masks are used to ensure that each element in a sequence can only attend to preceding elements or itself. These masks prevent the leakage of information from future positions by performing zero out of the attention weights for future positions, by applying the scaled dot-product attention calculation.

E. *compute_scaled_dot_product* function:

This function is used to implement a scaled dot-product attention mechanism to calculate the attention scores for the computation of an output and attention weights. This function is important for modeling long-range dependencies and relationships in the sequence.

F. *built_pointwise_feedforward_network* function:

This function is used for the definition of a feedforward neural network used for each position in the sequence independently to introduce non-linearity and for the recognition of complex patterns within the data. This is used for adding expressiveness to the model for modeling the complex dependencies. This function uses the 'd_model' parameter, which is basically the parameter to specify the dimensionality of the model and is used to set the same dimension as the embeddings of the model.

G. *generate_masks* function:

This function is used for the creation of numerous masks for the sequence-to-sequence models to control the attention mechanism and masking out certain elements during the process of training. These masks function to guide the self-attention mechanism in both encoder and decoder, in order to improve the relevance and correctness of the generated sequence, thus, improving the summarizing quality of the model.

4. Definition of custom learning rate schedule, loss function and accuracy function:

A custom learning rate schedule and custom loss functions have to be defined to train the model. The custom learning rate scheduler would allow quicker convergence, as it regulates the step size that is used to update the model weights during the training, i.e.

the learning rate. The Transformer model requires different learning rates at different stages of training, due to which the custom learning rate schedule tends to be very beneficial for this model. To improve the convergence of the model and achieve a better performance measure, the implemented custom learning rate schedule was used to adjust the learning rate during training. The purpose of a custom learning rate schedule is mainly to gradually increase the learning rate at the beginning of the training, which is also known as the warm-up phase. Gradually, the learning rate decreases with the reciprocal square root as the training progresses. This efficiently works to stabilize the model and ensures that the model converges effectively during training when using the Transformer model, where the learning rate plays a vital role. To train the model, a custom learning rate scheduler and custom loss functions were defined. The custom learning rate scheduler, which regulates the step size used to update the model weights during training, i.e., the learning rate, allows for faster convergence. The Transformer model finds significant advantages in a customized learning rate schedule since it needs distinct learning rates at different training phases.

On the other hand, a loss function, also called the cost function or objective function, can indicate how well the predictions made by the network match the actual target values. Moreover, the loss function is also a fundamental measure of the difference between the predicted values and the true values. In text summarizations, the loss functions are used to measure the difference between the predicted summary and the actual summary. The gradient of the loss during the training, concerning the model's parameters, can be used for weight updations by utilizing the concept of backpropagation. The degree of dissimilarity between the true target sequence and prediction of the model was measured using a sequence-to-sequence loss function, without considering the effect of padding tokens. The loss function is very important for the Transformers model, where the computed loss is used for comparing the predictions to the actual target sequences, after which backpropagation is performed to reduce the internal parameters to reduce the loss.

Additionally, an accuracy function is used to compute the accuracy of the predictions of the model and is used to evaluate the performance of the model to determine the degree of correctness of the model in capturing the content and structure of the target sequence. Here, the implemented accuracy function was used to compute the sequence-to-sequence prediction of the model for assessing the correctness of the generated sequences while considering the effects of padding tokens for fair evaluation.

5. **Training the model:**

   For training the model, a specified number of epochs is defined, after which the model is trained on the specified dataset using the transformer-based model. Any model's accuracy is based upon the quality and quantity of the dataset, in addition to several hyperparameters that must be appropriately calibrated. For this model, a single step of training was performed for the updation of the parameters of the model based on the gradients whilst also considering the training loss and accuracy.

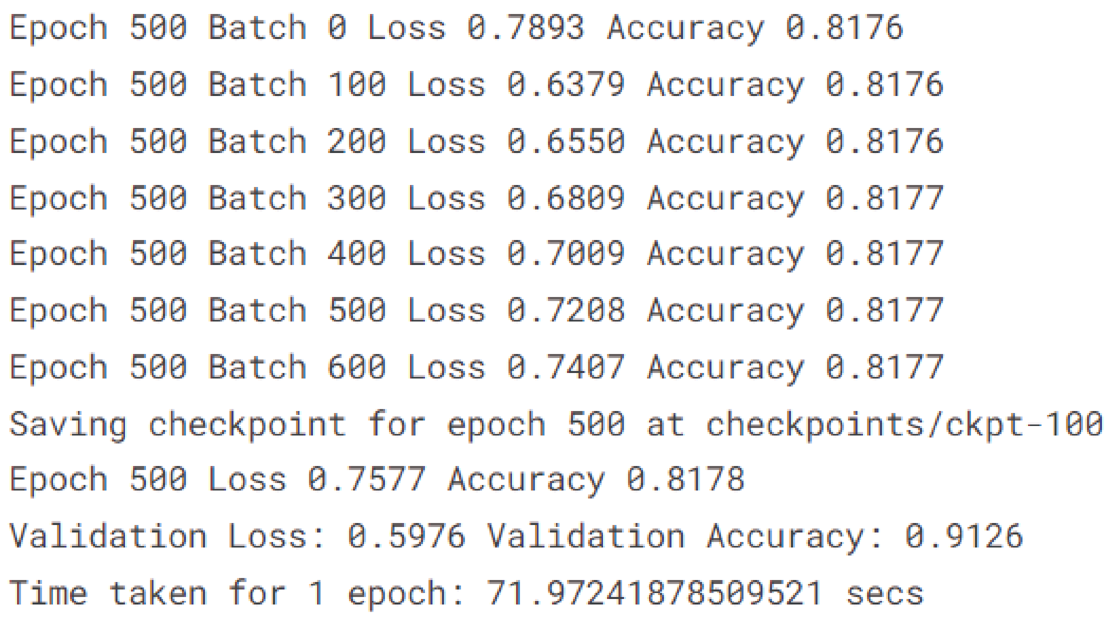
6. **Fine-tuning the hyperparameters:**

   Hyperparameters are the configuration settings for a model whose value is used to control the learning process. By modifying these parameters, the accuracy of the model can be improved. Therefore, choosing the correct hyperparameters is a key part of developing the model. Some examples of hyperparameters are learning rates, batch sizes, the number of hidden layers in a neural network, etc. Fine-tuning of hyperparameters includes gradually modifying these hyperparameters to improve the model's performance. Some techniques used for hyperparameter tuning are grid search, random search, Bayesian optimization, gradient-based optimization, etc. There are also some automated tools available that can efficiently perform the hyperparameter tuning process. The model that has been developed up to present change will be further improved by experimenting with the hyperparameter settings, which includes trying out different combinations of learning rates, batch sizes and model depth to find the most appropriate combination that yields the maximum accuracy. The parameters for any model are basically internal and can be estimated and adjusted based on the data, but the hyperparameters such as the learning rate, are external, cannot be estimated from the data and need to be manually specified and tuned, because of which this is an important and equally ambiguous task when tuning such a deep learning model. Basically, the hyperparameter fine-tuning is used to optimize the model. This can be done simply by showing the dataset to the model multiple times, i.e., increasing the number of epochs, or adjusting the learning rate.

CHAPTER 3: DISCUSSION ON THE ACHIEVEMENTS

3.1 EVALUATION OF THE SYSTEM:

The text summarization system has been evaluated based on its accuracy, which is 0.8177 for the training set and 0.9126 for the validation set when trained for 500 epochs, as shown in the figure below:-



```
Epoch 500 Batch 0 Loss 0.7893 Accuracy 0.8176
Epoch 500 Batch 100 Loss 0.6379 Accuracy 0.8176
Epoch 500 Batch 200 Loss 0.6550 Accuracy 0.8176
Epoch 500 Batch 300 Loss 0.6809 Accuracy 0.8177
Epoch 500 Batch 400 Loss 0.7009 Accuracy 0.8177
Epoch 500 Batch 500 Loss 0.7208 Accuracy 0.8177
Epoch 500 Batch 600 Loss 0.7407 Accuracy 0.8177
Saving checkpoint for epoch 500 at checkpoints/ckpt-100
Epoch 500 Loss 0.7577 Accuracy 0.8178
Validation Loss: 0.5976 Validation Accuracy: 0.9126
Time taken for 1 epoch: 71.97241878509521 secs
```

Figure 3.1.1: Accuracy results for 500 epochs

It is crucial to assess the efficiency of text summarizing models. It aids in evaluating the models' ability to provide summaries that comprehensively cover all relevant information included in the original text. For summarizing tasks, many evaluation metrics are often used in the field of natural language processing. Several assessment measures include METEOR (Metric for assessment of Translation with Explicit ORdering), BLEU (Bilingual Evaluation Understudy), ROUGE (Recall-Oriented Understudy for Gisting Evaluation), ROUGE-L (Longest Common Subsequence), Precision, Recall, and F1 Score. In this project, BLEU and ROUGE have been used.

ROUGE is a widely used metric to evaluate the quality of summaries by comparing them to reference summaries. It consists of several sub-metrics, such as ROUGE-N (precision, recall,

and F1 score based on n-grams), ROUGE-L (longest common subsequence), and ROUGE-W (weighted longest common subsequence). The ROUGE scores achieved are:-
ROUGE-N (Unigram): Precision = 0.5, Recall = 0.5, F1 = 0.4999999950000001
ROUGE-L: Precision = 0.5, Recall = 0.5, F1 = 0.4999999950000001

Precision, recall, and F1 score should ideally be as close to 1 as is feasible for a well-performing system. When comparing the generated and reference summaries for matching unigrams or longest common subsequences, a score of 0.5 indicates that there is room for improvement.

BLEU is widely used to evaluate machine-generated text by comparing it with one or more reference texts. It determines accuracy by comparing n-grams—usually up to 4-grams—between the generated summary and the reference summaries. A BLEU score of 0.4141 was achieved. The ideal match has a BLEU score of 1. The possible scores are 0 to 1. In this case, a BLEU score of 0.4141 indicates that although there is some overlap between the reference summaries and the generated summary, there is still room for development.

3.2 LOSS CURVE:

The figure below represents the loss curve of the model, showing both training loss and validation loss. In this model, it can be seen from the given figure that both training loss and validation loss decrease exponentially as the number of epochs increases.

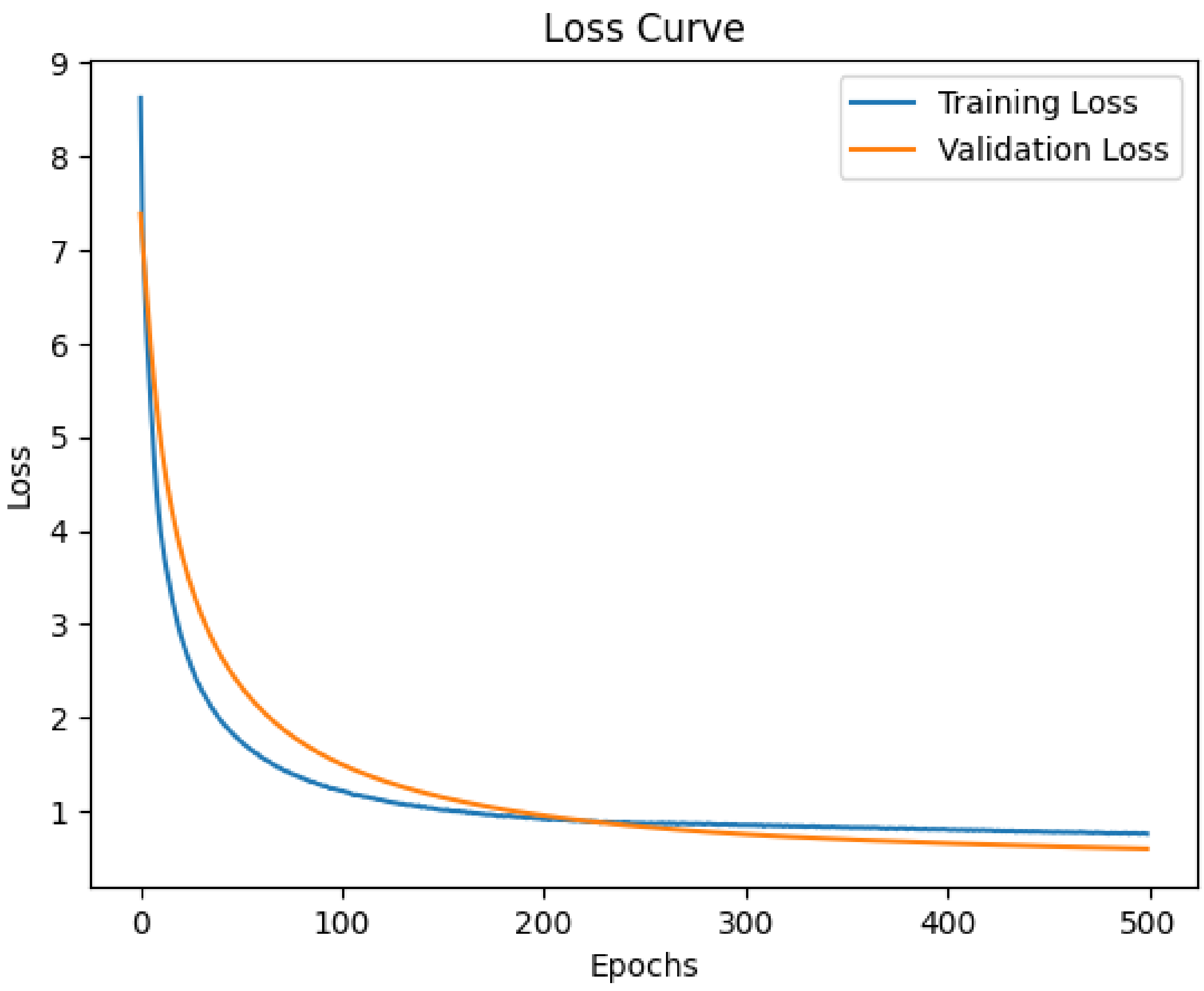


Figure 3.2.1: Loss curve showing loss versus number of epochs

3.3 ACCURACY CURVE:

The figure below represents the accuracy curve that was obtained while training the model. In this model, it can be seen from the given figure that the accuracy increases logarithmically as the number of epochs increases.

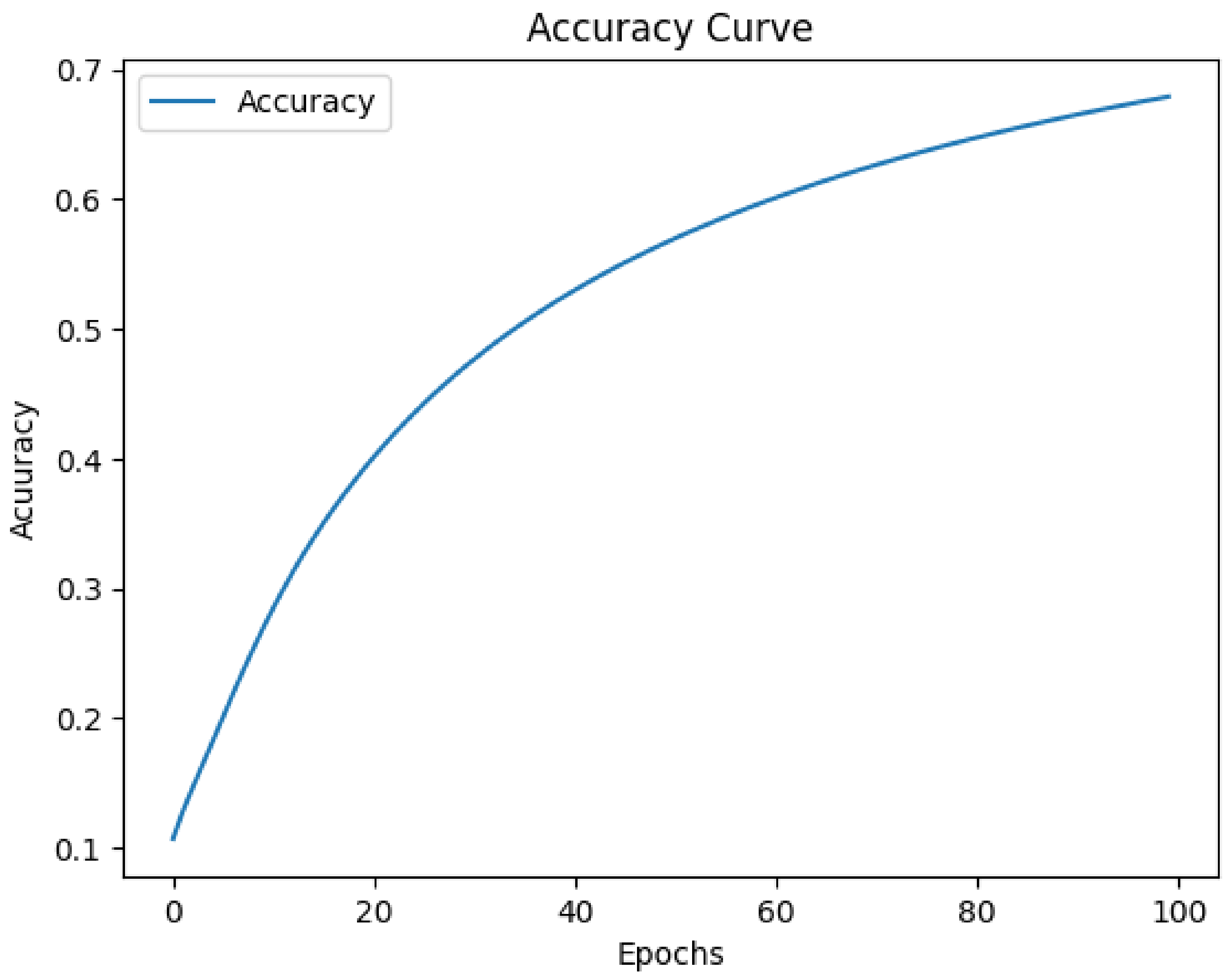


Figure 3.3.1: Accuracy curve showing accuracy versus number of epochs

3.4 WEB APP:

The web app has been developed with the intent to ease the users to get abstractive summaries simply by providing a text whose summaries are required and pressing the submit button. Then, the web app will use the model developed to provide an output, i.e., an abstractive summary of the input text. A preview of the web app is shown in the figure below:-

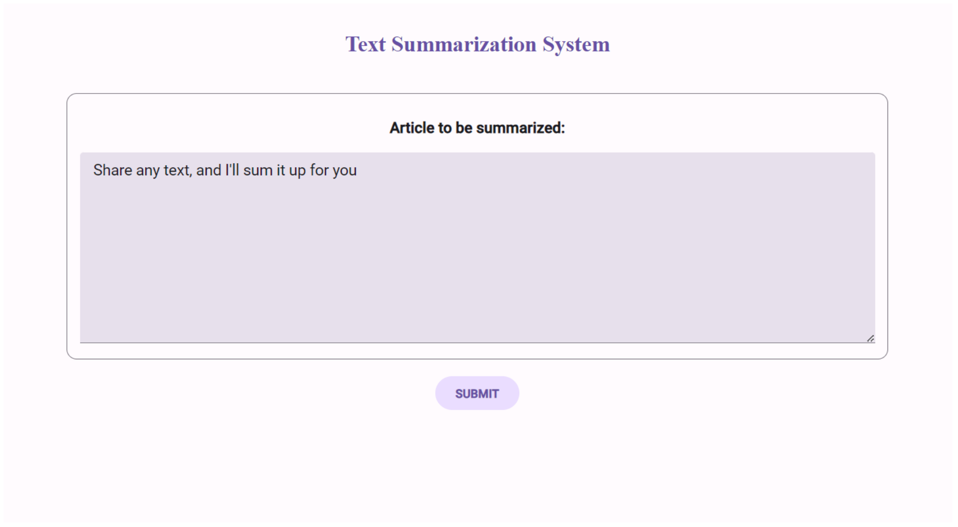
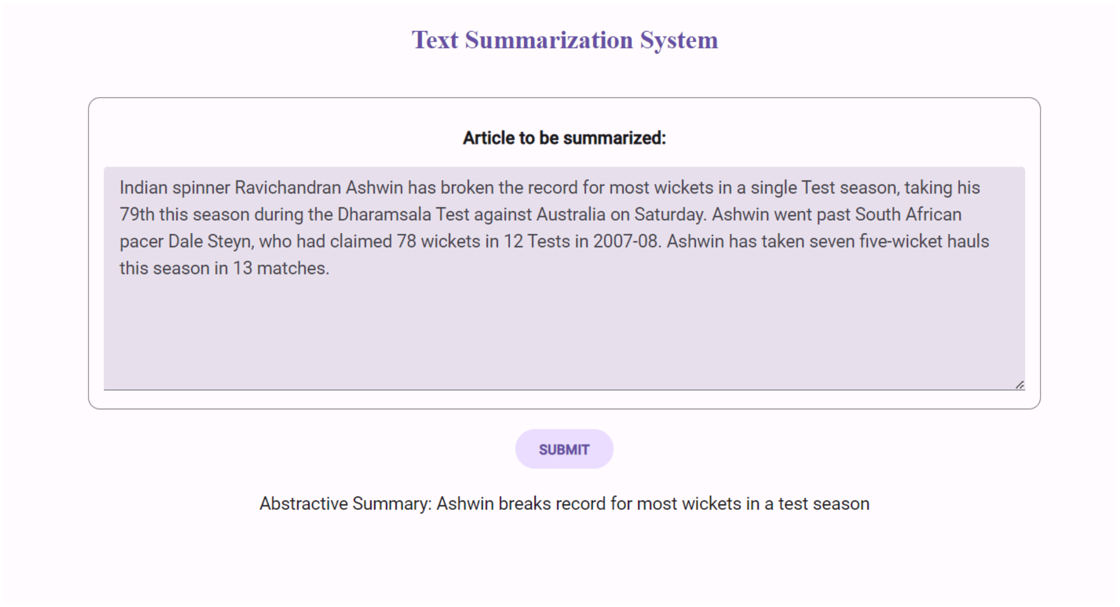

Text Summarization System
Article to be summarized:
Share any text, and I'll sum it up for you
SUBMIT

Figure 3.4.1: Text Summarization System web app interface

3.5 SYSTEM RESULTS:

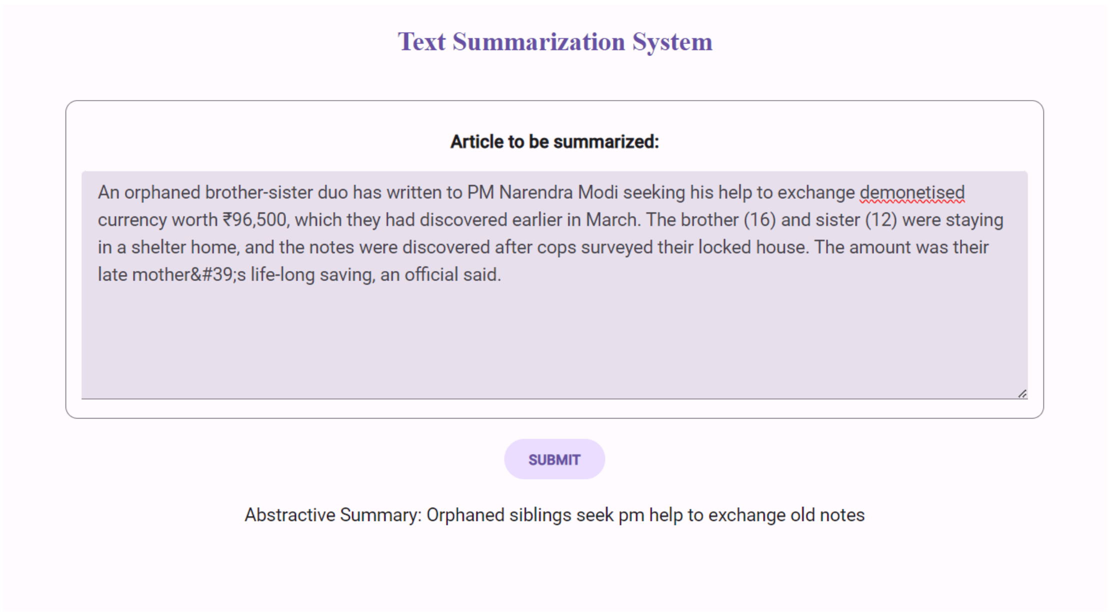
The results obtained from the Text Summarization System are shown below:-



Indian spinner Ravichandran Ashwin has broken the record for most wickets in a single Test season, taking his 79th this season during the Dharamsala Test against Australia on Saturday. Ashwin went past South African pacer Dale Steyn, who had claimed 78 wickets in 12 Tests in 2007-08. Ashwin has taken seven five-wicket hauls this season in 13 matches.

Abstractive Summary: Ashwin breaks record for most wickets in a test season

Figure 3.5.1: Summarization Result 1

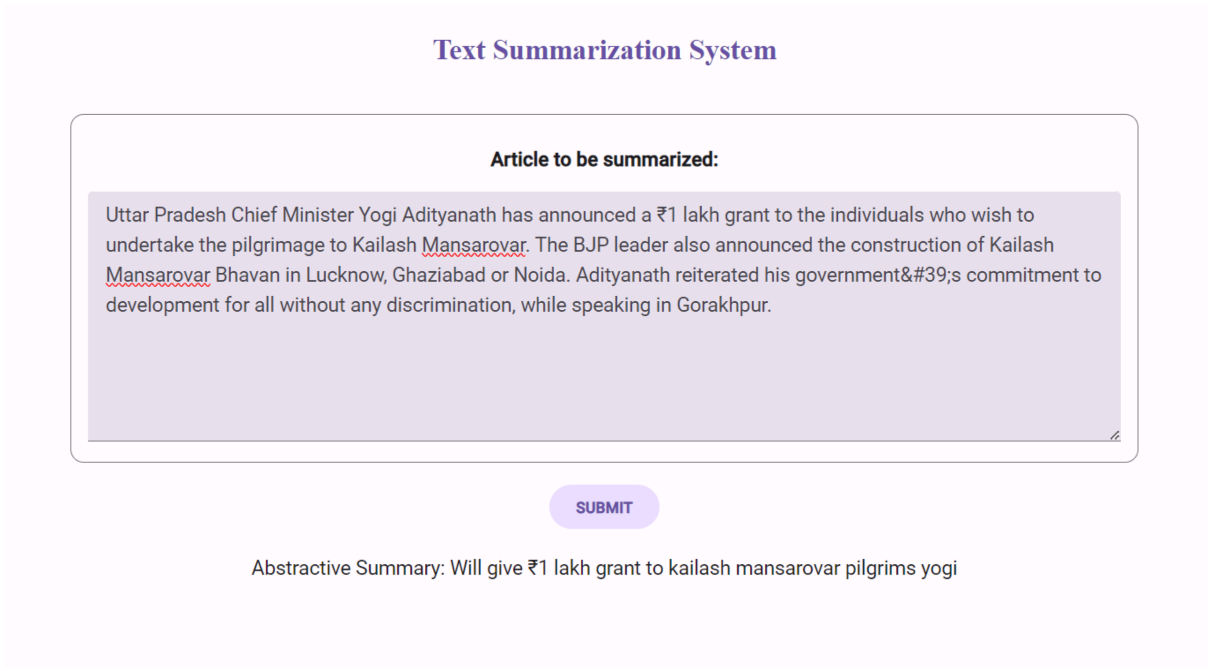

Text Summarization System

Article to be summarized:

An orphaned brother-sister duo has written to PM Narendra Modi seeking his help to exchange demonetised currency worth ₹96,500, which they had discovered earlier in March. The brother (16) and sister (12) were staying in a shelter home, and the notes were discovered after cops surveyed their locked house. The amount was their late mother&#39;s life-long saving, an official said.

Abstractive Summary: Orphaned siblings seek pm help to exchange old notes

Figure 3.5.2: Summarization Result 2


Text Summarization System

Article to be summarized:

Uttar Pradesh Chief Minister Yogi Adityanath has announced a ₹1 lakh grant to the individuals who wish to undertake the pilgrimage to Kailash Mansarovar. The BJP leader also announced the construction of Kailash Mansarovar Bhavan in Lucknow, Ghaziabad or Noida. Adityanath reiterated his government&#39;s commitment to development for all without any discrimination, while speaking in Gorakhpur.

Abstractive Summary: Will give ₹1 lakh grant to kailash mansarovar pilgrims yogi

Figure 3.5.3: Summarization Result 3

CHAPTER 4: CONCLUSION AND RECOMMENDATION

4.1 LIMITATIONS:

Most of the important objectives of the project were achieved. However, due to limited time constraints, there were a few shortcomings in the system. The limitations of Text Summarization System are as follows:-

1. It was very difficult to find the free services that could meet the storage requirements for the large dependencies in the backend, so Anvil Editor was used along with the Kaggle notebook to produce the summaries whenever long texts are provided as input.
2. An accuracy of 0.8177 for the training set and 0.9126 for the validation set when trained for 500 epochs was obtained, which is quite satisfactory, but it is still possible to further improve the accuracy of the model.
3. The system is still unable to generate correct punctuation marks when generating the summary, as these were replaced with white spaces by TensorFlow's 'Tokenizer' class during the process of Tokenization.

4.2 FUTURE ENHANCEMENTS:

Some of the future enhancements that have been planned for the Text Summarization System are as follows:-

1. Improving the accuracy of the system significantly for the generation of the most appropriate abstractive summary for any input text.
2. Hosting the web app on a proper hosting platform for easy access by regular users.
3. Enhancing the model and the system in its entirety further more so that it can generate summaries with correct punctuation marks.

4.3 GANTT CHART:

The diagram below represents a Gantt Chart for representing all the stages of the project with respect to the time allocated:-

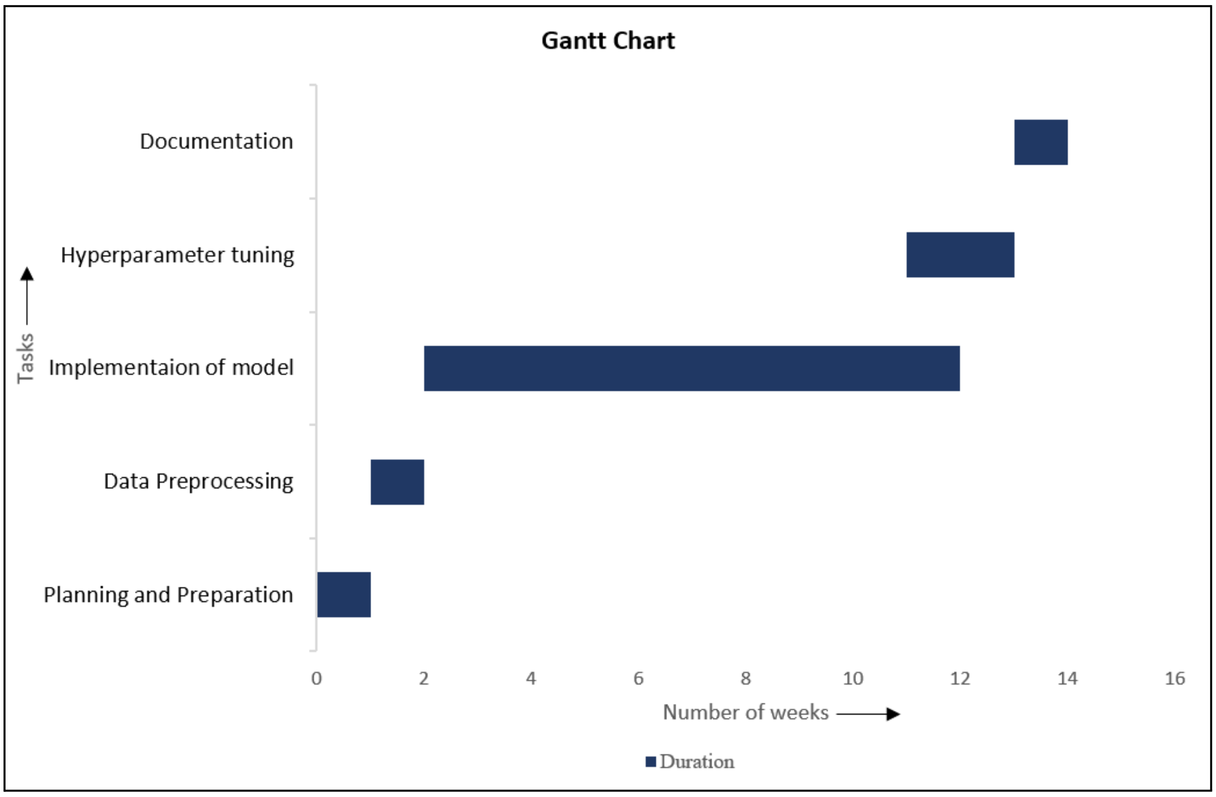


Table 4.3.1: Gantt Chart

The table below shows the tasks carried out and the scheduled time duration for the corresponding tasks:-

S. No.	Tasks	Duration
1.	Planning and Preparation	September 3, 2023 - September 9, 2023
2.	Data Preprocessing	September 10, 2023 - September 16, 2023
3.	Implementation of model	September 17, 2023 - November 25, 2023
4.	Hyperparameter tuning	November 19, 2023 - December 2, 2023
5.	Documentation	December 3, 2023 - December 9, 2023

Table 4.3.2: Project Timeline Table

REFERENCES

Intro to Transformers: The Decoder Block. Edlitera. (n.d.). https://www.edlitera.com/blog/posts/transformers-decoder-block

Joshi, P. (2023a, November 7). *How do transformers work in NLP? A guide to the latest state-of-the-art models*. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2019/06/understanding-transformers-nlp-state-of-the-art-models/

The structure of vision transformer encoder. - researchgate. (n.d.). https://www.researchgate.net/figure/The-structure-of-Vision-Transformer-Encoder_fig1_366911975