Content Summarization using BERT

By

Aditya Tushar Wadnerkar

Rimzim Thube

Shree Gowri Radhakrishna

Vijaylaxmi Nagurkar

Table of Contents

[1. Abstract 3](#_Toc58879858)

[2. Experimentation 4](#_Toc58879859)

[3. Model 9](#_Toc58879860)

[4. Data 12](#_Toc58879861)

[5. Methods 13](#_Toc58879862)

[6. References 18](#_Toc58879863)

# Abstract

With the advancement of technology, we are now able to get access to information equivalent to the information stored in millions of books. We can get knowledge about anything on the internet with a click of a button. However, with the amount of information that we get from a search engine it is very difficult to go through each section to filter the documents based on the relevance of the summary of the article.

There are traditional ways of highlighting important points from the content. Some articles do have a short one page abstract or summary that describes the literature in a precise format. However, with time constraints sometimes it becomes a tedious task to go through the entire one-page summary when we just want to extract the key features from the content.

At the same time, we come across so many online publications that we want to get information from. Be it news websites or blogs, we cannot go on reading the entire post when we just want the bytes or small clippings from the article. Moreover, the post might contain much detailed description on each topic being discussed in it. It could also contain quotes from people along with their reaction to a particular news, which we might want to skip as we just want the gist of the scenario.

In the case of YouTube videos some clips do have a small description of what is discussed in the video, but they do not reveal what conclusions they have derived for the topic. Moreover, there might be some important things they have discussed in the video but did not mention them in the description. However, videos which are of longer duration, like one hour or more, can be viewed at faster speeds, only to cut the time required by half. That means it will still take more than half an hour to even go through the content and take notes.

Text summarization is one of the most challenging tasks in the field of natural language processing for performing language generation and information compression. To solve this problem, we designed a web application that could summarize the information delivered by the article. It is a python based end-to-end deep learning web application. It uses the BERT model to generate a summary of the article. The application takes input from the user in the form of a text or web link and produces the summary of the article. It also provides a word cloud of highlights of important keywords extracted from the text to relate with the context of the article.

# Experimentation

There are two main forms of Text Summarization, extractive and abstractive:

1. Extractive: A method to algorithmically find the most informative sentences within a large body of text which are used to form a summary.
2. Abstractive: A method to algorithmically generate concise phrases that are semantically consistent with the large body of text. This is far more in line with how humans summarize text but is also far more difficult to implement.

There have been many methods to create text summarization in the past. In this project, we experimented with different methods of both *extractive and abstractive summarization* and evaluated them against a golden benchmark to identify their performance. The methods start from a basic model and extends to a more complex transformer models and are as follows:

1. Word frequency model
2. Text rank model
3. BERT extractive summarizer
4. T5 transformer model
   1. **Rouge Score Analysis**

ROUGE stands for Recall-Oriented Understudy for Gisting Evaluation. It is essentially a set of metrics for evaluating automatic summarization of texts as well as machine translation. It works by comparing an automatically produced summary or translation against a set of reference summaries (typically human produced). In order to quantify the effectiveness of summarization, we have relied up on ROUGE scores for precision, recall and F1 values for all the models of summarization experimented in this project

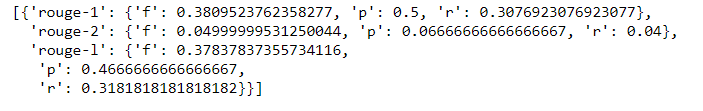
****

Figure 1: ROUGE score of summary column 1 vs 2 in Opinosis dataset

* 1. **Opinosis dataset:**

This dataset contains opinions collected from users on 51 topics with each data. The dataset comes with gold summarization that has been used to calculate the ROUGE scores for our models. The following are the columns found in this dataset.

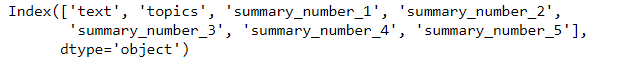
****

Figure 2: Columns found in Opinosis dataset

* 1. **Implementation details**

The below section shows the implementation of these models in details with their evaluation using rouge scores:

* + 1. **Word frequency model**

The first model tried was the simple bag of words module to create a benchmark to check the performance of deep learning models. The following are the steps involved in its creation:

1. compute word frequency
2. score each sentence according to word frequency (can be weighted)
3. generate threshold of sentence selection (average score, etc.)
4. Selected sentence (score > threshold) as summary

The figure below shows the ROUGE score of summarizations done with word frequency model against gold summaries of the Opinosis dataset:

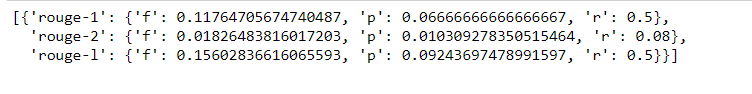
****

Figure 3: ROUGE scores of Word Frequency model

* + 1. **Text rank model**

Text rank is a graph based ranking algorithm for NLP. Graph-based ranking algorithms work by deciding the importance of a vertex within a graph, based on global information recursively drawn from the entire graph. To implement the text rank module, we have used the python library *Gensim* designed to automatically extract summaries from texts. The following are the steps involved in the generation of summarization using the Text Rank model:

1. Cleaning Text (remove punctuation, Stopwords, stemming)
2. Generate Vector representation of sentences
3. Use cosine similarity find similarity of sentences
4. Apply PageRank algorithm
5. Extract top N sentences as summary

The figure below shows the ROUGE score of gensim text rank model:

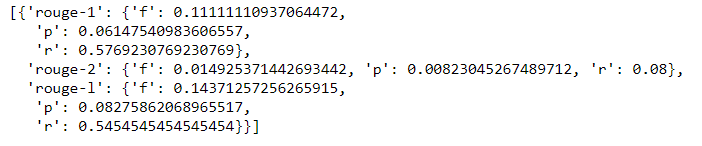


Figure 4: ROUGE scores of text rank model

* + 1. **BERT extractive summarizer**

In this module we directly tried the pretrained implementation of the HuggingFace BERT transformer, to benchmark its performance and select the best model for the project. More information about the BERT model and implementation can be found under the model and methods section.

The basic steps involved are as shown below:

1. Tokenize paragraph into sentences
2. Format each sentence as Bert input format, and Use Bert tokenizer to tokenize each sentence into words
3. Call Bert pretrained model, conduct word embedding, obtain embedded word vector for each sentence
4. Apply pooling strategy to obtain sentence embedding from word embedding
5. Obtain sentence vector for each sentence in the paragraph, apply K-means algorithm to cluster similar sentence
6. Return the closest sentence to each centroid (euclidean distance) as the summary, ordered by appearance

The following are the ROUGE scores of the BERT summarizer:

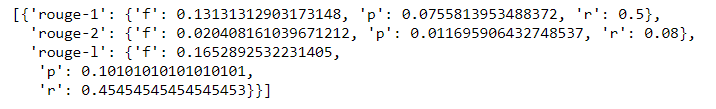


Figure 5: ROUGE scores of BERT extractive summarizer

* + 1. **T5 transformer model**

T5 is a new transformer model from Google that is trained in an end-to-end manner with text as input and modified text as output. It achieves state-of-the-art results on multiple NLP tasks like summarization, question answering, machine translation etc using a text-to-text transformer trained on a large text corpus. T5 is an *abstractive summarization algorithm*.

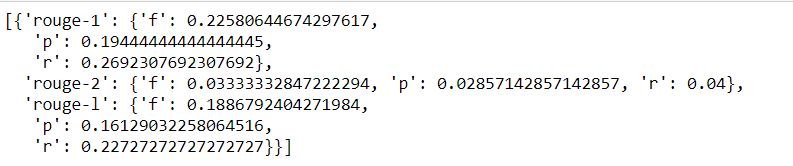
****

Figure 6: ROUGE scores of T5 model

To compare the performance of the above models side by side, we plot the graphs shown below. The models were run against all texts of the Opinosis dataset, and ROUGE scores were calculated for each of them. The first graph shows the mean of ROUGE-1 F1 score values for each of the models. We can see that the T5 model performs better than all the other models with maximum mean value.

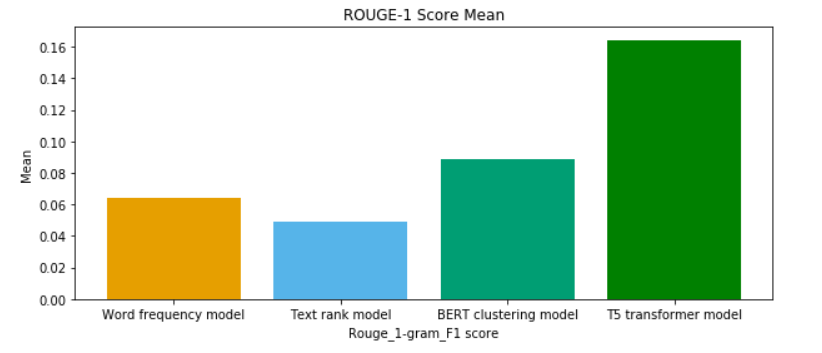


Figure 7: ROUGE mean values for all models visualized

In the second plot, the density distribution of ROUGE-1 F1 score for these models.

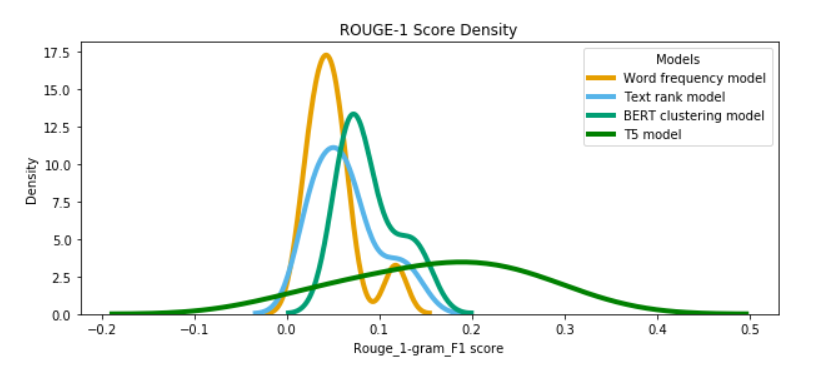


Figure 8: Density distribution of the models

From the above experiments on both extractive and abstractive models, we conclude that the *T5 transformer model from huggingFace performs better than all the other models for text summarization problem.*

* + 1. **Fine-tuning of the T5 model:**

The T5 transformer model was then fine-tuned on the custom dataset - News Summary dataset obtained from Kaggle:

1. The *CustomDataset* class is used to create 2 datasets, for training and for validation.
2. *Training Dataset* is used to fine tune the model: **80% of the original data**
3. *Validation Dataset* is used to evaluate the performance of the model. The model has not seen this data during training.

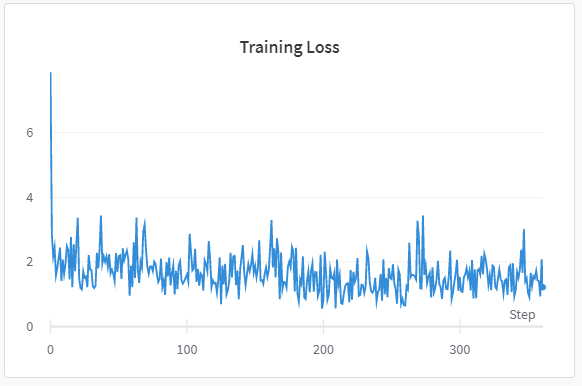


Figure 9: Training loss plotted for fine-tuning of T5 model

# Model

* 1. **BERT model**

*Bidirectional Encoder Representations from Transformers (BERT)* is a Transformer-based machine learning technique for natural language processing (NLP) pre-training developed by Google. The original English-language BERT model comes with two pre-trained general types: the BERT base model with 12-layer, 768-hidden, 12-heads, 110M parameter neural network architecture. BERT Large model, a 24-layer, 1024-hidden, 16-heads, 340M parameter neural network architecture. These models are trained on the BooksCorpus with 800M words, and a version of the English Wikipedia with 2,500M words. BERT makes use of Transformer, an attention mechanism that learns contextual relations between words (or sub-words) in a text. Since BERT’s goal is to generate a language model, only the encoder mechanism is necessary. Transformer encoder reads the entire sequence of words at once. Therefore, it is considered bidirectional. This characteristic allows the model to learn the context of a word based on all its surroundings (left and right of the word). The input is a sequence of tokens, which are first embedded into vectors and then processed in the neural network. The output is a sequence of vectors of size H, in which each vector corresponds to an input token with the same index.

* 1. **Masked LM (MLM)**

Before feeding word sequences into BERT, 15% of the words in each sequence are replaced with a [MASK] token. The model then attempts to predict the original value of the masked words, based on the context provided by the other, non-masked, words in the sequence.

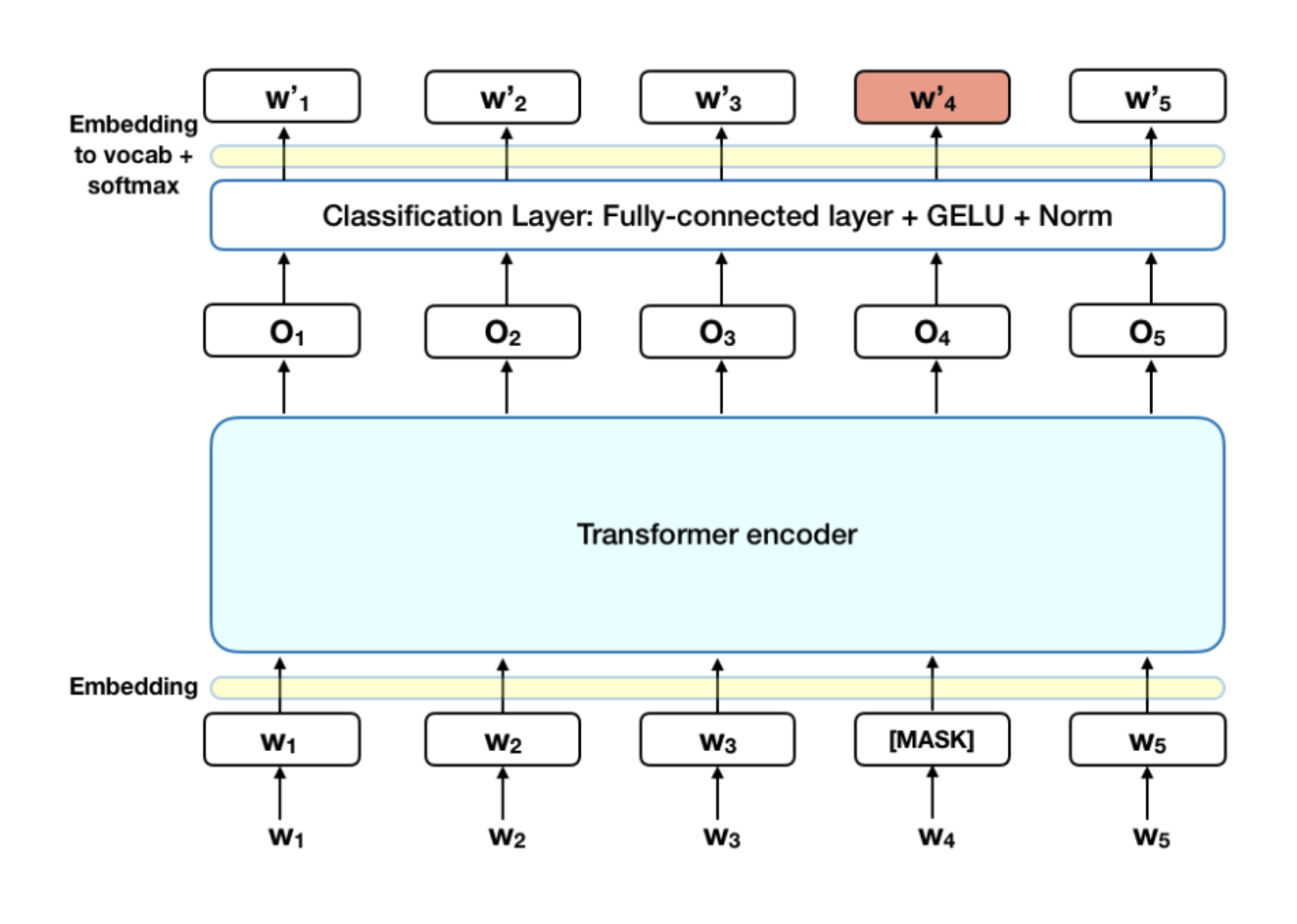


Figure 10: Masked Language Model

* 1. **Next Sentence Prediction (NSP)**

During training, 50% of the inputs are a pair in which the second sentence is the subsequent sentence in the original document, while in the other 50% a random sentence from the corpus is chosen as the second sentence. The assumption is that the random sentence will be disconnected from the first sentence.

To help the model distinguish between the two sentences in training, the input is processed in the following way before entering the model:

1. A [CLS] token is inserted at the beginning of the first sentence and a [SEP] token is inserted at the end of each sentence.
2. A sentence embedding indicating Sentence A or Sentence B is added to each token. Sentence embeddings are similar in concept to token embeddings with a vocabulary of 2.
3. A positional embedding is added to each token to indicate its position in the sequence. The concept and implementation of positional embedding are presented in the Transformer paper.

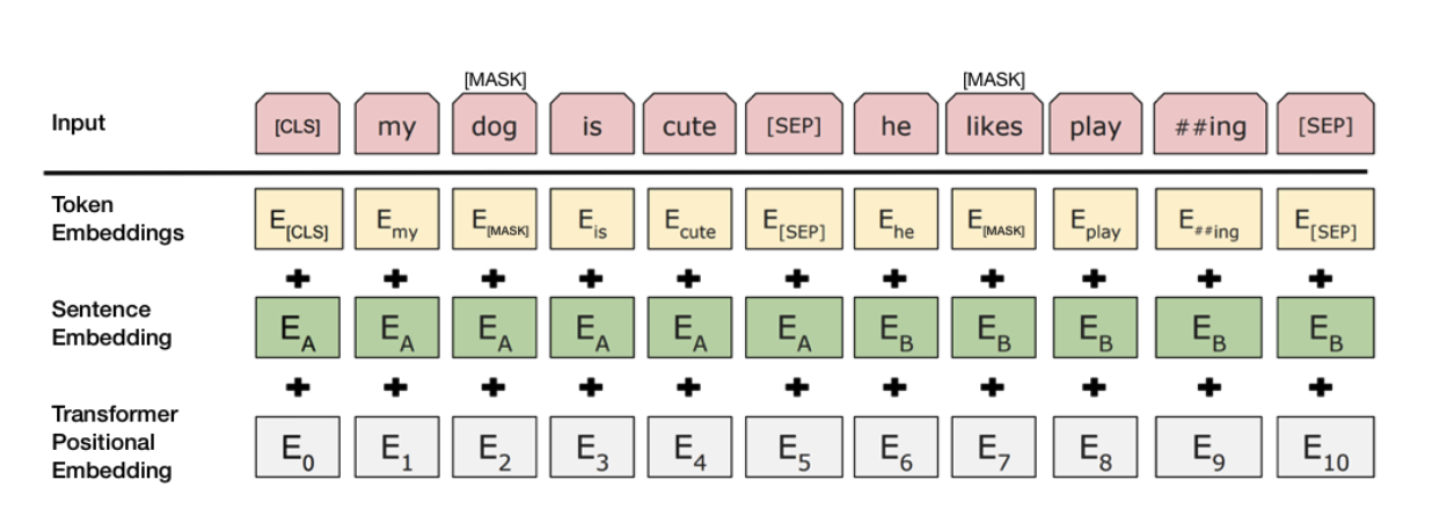


Figure 11: Inputs to BERT model

* 1. **Fine-tuned BERT Summarised model** 
     1. **Input**

In order to represent individual sentences, we insert external [CLS] tokens at the start of each sentence, and each [CLS] symbol collects features for the sentence preceding it.Interval segment embedding are used to distinguish multiple sentences within a document. For segment embedding we assign EA or EB depending on whether the sentence is odd or even. For example, for document [sent1, sent2, sent3, sent4, sent5], would assign embedding [EA, EB, EA, EB, EA]. Document representations are learned hierarchically where lower Transformer layers represent adjacent sentences, while higher layers, in combination with self-attention, represent multi-sentence discourse.

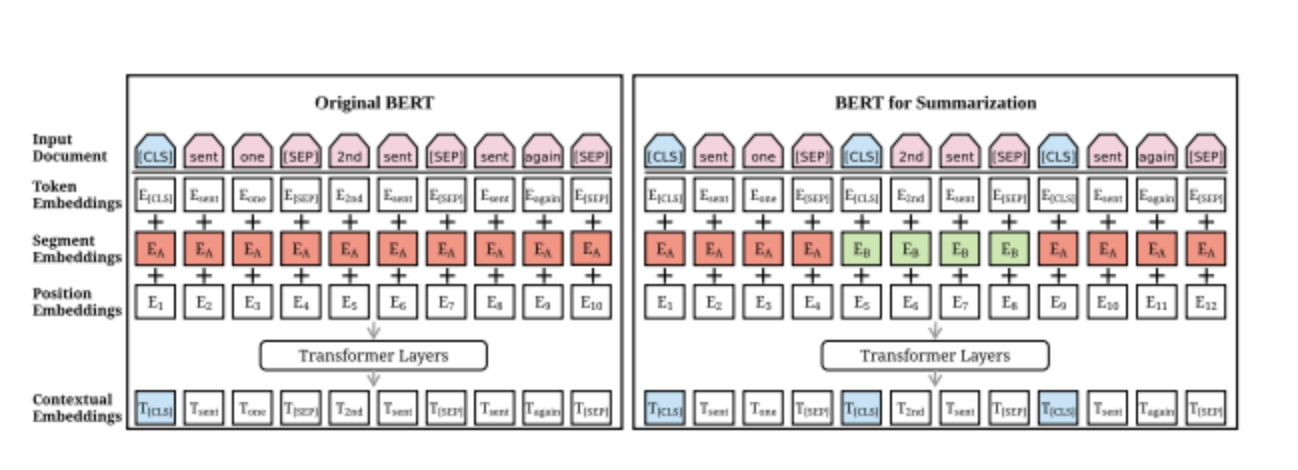


Figure 12: Fine-tuned BERT summarized model

* + 1. **Output**

BertSum outputs a list of scores shows the representativeness of sentence toward documents. For documents [sent1, sent2, sent3, sent4, sent5] has scores [score1, score2, score3, score4, score5]. The higher score is, the more representative the sentence is.

* + 1. **Summarization**

The output of Bert was then fed into Summarization Layers for summarization. In paper, the author tested numbers of summarization layers’ structure, and in published GitHub it’s still selectable. There are RNN based, Classifier based, and Transformer based as options. Basically, it just stacks numbers of base layers and ends up with scores.

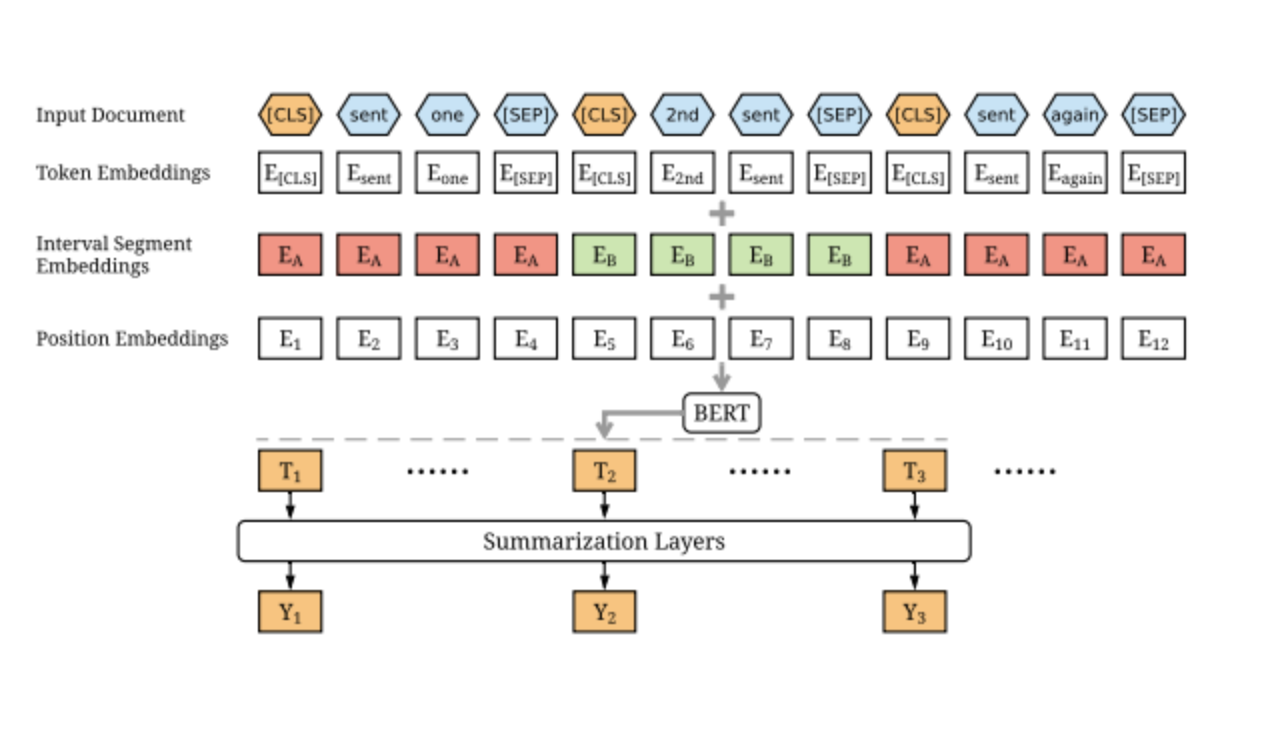


Figure 13: BERT summarization

# Data

The project is based on summarizing text content. The input to the application is an URL link of a website. The application will extract the textual content in the link. Pre-processing is performed on the textual content applying the model on it. NLTK library is used to perform natural language processing on the input text. The input text is tokenized. In case the length of the sentence is less than 75 or if the sentence starts with conjunctions, those sentences are emitted. The application will then perform extractive summarization on the textual content using the fine-tuned BERT model and summarize it. The fine-tuned BERT model is trained on CNN news articles having approximately 90k documents and 380k questions. It is also trained on news articles from Daily Mail having approximately 197k documents and 879k questions. The application will also generate word clouds for the textual content. The output which is the summarized text is shown in a web application.

# Methods

This project utilizes the BERT model to perform extractive text summarization on various text excerpts like new articles or any kind of raw text. BERT performs better than other NLP algorithms on sentence embedding. BERT builds on top of the transformer architecture, but its objectives are specific for pre-training. On one step, it randomly masks out 10% to 15% of the words in the training data, attempting to predict the masked words, and the other step takes in an input sentence and a candidate sentence, predicting whether the candidate sentence properly follows the input sentence. It comprises two services: text service and summarization service. The text service is used to create, store and retrieve articles whereas summarization service creates and stores summary of given articles.

To create summary from saved articles, summarization service tokenizes the input into clean sentences and pass it to BERT model for inference to output embeddings. K-means is used to cluster these embeddings. the sentences which are closest to the centroid are chosen as the appropriate summary sentences.

* 1. **Architecture**

The project architecture is shown in the figure below. We have used docker to containerize our application. It exposes a Rest API to manage articles and generate summary. This Rest API can be consumed directly or through a web application. There are two services: text and summarization. All the text will be processed and stored to the database. We also have a BERT summarization model which will interact with the database to generate summary and display visualizations. The text service is used to manage articles and store raw articles to a database. Summarization service helps to generate and store the summarization in a database which can be fetched through APIs.

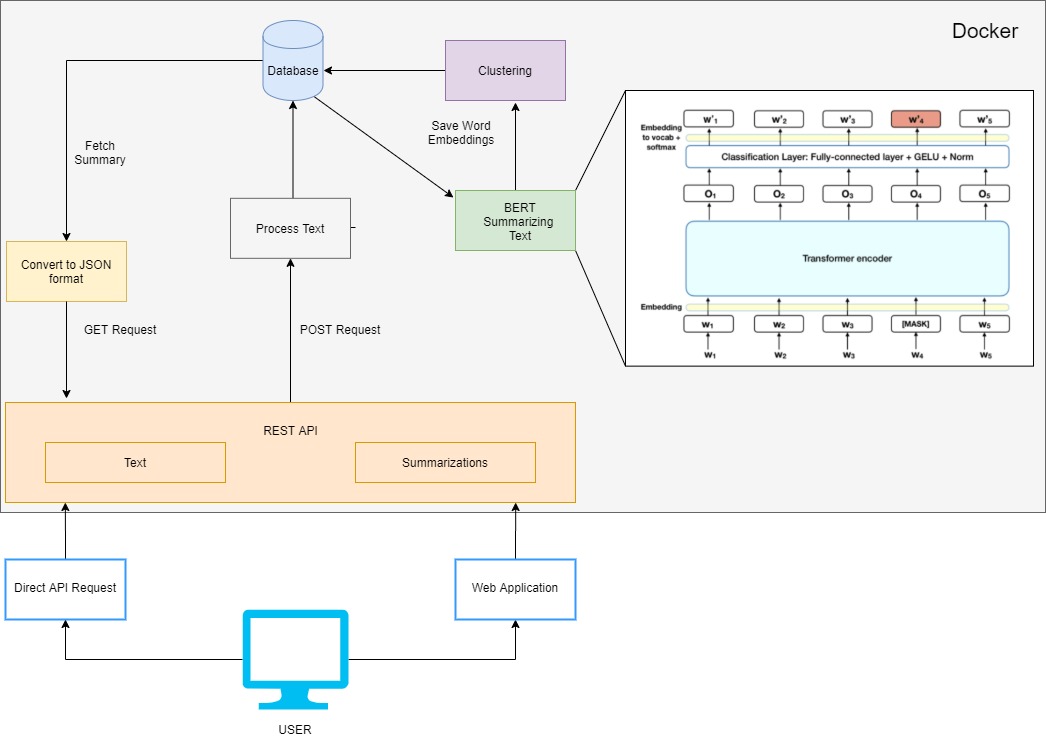


Figure 14: Architecture diagram of content summarization application

* 1. **RESTful API**

The contents of this project include a RESTful API to serve these summaries. The list of APIs are as follows:

|  |  |  |  |
| --- | --- | --- | --- |
| **API Path** | **Action** | **Description** | **Parameters** |
| /articles | POST | The user can upload article content either in the form of raw text or from any url, which can then be utilized for summarization. | article, content, name |
| /articles | GET | Get all uploaded articles | article name |
| /articles/{id} | GET | Retrieve an article | NA |
| /articles/{id}/summaries | POST/GET | Create a summarization or GET all summarizations for a given article. | name, compression ratio, custom tag |
| /articles/url | POST | Scrape and create an article from given URL | article, URL, name |
| /articles/{id}/summaries/{summarization\_id} | GET/DELETE | Get or delete a summarized article. | NA |
| /articles/eda/{id} | GET | Generate word cloud for a given article | NA |

* 1. **Client**

The front-end application is designed using ReactJS. Figure 15 shows the application, where the summary is generated using an article which is extracted from a URL. We can directly provide an URL for any article and generate a summary. It also highlights and displays the keywords in the article. Word cloud adds to the visualization clarity.

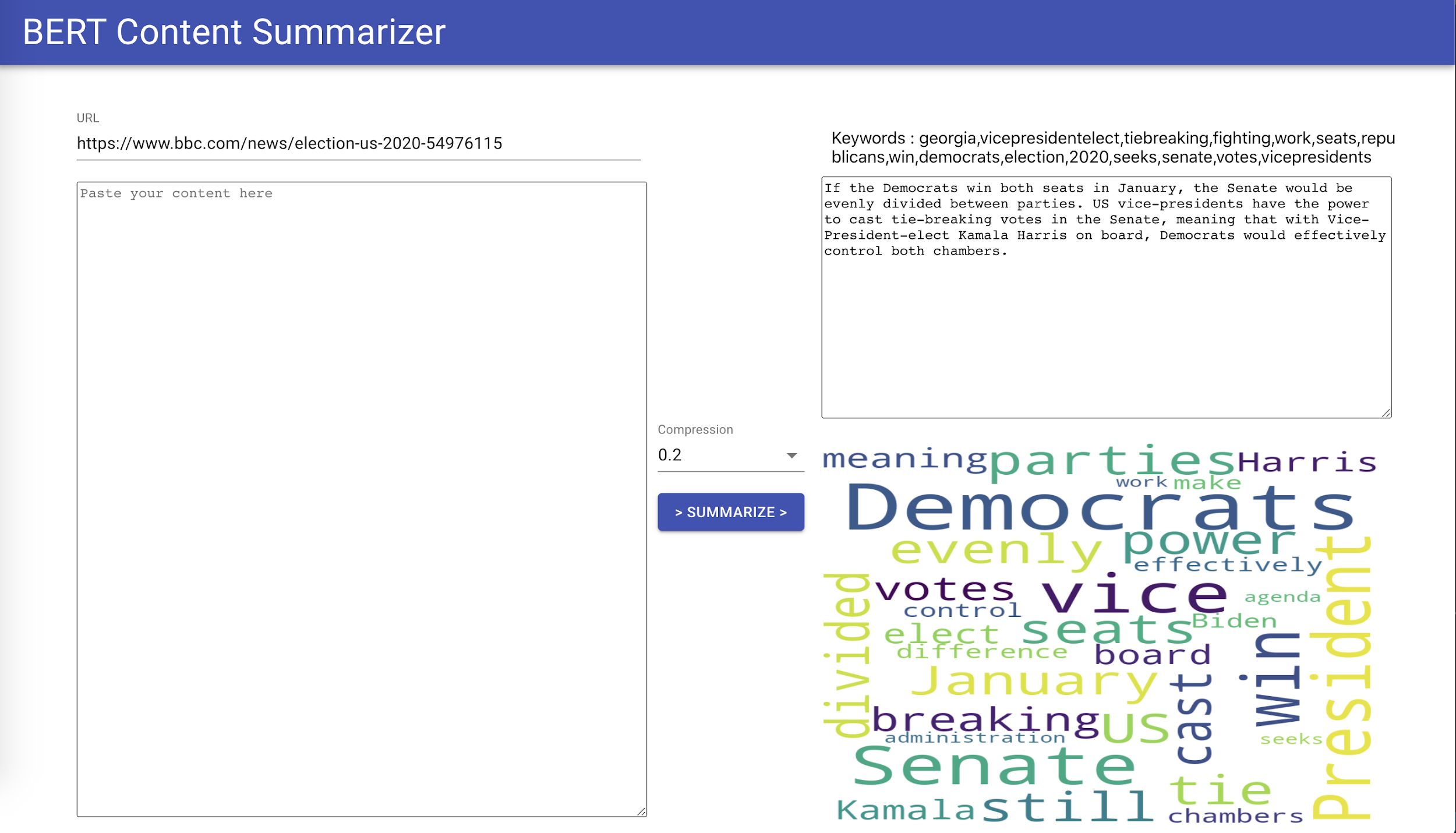


Figure 15: UI view of Content summarization using BERT application

We can also generate the summary by providing raw text in the text area on the left side:

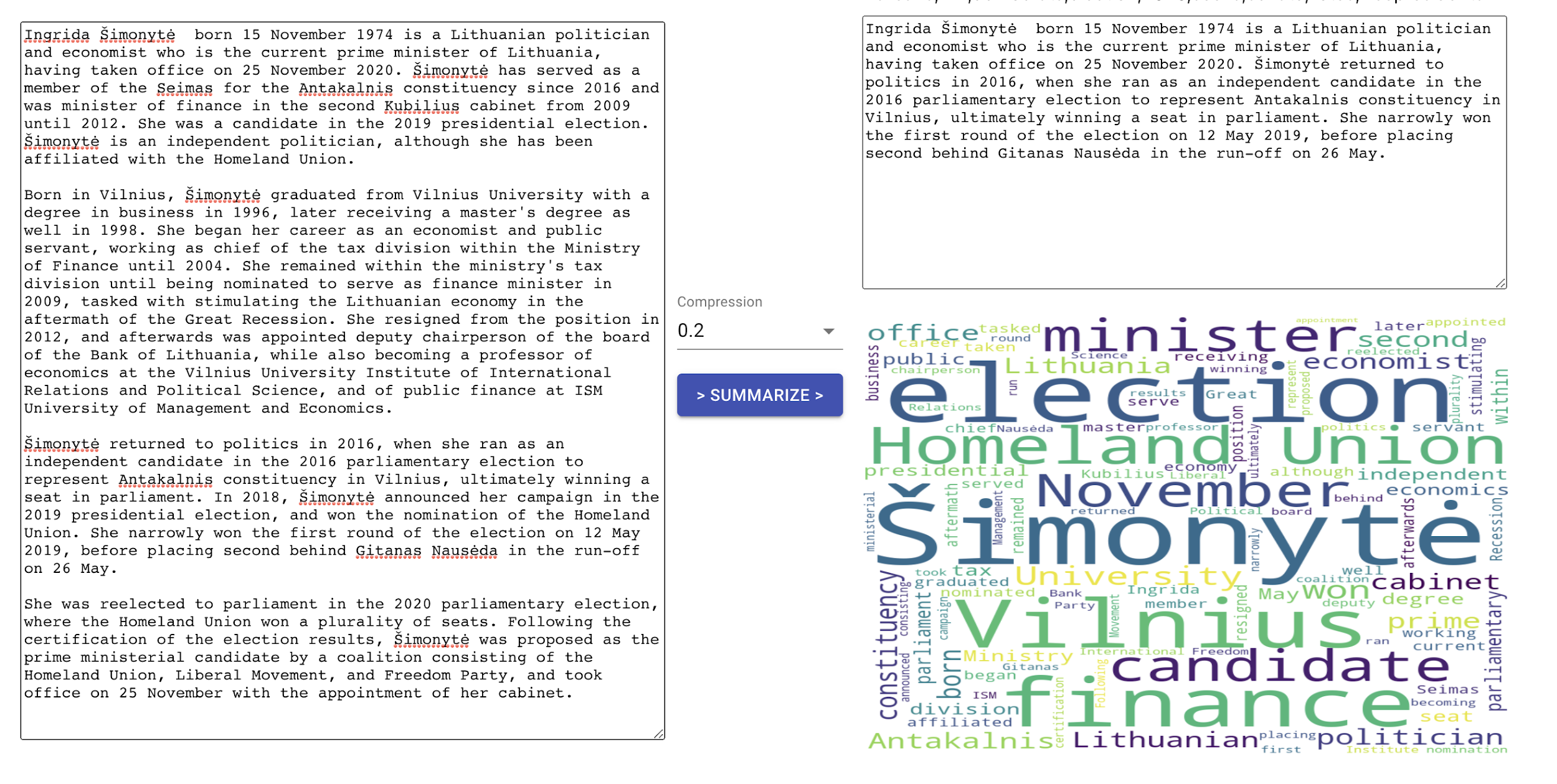


Figure 16: UI view showing summary and word cloud generated

* 1. **Workflows**

1. Steps to add articles in the application

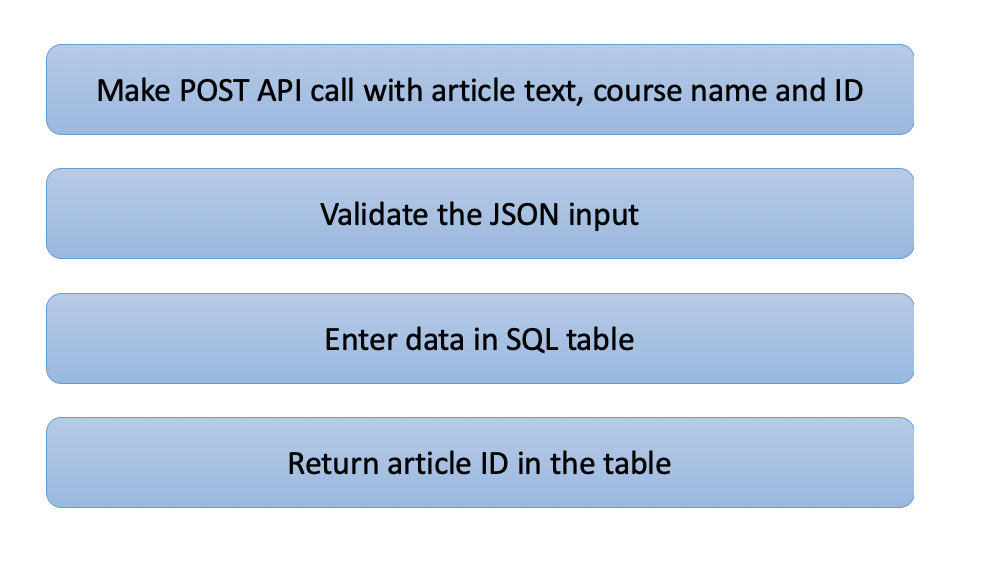
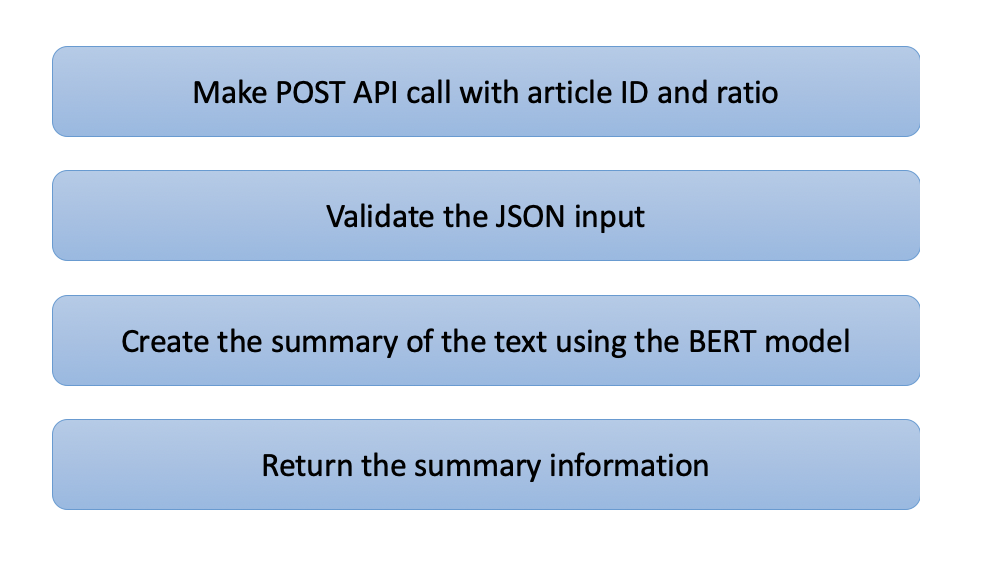


Figure 17: Flow of adding articles

1. Steps to create article summary

Figure 18: Flow of creating summaries

1. Steps to get summary of articles

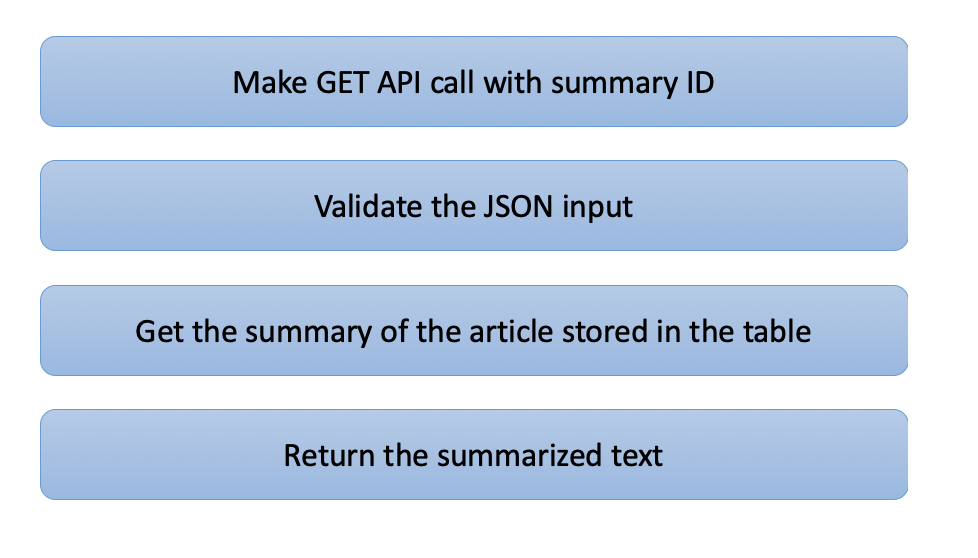


Figure 19: Flow of retrieving summaries

# References

1. Miller, D., “Leveraging BERT for Extractive Text Summarization on Lectures”, *arXiv e-prints*, 2019. <https://arxiv.org/abs/1906.04165>
2. <https://kavita-ganesan.com/opinosis-opinion-dataset/#.X9FQaNhKhPY>
3. <http://www.ccs.neu.edu/home/vip/teach/DMcourse/5_topicmodel_summ/notes_slides/What-is-ROUGE.pdf>
4. <https://medium.com/lsc-psd/a-bert-based-summarization-model-bertsum-88b1fc1b3177>
5. <https://huggingface.co/transformers/model_doc/t5.html>
6. <https://minimaxir.com/2019/09/howto-gpt2/>