

# 1.INTRODUCTION

# INTRODUCTION

## 1.1 INTRODUCTION

We are current students of 4th year cse. As part of Mini Project-II. As we all have a common interest in Natural Language Processing (NLP) we have chosen that. YouTube Video Transcript Summarization Using NLP is one of the top applications of NLP.

This project is all about YouTube Video Transcript Summarization Using NLP and the featured NLP summarization techniques and models.

### **Why Natural Language Processing?**

Natural Language Processing(NLP) is one of the fastest-growing tech fields right now. From message spam filters to medical diagnosis with a chatbot, NLP is everywhere. Some of the hot use cases of NLP right now are text summarization, chatbot, machine translation, text generation, etc.

NLP is the ability of a computer program to understand the human language as it is spoken or written to be referred to as Natural Language.

### **Why Transcript Summarization?**

YouTube is a video sharing platform, the second most visited website, the second most used search engine, and is stronger than ever after more than 17 years of being online. YouTube uploads about 720,000 hours of fresh video content per day. The number of videos available on the web platform is steadily growing. It has become increasingly easy to watch videos on YouTube for anything, from cooking videos to dance videos to motivational videos and other bizarre stuff as well. The content is available worldwide primarily for educational purposes. The biggest challenge while extracting information from a video is that the viewer has to watch the entire video to understand the context, unlike images, where data can be gathered from a single frame. If a viewer has low network speed or any other device limitation can lead to watching video with a low resolution that makes it blurry and hectic to watch. Also, in-between advertisements are too frustrating. So, removing the junk at the start and end of the concerned video as well as skipping advertisements, and getting its summary to directly jump to your part of interest is valuable and time efficient. This project focuses on reducing the length of the script

for the videos. Summarizing transcripts of such videos automatically allows one to quickly lookout for the important patterns in the video and helps to save time and effort to go through the whole content of the video. The most important part of this project will be its ability to string together all the necessary information and concentrate it into a small paragraph. Video summarization is the process of identifying the significant segments of the video and produce the output video whose content represents the entire input video. It has advantages like reducing the storage space used for the video. This project will give an opportunity to have hands-on experience with state-of-the-art NLP technique for abstractive text summarization and implement an interesting idea suitable for intermediates and a refreshing hobby project for professionals.

### **Our Expectations?**

We expect to get good mentorship on this project along with proper guidance wherever necessary. We also expect to get help from mentors in certain situations where we may get stuck along with getting continuous feedback and suggestions for improvement on my work in terms of efficiency and productivity. This would help us to give our best in this project.

### **Learning Expectations?**

We expect to get outstanding work experience in this project. The guidance from an expert and experienced mentor would not only help us grow as a professional, but would also be beneficial in acquiring insights and perspectives on the domain of skills required for this idea. Along with all this, we are also expecting to take our knowledge to another level of efficiency through this idea by getting experience on problem solving, feature adding, clean coding etc.

## **1.2 MOTIVATION**

This project will provide a great opportunity to work with an amazing group of people and being a learner and learning new things every day will make us technically strong. Moreover, we will get the opportunity to make connections with a group of great minded people which would be really crucial in beginning our career and later on in life.

## **1.3 PROBLEM DEFINITION**

Video transcript summarizer has got a lot of scope in today's world. It highlights the important topics from the video. People spend a noticeable amount of time binge watching YouTube videos, be it for entertainment, education purposes, or getting some important information or exploring their interests. If you wish to find a video to get any important

information about a topic, it is a very difficult task to achieve as most of the videos are filled with insignificant buffer material. In most of the cases, the overall intent is to obtain some form of quality information from the video. This project brings forward a video summarization system based on Natural Language Processing and Machine Learning to generalize YouTube video transcripts for abstractive text summarization without losing the main elements and content. This project focuses on reducing the length of the script for the videos.

## **1.4 OBJECTIVE OF THE PROJECT**

Video transcript summarizer has got a lot of scope in today's world. It highlights the important topics from the video. People spend a noticeable amount of time binge watching YouTube videos, be it for entertainment, education purposes, or getting some important information or exploring their interests. If you wish to find a video to get any important information about a topic, it is a very difficult task to achieve as most of the videos are filled with insignificant buffer material. In most of the cases, the overall intent is to obtain some form of quality information from the video. This project brings forward a video summarization system based on Natural Language Processing and Machine Learning to generalize YouTube video transcripts for abstractive text summarization without losing the main elements and content. This project focuses on reducing the length of the script for the videos.

## **1.5 LIMITATIONS OF THE PROJECT**

The performance of the proposed project suffers in terms of response time. In addition, it also applies to data in real time.

## **1.6 ORGANISATION OF THE REPORT**

Report includes the stepwise implementation of the working application generated and it describes the overview of how effectively the project can be implemented and allows the viewer of the report to get an overview of the project.

- 1.6.1 Chapter two includes project software and hardware specifications.
- 1.6.2 Chapter three includes what is the proposed system.
- 1.6.3 Chapter four includes module organization and feasibility study.

- 1.6.4 Chapter five includes flow diagrams of Proposed approach.
- 1.6.5 Chapter six includes source code and output screens.
- 1.6.6 Chapter seven includes a conclusion.
- 1.6.7 Chapter eight includes Reference.

## **2.SYSTEM SPECIFICATIONS**

## **2.SYSTEM SPECIFICATIONS**

### **2.1 SOFTWARE SPECIFICATIONS**

- Operating system : Windows 7/10
- Coding Language : Python.
- Platform : Google Colabs, studios.
- Dataset : Kaggle data set.
- Modules : NumPy, Natural Language Toolkit (NLTK), re and Sklearn libraries

### **2.2 HARDWARE SPECIFICATIONS**

- Processor : Intel core i3/ i5
- Hard Disk : 64GB/128GB
- Monitor : 15 VGA Color.
- Mouse : Logitech.
- RAM : 4 GB & more.

## **3. LITERATURE SURVEY**



### 3.LITERATURE SURVEY

#### 3.1 INTRODUCTION

The project that we would like to work on for Mini Project-II is YouTube video transcript summarization using NLP.

Natural Language Processing(NLP) is one of the fastest-growing tech fields right now. From message spam filters to medical diagnosis with a chatbot, NLP is everywhere. Some of the hot use cases of NLP right now are text summarization, chatbot, machine translation, text generation, etc. Have you ever imagined getting a short summary of a big YouTube tutorial or video for quick reading before watching the video, definitely this will help you to save a lot of your time by getting a quick understanding or summarization about the video in a short time? This project is about discussing a mini-NLP project, a YouTube Transcript Summarizer which will summarize the content of the YouTube video. For many videos, the main content of the videos is only 50-60% of the total length, so the YouTube summarizer will summarize the content of the video by keeping all the important points and making it short and easily understandable. This will be useful in getting the summary of several lecture videos easily.

Performing text summarization: This task consists of shortening a large form of text into a precise summary that keeps all the necessary information intact and preserves the overall meaning. For this purpose, in NLP for text summarization, there are two types of methods used: Extractive Summarization: In this type of text summarization, the output is only the important phrases and sentences that the model identifies from the original text. For the purpose of extractive summarization, we have used the TF-IDF model with Text Rank Algorithm. Term Frequency - Inverse Document Frequency(TF-IDF) After the cleaning process, we have to convert the words into its vectorized form so that our algorithm will process it by using TF-IDF. This is a technique to measure the quantity of a word in documents, we compute a weight to each word which signifies the importance of the word in the document and corpus. TF (Term Frequency): TF calculates the frequency of a word in a document.  $TF = \text{No. of repetition of the word in the sentence} / \text{No. of words in a sentence}$  IDF (Inverse Document Frequency): IDF is the inverse of the document frequency which measures the informativeness of term  $t$ .  $IDF = \log(\text{No. of sentences} / \text{No. of sentences containing words})$  After this, we will multiply both matrices to obtain the vectorized form which tells us which words are the most important.

### 3.2 EXISTING SYSTEM

#### 3.2.1. Youtube Video Transcript Summarization using NLP

Existing system is built through Term Frequency-Inverse Document Frequency(TF-IDF) is a vectorizer that converts the text into a vector.

Term frequency-inverse document frequency is a text vectorizer that transforms the text into a usable vector. It combines 2 concepts, Term Frequency (TF) and Document Frequency (DF). The term frequency is the number of occurrences of a specific term in a document. Term frequency indicates how important a specific term is in a document. Term frequency represents every text from the data as a matrix whose rows are the number of documents and columns are the number of distinct terms throughout all documents. Document frequency is the number of documents containing a specific term. Document frequency indicates how common the term is. Inverse document frequency (IDF) is the weight of a term, it aims to reduce the weight of a term if the term's occurrences are scattered throughout all the documents. IDF can be calculated as

follow:  $idf_i = \log(\frac{n}{df_i})$  Where  $idf_i$  is the IDF score for term  $i$ ,  $df_i$  is the number of documents containing term  $i$ , and  $n$  is the total number of documents. The higher the DF of a term, the lower the IDF for the term. When the number of DF is equal to  $n$  which means that the term appears in all documents, the IDF will be zero, since  $\log(1)$  is zero, when in doubt just put this term in the stopwords list because it doesn't provide much information. The TF-IDF

$w_{i,j} = tf_{i,j} \times idf_i$  score as the name suggests is just a multiplication of the term frequency matrix with its IDF, it can be calculated as follow:

Where  $w_{ij}$  is TF-IDF score for term  $i$  in document  $j$ ,  $tf_{ij}$  is term frequency for term  $i$  in document  $j$ , and  $idf_i$  is IDF score for term  $i$ .

### 3.3 DISADVANTAGES OF EXISTING SYSTEM

First, there is what I call the “zero value issue”, which stems from the inverse document frequency calculation. By construction, if the word of interest appears in all documents, the tf-idf value will be zero; the ratio of the number of documents in the analysis to the number of

documents the word appears in will be one, and the natural log of one is zero. This would suggest that the word is not uniquely important to any given document.

In many instances this may be reasonable: the idea that if a word appears everywhere it is not unique anywhere seems like a solid heuristic. However, there are cases where this causes meaningful challenges for an analysis and where, in my opinion, the assigned importance is incorrectly low.

To see how this can be problematic, consider the following scenario. Imagine we have three documents: A, B, and C. Assume we want to get a tf-idf value for the word “apple” in document A, and imagine that apple makes up a full 75% of words in that document. If apple appears just once in document B and not at all in document C, the tf-idf value would be rightfully high. However, if apple appeared just once in *both* document B *and* C, the tf-idf would fall all the way to *zero*. Despite the fact that the only change is that apple now appears once more outside of document A, and despite document A being almost completely made up of the word apple, the tf-idf statistic plummets from something relatively high to nothing at all. This is an undesirable feature of the measure both because it suggests that the two cases are far more different than they are and because it suggests that apple is not uniquely important to A even though it clearly is.

The second drawback, similar to the first, is what I call the “extensive margin issue.” This arises because the inverse document frequency portion of tf-idf does not take into account how *often* a word appears in other documents (the intensive margin), just whether it appears at all (the extensive margin). This causes it to be overly sensitive when there are changes on the extensive margin and overly resistant to change when there are changes on the intensive margin, even though the latter is arguably more important.

To see this in action, consider the same setup from above, this time with apple making up 10% of words in document A. If apple did not appear at all in document B, the tf-idf value would be relatively high. However, if it were to appear just a single time in B, the value would plummet dramatically (though not quite to zero so long as it doesn’t appear in C). Despite the two scenarios being almost identical, with the only change being one more outside appearance of apple, the value changes dramatically, all because that change came on the extensive margin (i.e. apple went from being in no outside documents to one). Of course, in practical terms there is

little difference between the word apple appearing just once or not at all outside of document A. Yet tf-idf would suggest that the two cases are radically different.

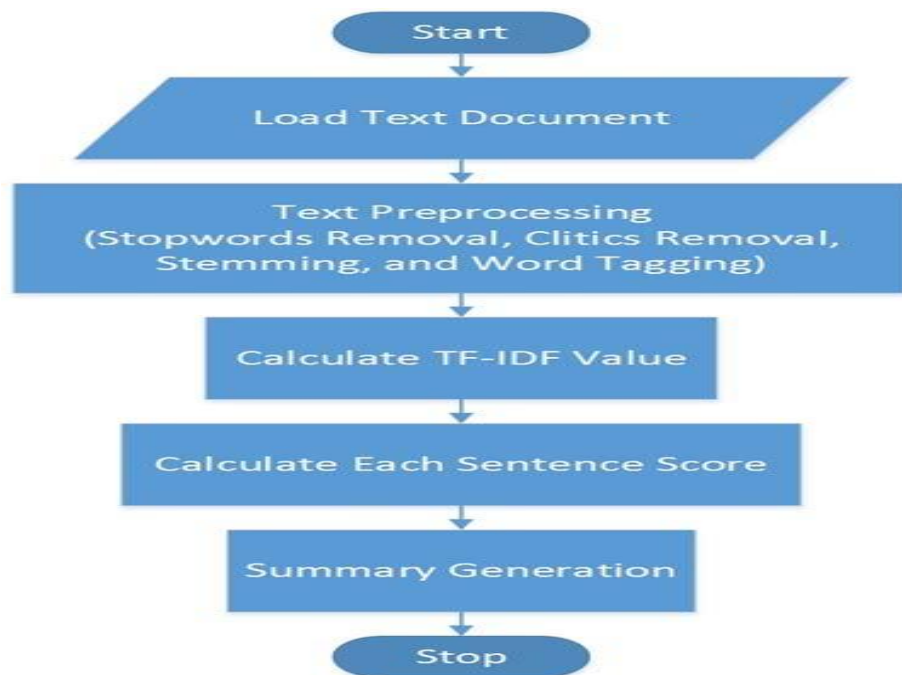


Fig. 3.3.1: TF-IDF working flowchart

On the flip side, the tf-idf would not change at all if apple were to go from appearing just once in document B to making up literally 100% of the document's words. That is, it doesn't matter *how many* times the word apple appears in document B, because tf-idf ignores the intensive margin. It only matters whether it appears at all. However, if our goal is to get a measure that indicates relative importance, it is quite critical to distinguish between these scenarios. In the former, apple is relatively important to document A compared to B (only showing up once in B), while in the latter apple is not at all uniquely important for document A (at 10% of its words) relative to how important it is for document B (100% of its words). Yet tf-idf sees no difference.

To recap: the zero value issue means that tf-idf will suddenly plummet to zero if the word of interest appears in all of the outside documents, even if it only appears once in each of them. The extensive margin issue means that tf-idf will differ dramatically when there are minor and relatively unimportant changes on the extensive margin, but will not differ at all when there are changes, even substantial ones, on the intensive margin.

### 3.4. PROPOSED SYSTEM

Bidirectional and Auto-Regressive Transformer(BART) is a transformer that is now commonly used for sequence-to-sequence problems. Its architecture mainly consists of a Bidirectional encoder and a left-to-right decoder. BART is suitable for summarization, machine translation, question-answering, etc.

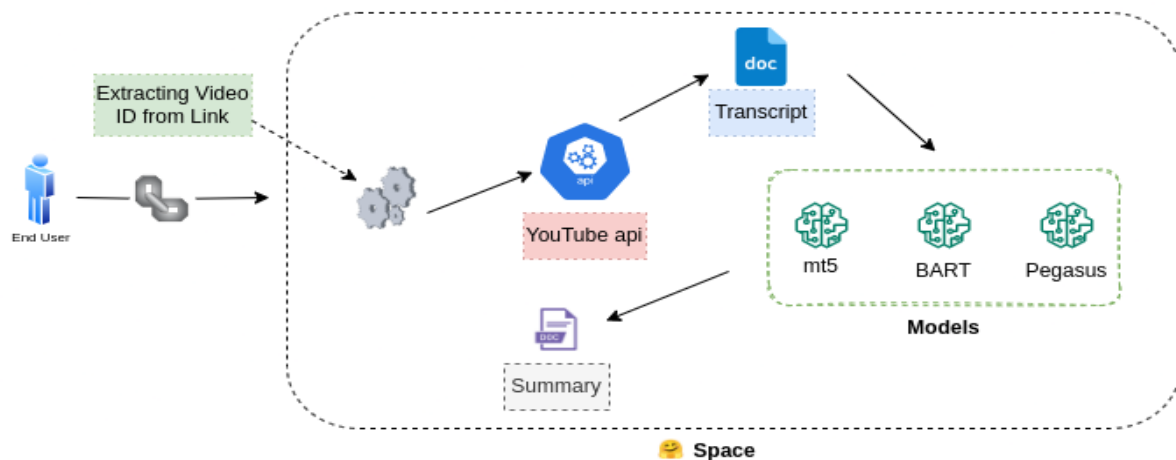


Fig. 3.4.1: BART Model

We present BART, a denoising autoencoder for pretraining sequence-to-sequence models. BART is trained by (1) corrupting text with an arbitrary noising function, and (2) learning a model to reconstruct the original text. It uses a standard Transformer-based neural machine translation architecture which, despite its simplicity, can be seen as generalizing BERT (due to the bidirectional encoder), GPT (with the left-to-right decoder), and many other more recent pre-training schemes. We evaluate a number of noising approaches, finding the best performance by both randomly shuffling the order of the original sentences and using a novel in-filling scheme, where spans of text are replaced with a single mask token. BART is particularly effective when fine-tuned for text generation but also works well for comprehension tasks. It matches the performance of RoBERTa with comparable training resources on GLUE and SQuAD, achieves new state-of-the-art results on a range of abstractive dialogue, question answering, and summarization tasks, with gains of up to 6 ROUGE. BART also provides a 1.1

BLEU increase over a back-translation system for machine translation, with only target language pretraining. We also report ablation experiments that replicate other pre training schemes within the BART framework, to better measure which factors most influence end-task performance.

The [original Transformer](#) is based on an encoder-decoder architecture and is a classic sequence-to-sequence model. The model's input and output are in the form of a sequence (text), and the encoder learns a high-dimensional representation of the input, which is then mapped to the output by the decoder. This architecture introduced a new form of learning for language-related tasks and, thus, the models spawned from it achieve outstanding results overtaking the existing deep [neural network-based methods](#).

Since the inception of the vanilla Transformer, several recent models inspired by the Transformer used the architecture to improve the benchmark of [NLP tasks](#). Transformer models are first pre-trained on a large text corpus (such as Book Corpus or Wikipedia). This pretraining makes sure that the model “understands language” and has a decent starting point to learn how to perform further tasks. Hence, after this step, we only have a language model. The ability of the model to understand language is highly significant since it will determine how well you can further train the model for something like text classification or text summarization.

BART is one such Transformer model that takes components from other Transformer models and improves the pretraining learning. BART or Bidirectional and Auto-Regressive

BART was trained as a denoising autoencoder, so the training data includes “corrupted” or “noisy” text, which would be mapped to clean or original text. The training format is similar to the training of any denoising autoencoder. Just how in [computer vision](#), we train autoencoders to remove noise or improve the quality of an image by having noisy images in the training data mapped with clean, original images as the target.

So, what exactly counts as noise for text data? The authors of BART settle on using some existing and some new noising techniques for pretraining. The noising schemes they use are Token Masking, Token Deletion, Text Infilling, Sentence Permutation, and Document Rotation. Looking into each of these transformations:

- Token Masking: Random tokens in a sentence are replaced with [MASK]. The model learns how to predict the single token based on the rest of the sequence.
- Token Deletion: Random tokens are deleted. The model must learn to predict the token content and find the position where the token was deleted from.
- Text Infilling: A fixed number of contiguous tokens are deleted and replaced with a single [MASK] token. The model must learn the content of the missing tokens and the number of tokens.
- Sentence Permutation: Sentences (separated by full stops) are permuted randomly. This helps the model to learn the logical entailment of sentences.
- Document Rotation: The document is rearranged to start with a random token. The content before the token is appended at the end of the document. This gives insights into how the document is typically arranged and how the beginning or ending of a document looks like.

However, not all transformations are employed in training the final BART model. Based on a comparative study of pre-training objectives, the authors use only text infilling and sentence permutation transformations, with about 30% of tokens being masked and all sentences permuted.

- These transformations are applied to 160GB of text from the English Wikipedia and Book Corpus dataset. With this dataset, the vocabulary size is around 29000, and the maximum length of the sequences is 512 characters in the clean data.

Although we separate the decoder from an encoder, the input to the decoder would still be a learned representation (or embedding) of the original text sequence. Thus, BART attaches the bi-directional encoder to the autoregressive decoder to create a denoising auto-encoder architecture.

The pre-training is done using the masked sequences as discussed previously and shown below. While [BERT](#) was trained by using a simple token masking technique, BART empowers the BERT encoder by using more challenging kinds of masking mechanisms in its pre-training.

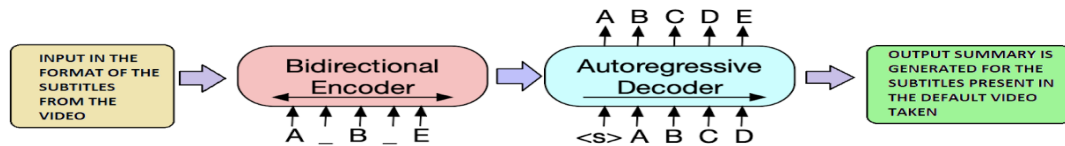


Fig. 3.4.2: BART Working

In the above figure, the input sequence is a masked (or noisy) version of [ABCDE] transformed into [A[MASK]B[MASK]E]. The encoder looks at the entire sequence and learns high-dimensional representations with bi-directional information. The decoder takes these thought vectors and regressively predicts the next token. Learning occurs by computing and optimizing the negative log-likelihood as mapped with the target [ABCDE].



## 4. ANALYSIS

## 4.ANALYSIS

This section first provides a brief background about the research domain. Then, the related work is presented in detail.

### A. Background

The background refers to the YouTube video transcript summarization using the NLP field in terms of the intersection of multiple research sectors. This field can be viewed as the intersection of four main domains; the definitions of the domains and terms that are applied in this study are listed below.

#### 4.1 Abstractive Summarization:

The Abstractive methods use advanced techniques to get a whole new summary. Some parts of this summary might not even appear within the original text.

Abstractive summarization uses the Pegasus model by Google. The model uses Transformers Encoder-Decoder architecture. The encoder outputs masked tokens while the decoder generates Gap sentences.

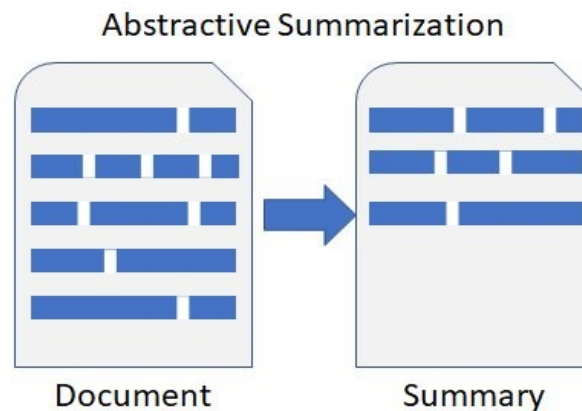


Fig. 4.1.1: Abstractive Summarization

Abstractive summarization aims to take a body of text and turn it into a shorter version. Not only does abstractive summarization shorten the body of texts, but it also generates new sentences. This is not the case for previous versions of text summarizations which only aim to generate accurate and concise summaries from input documents. It copies informative fragments from input sentences.

## 4.2 Extractive summarization:

In Extractive Summarization, we identify essential phrases or sentences from the original text and extract only these phrases from the text. These extracted sentences would be the summary.

**Extractive text summarization** methods function by identifying the important sentences or excerpts from the text and reproducing them verbatim as part of the summary. No new text is generated;

only existing text is used in the summarization process.

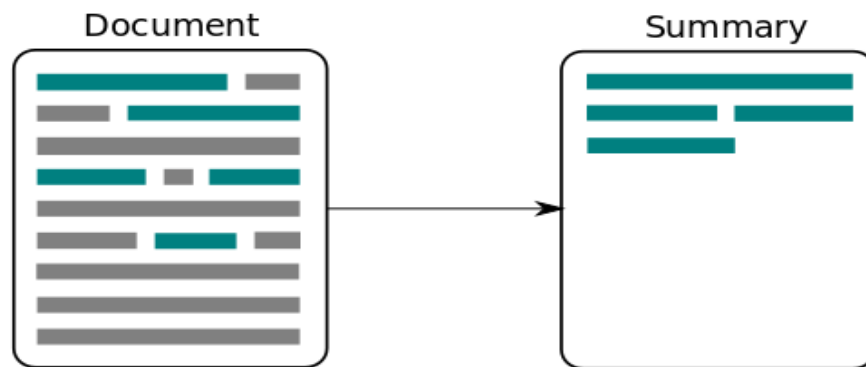


Fig 4.2.1: Extractive text Summarization

Extractive summarization methods work just like that. It takes the text, ranks all the sentences according to the understanding and relevance of the text, and presents you with the most important sentences.

This method does not create new words or phrases, it just takes the already existing words and phrases and presents only that. You can imagine this as taking a page of text and marking the most important sentences using a highlighter.

## 4.3 Summarization using TF-IDF vectorizer

First, let's discuss summarization using TF-IDF vectorizer.

TF-IDF or term frequency-inverse document frequency is a vectorizer that converts the text into a vector. It has 2 terms term frequency and inverse document frequency. TF-IDF value is the product of these 2 terms. Term frequency is the number of repetitions of words in a sentence by the total number of words in that sentence. Inverse document frequency is the log of no of sentences by the number of sentences containing the given word.

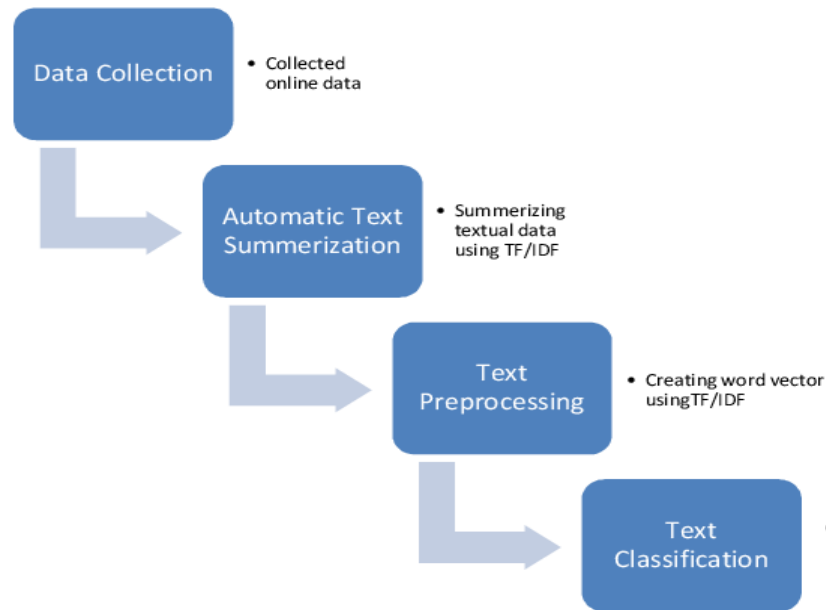


Fig 4.3.1: TF-IDF Vectorizer

Performing text summarization: This task consists of shortening a large form of text into a precise summary that keeps all the necessary information intact and preserves the overall meaning. For this purpose, in NLP for text summarization, there are two types of methods used: Extractive Summarization: In this type of text summarization, the output is only the important phrases and sentences that the model identifies from the original text. For the purpose of extractive summarization, we have used the TF-IDF model with Text Rank Algorithm. TF-IDF (Term Frequency - Inverse Document Frequency) After the cleaning process, we have to convert the words into its vectorized form so that our algorithm will process it by using TF-IDF. This is a technique to measure the quantity of a word in documents, we compute a weight to each word which signifies the importance of the word in the document and corpus. TF (Term Frequency): TF calculates the frequency of a word in a document.  $TF = \text{No. of repetition of the word in the sentence} / \text{No. of words in a sentence}$  IDF (Inverse Document Frequency): IDF is the inverse of the document frequency which measures the informativeness of term  $t$ .  $IDF = \log (\text{No. of sentences} / \text{No. of sentences containing words})$  After this, we will multiply both matrices to obtain the vectorized form which tells us which words are the most important.

**Text Rank Algorithm:** It is based on the PageRank algorithm which calculates the rank of web pages which is used by search engines such as Google. Using the concept of this we will rank the most important sentences in the text and generate a summary. For the task of automated

summarization, Text Rank models any document as a graph using sentences as nodes. A function is computing the similarity of sentences to build edges in between. This function is used to weight the graph edges, the higher the similarity between the sentences the more important the edge between them will be in the graph. Text Rank determines the relation of similarity between two sentences based on the content that both share. This overlap is calculated simply as the number of common lexical tokens between them, divided by the length of each to avoid promoting long sentences. Cosine similarity of two words A & B is computed using following formula:

$$\text{similarity}(A,B) = \frac{A \cdot B}{\|A\| \times \|B\|} = \frac{\sum_{i=1}^n A_i \times B_i}{\sqrt{\sum_{i=1}^n A_i^2} \times \sqrt{\sum_{i=1}^n B_i^2}}$$

#### 4.4 Summarization using BART

BART (Bidirectional and Auto-Regressive Transformer) is a transformer that is now commonly used for sequence-to-sequence problems. Its architecture mainly consists of a Bidirectional encoder and a left-to-right decoder. BART is suitable for summarization, machine translation, question-answering, etc.

BART is one such Transformer model that takes components from other Transformer models and improves the pretraining learning. BART or Bidirectional and Auto-Regressive. BART is a sequence-to-sequence model trained as a denoising autoencoder. This means that a fine-tuned BART model can take a text sequence (for example, English) as input and produce a different text sequence at the output (for example, French). This type of model is relevant for machine translation (translating text from one language to another), question-answering (producing answers for a given question on a specific corpus), text summarization (giving a summary of or paraphrasing a long text document), or sequence classification (categorizing input text sentences or tokens). Another task is sentence entailment which, given two or more sentences, evaluates whether the sentences are logical extensions or are logically related to a given statement.

**1.Encoder** - It consists of several recurrent units such as LSTM, which captures the context of the input sequence in the form of a hidden state vector and generates a final embedding at the

end of the sequence which is known as context vector, this is then forward to the decoder. - In our project, we have used three encoders consisting of recurrent unit LSTM.

**2.Decoder** - A context vector is then sent to the decoder which also consists of recurrent unit LSTM, which then uses it to predict the output sequence  $y(t)$  at a time step  $t$ , and after each successive prediction, it uses the previous hidden state to predict the subsequent sequence.

Since the unsupervised pre training of BART results in a language model, we can fine-tune this language model to a specific task in [NLP](#). Because the model has already been pre-trained, fine-tuning does not need massive labelled datasets (relative to what one would need for training from scratch). The BART model can be fine-tuned to domain-specific datasets to develop applications such as medical conversational chatbots, converting natural text to programming code or SQL queries, context-specific language translation apps, or a tool to paraphrase research papers.

The GPT-1 model used an architecture similar to the decoder segment of the vanilla Transformers. GPT sequentially stacks 12 such decoders such that learning from only the past tokens can affect the current token calculation. The architecture is shown above. As seen in the original Transformer decoder, the GPT decoder also uses a masked multiheaded self-attention block and a feed-forward layer.

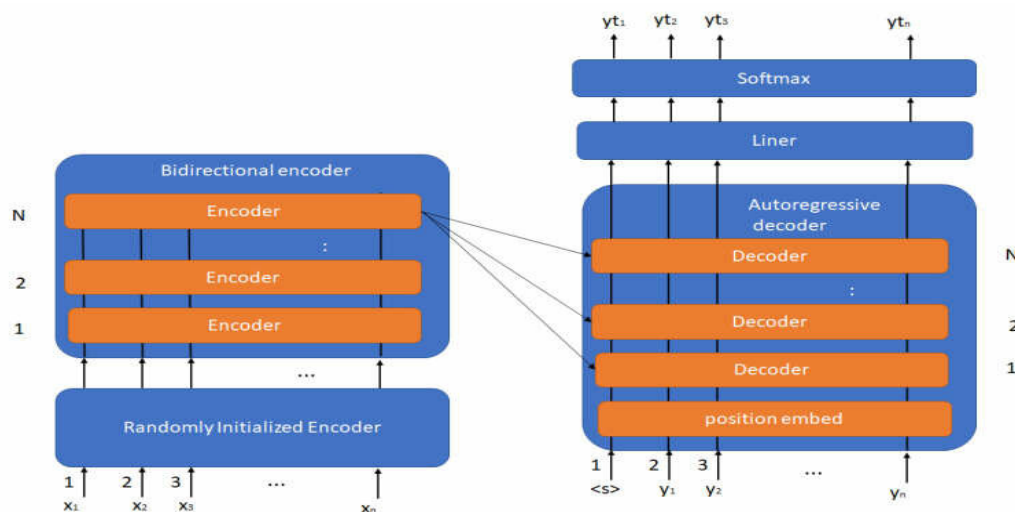


Fig 4.4.1: BART Architecture

BART is constructed from a bi-directional encoder like in BERT and an autoregressive decoder like GPT. BERT has around 110M parameters while GPT has 117M, such as trainable weights. BART being a sequenced version of the two, fittingly has nearly 140M parameters. Many parameters are justified by the supreme performance it yields on several tasks compared to fine-tuned BERT or its variations like RoBERTa, which has 125M parameters in its base model.

As more and more long-form content burgeons on the internet, it is becoming increasingly time-consuming for someone like a researcher or journalist to filter out the content they want. Summaries or paraphrase synopsis help readers quickly go through the highlights of a huge amount of textual content and save enough time to study the relevant documents.

Transformer models can automate this NLP task of text summarization. There are two approaches to achieve this: extractive and abstractive. Extractive summarization identifies and extracts the most significant statements from a given document as they are found in the text. This can be considered more of an information retrieval task. Abstractive summarization is more challenging as it aims to understand the entire document and generate paraphrased text to summarize the main points. Transformer models, including BART, perform the latter kind of summarization.

## **4.5 TOKENIZATION**

Natural language processing is one of the fields in programming where the natural language is processed by the software. This has many applications like sentiment analysis, language translation, fake news detection, grammatical error detection etc.

The input in natural language processing is text. The data collection for this text happens from a lot of sources. This requires a lot of cleaning and processing before the data can be used for analysis.



Fig. 4.5.1: Tokenization

These are some of the methods of processing the data in NLP:

- Tokenization
- Stop words removal
- Stemming
- Normalization
- Lemmatization
- Parts of speech tagging

Tokenization is breaking the raw text into small chunks. Tokenization breaks the raw text into words, sentences called tokens. These tokens help in understanding the context or developing the model for the NLP. The tokenization helps in interpreting the meaning of the text by analysing the sequence of the words.

For example, the text “It is raining” can be tokenized into ‘It’, ‘is’, ‘raining’.



There are different methods and libraries available to perform tokenization. NLTK, Gensim, Keras are some of the libraries that can be used to accomplish the task.

Tokenization can be done to either separate words or sentences. If the text is split into words using some separation technique it is called word tokenization and the same separation done for sentences is called sentence tokenization.

Stop words are those words in the text which do not add any meaning to the sentence and their removal will not affect the processing of text for the defined purpose. They are removed from the vocabulary to reduce noise and to reduce the dimension of the feature set.

There are various tokenization techniques available which can be applicable based on the language and purpose of modelling. Below are a few of the tokenization techniques used in NLP.

## **5.DESIGN**

## 5. DESIGN

### 5.1 Flow Chart of the Proposed Approach:

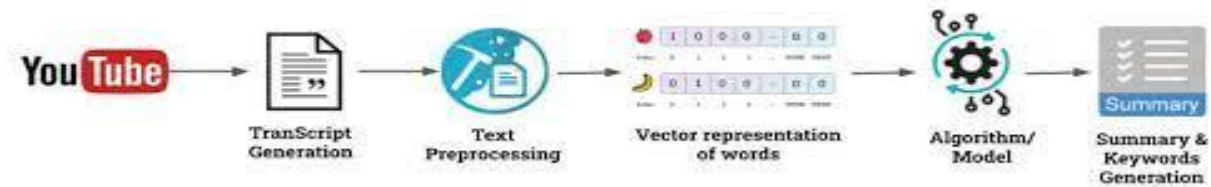


Fig. 5.1.1: Approach of working model

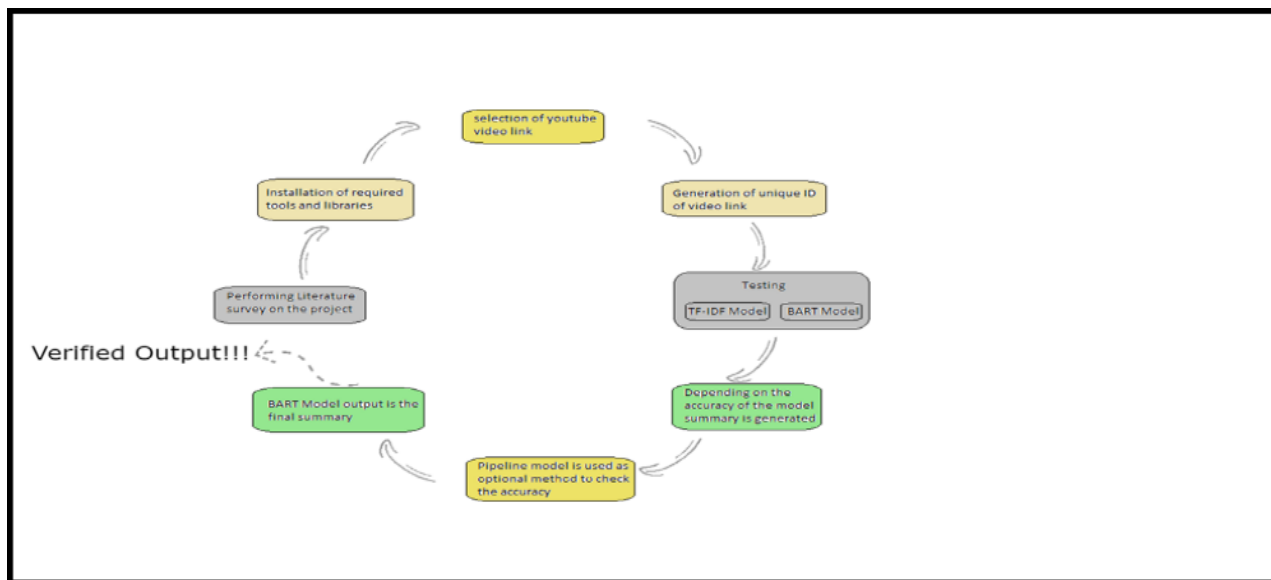


Fig. 5.1.2: Design phase of the Proposed Approach

So basically, it will be a chrome extension, having an option to copy to the current URL of the video being selected. After providing the link, it will access the transcript of the particular audio using the YouTube transcript API and then the transcript will be provided to a machine learning model that will in return provide the summarized text of the transcript. The summarized text would be downloadable by the user.

## **6. IMPLEMENTATION AND RESULTS ANALYSIS**

## 6. IMPLEMENTATION AND RESULT ANALYSIS

### 6.1 Method of Implementation:

The project goals are listed below:

- To build the summarizer we will be using 3 different types of models.
- To develop this the model used is the BART model.
- To be modular and robust
- To be test covered

#### Extended Project Goals

We want to spread our wings and expand our project and build a site which is less complex and can be easily available and helpful for the people.

#### 6.1.1 Source Code

```
!pip install -q transformers
!pip install -q youtube_transcript_api
import youtube_transcript_api
from youtube_transcript_api import YouTubeTranscriptApi
import nltk
import re
from nltk.corpus import stopwords
import sklearn
from sklearn.feature_extraction.text import TfidfVectorizer
link = "https://www.youtube.com/watch?v=Y8Tko2YC5hA"
unique_id = link.split("=")[-1]
sub = YoutubeTranscripts = YouTubeTranscriptApi.get_transcript(unique_id)
subtitle = " ".join([x['text'] for x in sub])
from nltk.tokenize import sent_tokenize
```

```
import nltk
```

```
nltk.download('punkt')
```

```
sentences = sent_tokenize(subtitle)
```

```
sentences
```

```
organized_sent = {k:v for v,k in enumerate(sentences)}
```

```
tf_idf = TfidfVectorizer(min_df=2,  
                        strip_accents='unicode',  
                        max_features=None,  
                        lowercase = True,  
                        token_pattern=r'w{1,}',  
                        ngram_range=(1, 3),  
                        use_idf=1,  
                        smooth_idf=1,  
                        sublinear_tf=1,  
                        stop_words = 'english')
```

```
N = 3
```

```
top_n_sentences= [sentences[index] for index in np.argsort(sent_scores, axis=0)[:N][::-1][:-1]]
```

```
mapped_sentences = [(sentence,organized_sent[sentence]) for sentence in  
top_n_sentences]
```

```
mapped_sentences = sorted(mapped_sentences, key = lambda x: x[1])
```

```
ordered_sentences = [element[0] for element in mapped_sentences]
```

```
summary = " ".join(ordered_sentences)
```

```
summary
```

```
!pip install transformers
```

```
import transformers
```

```
from transformers import BartTokenizer, BartForConditionalGeneration
```

```
tokenizer = BartTokenizer.from_pretrained('facebook/bart-large-cnn')
```

```
model = BartForConditionalGeneration.from_pretrained('facebook/bart-large-cnn')
```

```
input_tensor = tokenizer.encode(subtitle, return_tensors="pt",
```

```
max_length=512, truncation=True)
```

```
outputs_tensor = model.generate(input_tensor, max_length=160, min_length=120,
```

```
length_penalty=2.0, num_beams=4, early_stopping=True)
outputs_tensor
print(tokenizer.decode(outputs_tensor[0]))
!pip install -q transformers
!pip install -q youtube_transcript_api

from transformers import pipeline
from youtube_transcript_api import YouTubeTranscriptApi
youtube_video = "https://www.youtube.com/watch?v=Y8Tko2YC5hA"

video_id = youtube_video.split("=")[1]
video_id
from IPython.display import YouTubeVideo
YouTubeVideo(video_id)
YouTubeTranscriptApi.get_transcript(video_id)
transcript = YouTubeTranscriptApi.get_transcript(video_id)
transcript[0:5]
result = ""
for i in transcript:
    result += ' ' + i['text']
print(len(result))
summarizer = pipeline('summarization')
num_iters = int(len(result)/1000)
summarized_text = []
for i in range(0, num_iters + 1):
    start = 0
    start = i * 1000
    end = (i + 1) * 1000
    print("input text \n" + result[start:end])
    out = summarizer(result[start:end])
    out = out[0]
```

```
out = out['summary_text']
```

```
print("Summarized text\n"+out)
```

```
summarized_text.append(out)
```

```
len(str(summarized_text))
```

```
str(summarized_text)
```



### 6.1.2 Output Screens:

```
!pip install -q transformers

|████████████████████████████████████████████████████████████████████████████████| 5.5 MB 4.0 MB/s
|████████████████████████████████████████████████████████████████████████████████| 7.6 MB 50.3 MB/s
|████████████████████████████████████████████████████████████████████████████████| 182 kB 43.1 MB/s

[ ] !pip install -q youtube_transcript_api

[ ] import youtube_transcript_api
    from youtube_transcript_api import YouTubeTranscriptApi
    import nltk
    import re
    import numpy as np
    from nltk.corpus import stopwords
    import sklearn
    from sklearn.feature_extraction.text import TfidfVectorizer

▶ link = "https://www.youtube.com/watch?v=Y8Tko2YC5h"
    unique_id = link.split("=")[-1]
    sub = YouTubeTranscriptApi.get_transcript(unique_id)
    subtitle = " ".join([x['text'] for x in sub])
```

```
[ ] subtitle
```

```
[ ] from nltk.tokenize import sent_tokenize
```

```
[ ] import nltk  
    nltk.download('punkt')
```

```
[ ] subtitle = subtitle.replace("\n","")  
    sentences = sent_tokenize(subtitle)
```

```
[ ] sentences
```

```
[ ] organized_sent = {k:v for v,k in enumerate(sentences)}
```

```
[ ] tf_idf = TfidfVectorizer(min_df=2,  
                             strip_accents='unicode',  
                             max_features=None,  
                             lowercase = True,  
                             token_pattern=r'w{1,}',  
                             ngram_range=(1, 3),  
                             use_idf=1,  
                             smooth_idf=1,  
                             sublinear_tf=1,  
                             stop_words = 'english')
```

```
[ ] sentence_vectors = tf_idf.fit_transform(sentences)  
    sent_scores = np.array(sentence_vectors.sum(axis=1)).ravel()
```

```
[ ] N = 3  
    top_n_sentences = [sentences[index] for index in np.argsort(sent_scores, axis=0)[::-1][:N]]
```

```
[ ] # mapping the scored sentences with their indexes as in the subtitle
    mapped_sentences = [(sentence,organized_sent[sentence]) for sentence in top_n_sentences]
    # Ordering the top-n sentences in their original order
    mapped_sentences = sorted(mapped_sentences, key = lambda x: x[1])
    ordered_sentences = [element[0] for element in mapped_sentences]
    # joining the ordered sentence
    summary = " ".join(ordered_sentences)
```

```
[ ] summary
```

```
[ ] above is tf idf.....below is bart
```

```
[ ] !pip install transformers
```

```
[ ] import transformers
    from transformers import BartTokenizer, BartForConditionalGeneration
```

```
▶ tokenizer = BartTokenizer.from_pretrained('facebook/bart-large-cnn')
    model = BartForConditionalGeneration.from_pretrained('facebook/bart-large-cnn')
```

```
[ ] input_tensor = tokenizer.encode( subtitle, return_tensors="pt", max_length=512,truncation=True)
```

```
[ ] outputs_tensor = model.generate(input_tensor, max_length=160, min_length=120, length_penalty=2.0, num_beams=4, early_stopping=True)
outputs_tensor
```

```
[ ] print(tokenizer.decode(outputs_tensor[0]))
```

```
[ ] above is bart and below is transformers
```

```
[ ] !pip install -q transformers
```

```
[ ] !pip install -q youtube_transcript_api
```

```
[ ] from transformers import pipeline
from youtube_transcript_api import YouTubeTranscriptApi
```

```
[ ] youtube_video = "https://www.youtube.com/watch?v=Y8Tko2YC5hA"
```

```
 video_id = youtube_video.split("=")[1]
```

```
[ ] video_id
```

```
[ ] from IPython.display import YouTubeVideo
    YouTubeVideo(video_id)
```

```
[ ] YouTubeTranscriptApi.get_transcript(video_id)
    transcript = YouTubeTranscriptApi.get_transcript(video_id)
```

```
[ ] transcript[0:5]
```


```
[ ] result = ""
    for i in transcript:
        result += ' ' + i['text']
    #print(result)
    print(len(result))
```

```
[ ] summarizer = pipeline('summarization')
```

```
[ ] num_iters = int(len(result)/1000)
    summarized_text = []
    for i in range(0, num_iters + 1):
        start = 0
        start = i * 1000
        end = (i + 1) * 1000
        print("input text \n" + result[start:end])
        out = summarizer(result[start:end])
        out = out[0]
        out = out['summary_text']
        print("Summarized text\n"+out)
        summarized_text.append(out)

    #print(summarized_text)
```

```
[ ] len(str(summarized_text))
```

```
 str(summarized_text)
```

[" Python is the world's fastest growing and most popular programming language . People from different disciplines use Python for a variety of different tasks, such as data analysis and visualization, artificial intelligence and machine learning . Automation is one of the big uses of Python amongst people who are not software developers .", ' Python is a multi purpose language . You can use it to build web, mobile and desktop applications as well as software testing or even hacking . With Python you can solve complex problems in less time with fewer lines of code . You could be an accountant, a mathematician, or a scientist, and use Python to make your life easier .', ' ' Python makes a lot of trivial things really easy with a simple yet powerful syntax . It's a high level language so you don't have to worry about complex tasks such as memory management, like you do in C++ . It has a huge community so whenever you get stuck, there is someone out there to help .", " Python is the number one language employers are looking for . The average Python developer earns a whopping 116,000 dollars a year . There's never been a better time to master Python, and it's the perfect time to learn . Here's my Python tutorial for beginners, it's a great starting point if you have limited or no programming experience .", ' Programming is programming, so click on the tutorial that is right for you and get started . Click here to learn how to use the tutorial to get started with programming . Do you know what you need to do with programming? Email us at <http://www.mailonline.co.uk.com/storystorystory> .']

# CONCLUSION

## CONCLUSION

Recently, video summarization has attracted considerable interest from researchers and as a result, various algorithms and techniques have been proposed. This project is to provide a web app or a chrome extension that can be used to summarize the YouTube video content and extract important information from those patterns by using state-of-the-art Natural Language Processing methods for abstractive text summarization and Machine Learning for classification.

In general, the YouTube summarizer will give us a brief summary of the video which will be useful for saving a lot of time (Note that, it is suitable for video with subtitles).

The accessed transcripts are then summarized with the transformers package. Then the summarized text is shown to the user in the chrome extension web page. This project helps the users a lot by saving their valuable time and resources. This helps us to get the gist of the video without watching the whole video. It also helps the user to identify the unusual and unhealthy content so that it may not disturb their viewing experience. This project also ensures great user interface experience in finding out the summarized text as chrome extensions have been used. This helps in getting the summarized text without copying the URL and pasting at terminals or by any third-party applications.

The increase in popularity of video content on the internet requires an efficient way of representing or managing the video. This can be done by representing the videos on the basis of their summary. Learning how to set up web services using API, create Google Chrome extensions and implement Cloud Computing. In addition, we also want to use HTML and CSS to develop WebApps and write software packages in python. We need to follow time management and fully grasp the difficulties which may occur and when to change the course of the project to use our time more efficiently. It is important to understand connections between different technologies and to account for possible bugs when incorporating different software packages into the corpus of a final software product. And finally build a full front-end work for this project.



## References

- [1].Gang Liu, Jiabao Guo. 2019. Bidirectional LSTM with attention mechanism and convolutional layer for text classification. School of Computer Science, Hubei University of Technology, Wuhan , China. (Feb,2019).
- [2]. Adhika Pramita Widyassari, Supriadi Rustad, Guruh FajarShidik, Edi Noersasongko, Abdul Syukur, Affandy Affandy, De Rosal Ignatius Moses Setiadi,Review of automatic text summarization techniques & methods,Journal of King Saud University - Computer and Information Sciences,2020, ISSN 1319-1578, <https://doi.org/10.1016/j.jksuci.2020.05.006>.
- [3].A. Dilawari and M. U. G. Khan, "ASoVS: Abstractive Summarization of Video Sequences," in IEEE Access, vol. 7, pp. 29253-29263, 2019, doi: 10.1109/ACCESS.2019.2902507.
- [4].Anuj Gupta,Bodhisattwa Prasad Majumder,Harshit Surana and Sowmya Vajjala."Practical Natural language Processing" A Comprehensive guide to build Real World NLP System.1st edition. O'Reilly Media, Inc.(June,2020)
- [5].Ankit Kumar, Zixin Luo & Ming Xu,"Text Summarization using Natural Language Processing". WPI Juniper Networks.(March,2018).
- [6].Gaurav Sharma, Shaba Parveen Khan, Shivanshu Sharma and Syed Ubed Ali (2021)."Summarizer For Easy Video Assessment(SEVA)".Vol.03. e-ISSN: 2582-5208. (April,2021).
- [7].Korra Sathya Babu,Santosh Kumar Bharti and Sanjay Kumar Jena.2017. "Automatic Keyword Extraction for Text Summarization: A Survey".(Feb,2017).
- [8].Pratik K. Biswas, Aleksander Lakubovich. (2020)" Extractive Summarization On Call Transcript".AI & Data Science, Global Network and Technology, Verizon Communications, New Jersey, USA.
- [9].Cuneyt M. Taskiran , Arnon Amir, Dulce Ponceleon and Edward J. Delp. 2009. "Automated Video Summarization Using Speech Transcripts".Video and Image Processing Laboratory (VIPER).(Jan, 2009)
- [10].Gang Liu, Jiabao Guo. 2019. Bidirectional LSTM with attention mechanism and convolutional layer for text classification. School of Computer Science, Hubei University of Technology, Wuhan , China. (Feb,2019)..