A Project Report

on

YouTube Transcript Summarizer

Submitted in partial fulfillment of the

requirement for** the **award of** the **degree of

Bachelor of Technology in Computer Science and Engineering

(Established under Galgotias University Uttar Pradesh Act No. 14 of 2011)

Under The Supervision ofMr.
Arvindhan M
Assistant Professor
Department of Computer Science and Engineering

Submitted By

19SCSE1140019 – PREMANSHU SINHA

19SCSE1010053 – VISHWANG GOYAL

SCHOOL OF COMPUTING SCIENCE AND ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERINGGALGOTIAS UNIVERSITY, GREATER NOIDA, INDIA

DECEMBER - 2021

ABSTRACT

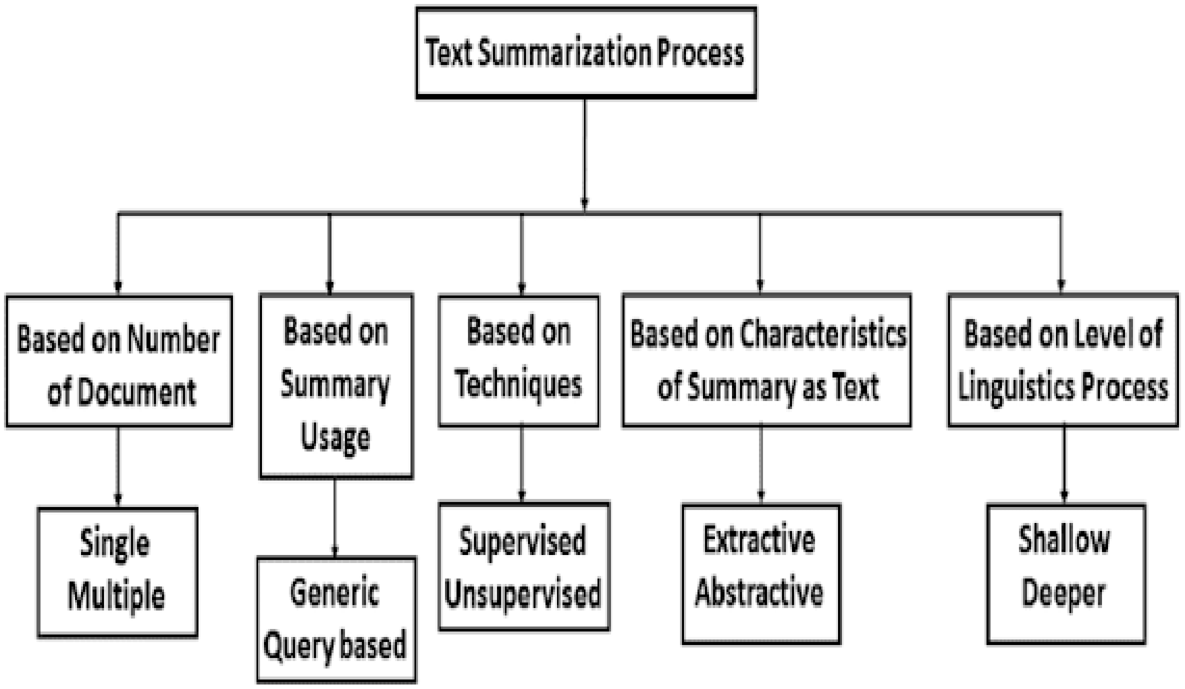
In this project, we will be creating a Chrome Extension which will make a request to a backend REST API where it will perform NLP and respond with a summarized version of a YouTube transcript. Enormous number of video recordings are being created and shared on the Internet throughout the day. It has become really difficult to spend time watching such videos which may have a longer duration than expected and sometimes our efforts may become futile if we couldn't find relevant information out of it. Summarizing transcripts of such videos automatically allows us to quickly lookout for the important patterns in the video and helps us to save time and effort to go through the whole content of the video. This project will give us 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.

INTRODUCTION

Enormous variety of video recordings area unit being created and shared on the net throughout the day. It’s become extremely tough to pay time in look such videos could which can have an extended period than expected and generally our efforts may become futile if we tend to couldn’t notice relevant info out of it. Summarizing transcripts of such videos mechanically permits USA to quickly look out for the necessary patterns within the video and helps USA to avoid wasting time and efforts to travel through the complete content of the video.

The number of YouTube users in two020 was more or less 2.3 billion, and has been increasing once a year. Each minute, three hundred hours of YouTube videos area unit uploaded. Nearly tierce of the YouTube viewers in Asian country access videos on their mobiles and pay over forty eight hours a month on the web site, a Google study aforementioned [11]. It’s frustrating and time intense to look for the videos that contain the data we tend to are literally trying to find. As an example, there are units several tough guy speak videos out there on-line within which the speaker talks for a protracted time on a given topic, however it's exhausting to seek out the content the speaker is principally that specialize in unless we tend to watch the whole video. several machines learning primarily based video account techniques area unit gift however they need devices with massive process powers, this is often as a result of every video contains thousands of frames and process all frames takes an awfully very long time. During this paper we tend to propose to use the LSA tongue process rule, which needs less process power and no coaching knowledge needed to coach the rule.

This project can enable us to own active expertise with progressive IP techniques for theoretic text account and implement a remarkable plan appropriate for intermediates and a refreshing hobby project for professionals.


Text Summarization Process
Based on Number of Document
Based on Summary Usage
Based on Techniques
Based on Characteristics of Summary as Text
Based on Level of Linguistics Process
Single
Multiple
Generic
Query based
Supervised
Unsupervised
Extractive
Abstractive
Shallow
Deeper

High-Level Approach

- Get transcripts/subtitles for a given YouTube video Id using a Python API.
- Perform text summarization on obtained transcripts using HuggingFace transformers.
- Build a Flask backend REST API to expose the summarization service to the client.
- Develop a chrome extension which will utilize the backend API to display summarized text to the user.

Applications

- Meetings and video-conferencing - A system that could turn voice to text and generate summaries from your team meetings.
- Patent research - A summarizer to extract the most salient claims across patents.

LITERATURE REVIEW

TASK 1 - BACKEND:

APIs changed the way we build applications, there are countless examples of APIs in the world, and many ways to structure or set up APIs. Here we are going to create a back-end application directory and structure it to work with the required files. We will isolate the back-end of the application to avoid conflicting dependencies from other parts of the project.
Requirements:

- Create back-end application directory containing files named as app.py and requirements.txt.

- Initialize app.py file with basic Flask RESTful BoilerPlate.

- Create a new virtual environment with pip installed which will act as an isolated location (a directory) where everything resides.

- Activate the newly installed virtual environment and install the following dependencies:

1. Flask
2. youtube_transcript_api
3. transformers[torch]

- Execute pip freeze and redirect the output to the requirements.txt file. This file is used for specifying what python packages are required to run the project

References

- Creating a Virtual Environment in Python
- Building RESTful APIs with Flask in Python BoilerPlate
- HuggingFace Transformer Python Installation

Expected Outcome

You are expected to initialize the back-end portion of your application with the required boiler plate as well as the dependencies.

TASK 2 – GET TRANSCRPT FOR A VIDEO:

.

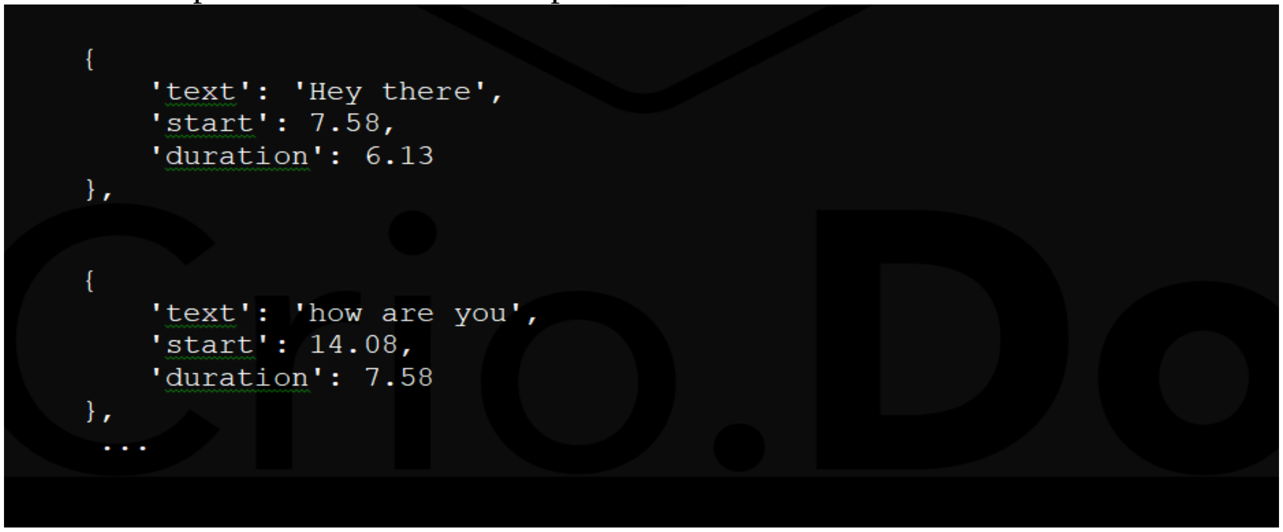
We will utilize a python API which allows transcripts/subtitles for a given YouTube video. It also works for automatically generated subtitles, supports translating subtitles, and does not require a headless browser like other Selenium-based solutions do.

Requirements:

In app.py,

- - Create a function which will accept YouTube video id as input parameter and return parsed full transcript as output.

- - The response from the Transcript API will return a list of dictionaries.



```
{
    'text': 'Hey there',
    'start': 7.58,
    'duration': 6.13
},

{
    'text': 'how are you',
    'start': 14.08,
    'duration': 7.58
},
 ...
```

References

- YouTube Transcript API Documentation
- Read, Write and Parse JSON using Python

Expected Outcome

You should be able to fetch the transcripts with the help of a function created which we will later utilize as a feed input for the NLP processor in the pipeline.

TASK 3 – PERFORM TEXT SUMMARIZATION:

Text summarization is the task of shortening long pieces of text into a concise summary that preserves key information content and overall meaning.

There are two different approaches that are widely used for text summarization:

- **Extractive Summarization:** This is where the model identifies the important

sentences and phrases from the original text and only output those.
- **Abstractive Summarization:** The model produces a completely different text that is shorter than the original; it generates new sentences in a new form, just like humans do. Here we'll use transformers for this approach.

We will use HuggingFace's transformers library in Python to perform abstractive text summarization on the transcript previously obtained.

Requirements:

In app.py,

- -Create a function which will accept YouTube transcript as an input parameter and return summarized script as output.

- -Instantiate a tokenizer and a model from the checkpoint name.

- -Summarization is usually done using encoder-decoder model, such as BART or T5.

- -Define the transcript that should be summarized.

- -Add the T5 specific prefix "summarize:"
- -Use the PreTrainedModel.generate() method to generate the summary.

NOTE: The transformer model used for the above project can take input size of maximum up to 1024 words. So the transcript size with more than 1024 words may throw Exception regarding the length of the transcript passed to it.

References

- How to Perform Text Summarization using Transformers in Python
- Transformers official documentation

Note

- The Transformer model used for the above project can only take text input size of maximum up to 1024 words. So the transcript size with more than 1024 words may throw Exception regarding the length of the transcript passed to it.

Expected Outcome

You should be able to verify that the model generates a completely new summarized text that is different from the original text.

TASK 4 – CREATE REST API ENDPOINT:

The next step is to define the resources that will be exposed by this backend service. This is an extremely simple application, we only have single endpoint, so our only resource will be summarized text.

Requirements: In app.py,

- Create a Flask API Route with GET HTTP Request method with a URI, http://[hostname]/api/summarize?youtube url=<url>.
- Extract the YouTube from YouTube URL which is obtained from the query params.
- Generate the summarized transcript by executing the transcript generation function following the execution of the transcript summarizer function.
- Return the summarized transcript with HTTP Status OK and handle HTTP exception if applicable.
- Run the Flask Application and test the endpoint in Postman to verify the appropriate results.

References

- Designing a RESTful API with Python and Flask
- Parsing REST API Payload and Query Parameters With Flask

Expected Outcome

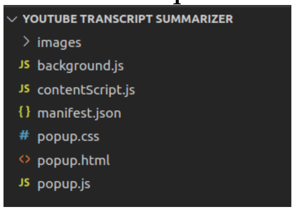
You should be able to create an endpoint to summarize YouTube video transcripts and test the response with different video URLs.

TASK 5 – CHROME EXTENSION:

Extensions are small software programs that customize the browsing experience. They enable users to tailor Chrome functionality and behaviour to individual preferences. They are built on web technologies such as HTML, CSS and JavaScript. In this task we will see how to create a recommended Chrome extension application directory and structure it to work with the required files.

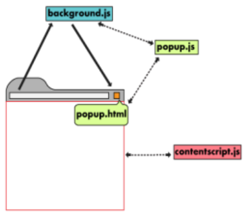
Requirements:

- Create a chrome extension application directory containing essential files required.

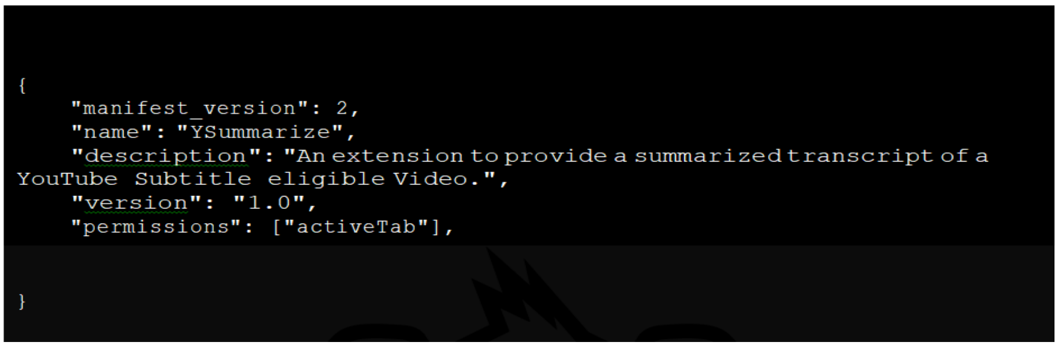


```
YOUTUBE TRANSCRIPT SUMMARIZER
  images
  background.js
  contentScript.js
  manifest.json
  popup.css
  popup.html
  popup.js
```

- The below diagram indicates the brief role of each of the files for building a chrome extension.



- Code:



```
{
    "manifest_version": 2,
    "name": "YSummarize",
    "description": "An extension to provide a summarized transcript of a
YouTube Subtitle eligible Video.",
    "version": "1.0",
    "permissions": ["activeTab"],

}
```

- All we need to do is:

- Just visit chrome://extensions and turn on developer mode from the top right-hand corner.

- Then click on load unpacked and select the folder containing the manifest file that we just created.

- There the extension is up and running.

References

- The Ultimate Guide to Building a Chrome Extension
- How to Create Chrome Extensions

NOTE: We’ll need to reload the extension every time we make a change in the extension.

Expected outcome

We should be able to create a recommended Chrome extension application directory and structure it to work with the required files.

Task 6

Build a User Interface for Extension Popup

We need a user interface so that the user can interact with the popups which are one of several types of user interface that a Chrome extension can provide. They usually appear upon clicking the extension icon in the browser toolbar.

```
{
 .
 .
 .
 "page_action": {
       "default_popup": "popup.html",
 }
 .
 .
}
```

Requirements

- Add the line below to page_actionin the manifest file which enables the User Interface for a Popup.
- In the popup.htmlfile,
  - Include the popup.cssfile to make the styles available to the HTML elements.
  - Include the popup.jsfile to enable user interaction and behavior with the HTML elements.
  - Add a buttonelement named Summarizewhich when clicked will emit a click event which will be detected by an event listener to respond to it.
  - Add a divelement where summarized text will be displayed when received from backend REST API Call.
- In popup.cssfile,
  - Provide appropriate CSS styling to the HTML elements buttonand divto have a better user experience.

References
- Design the user interface
- What is page_action in a manifest file?

Expected Outcome

The extension user interface should be purposeful and minimal and must enhance the browsing experience without distracting from it.

Task 7

Display Summarized transcript

We have provided a basic UI to enable users to interact and display the summarized text but there are some missing links which must be addressed. In this milestone, we will add functionality to allow the extension to interact with the backend server using HTTP REST API Calls.

Requirements
- In popup.js,
  - When DOM is ready, attach the event listener with event type as "click" to the Summarizebutton and pass the second parameter as an anonymous callback function.
  - In anonymous function, send an action message generate using chrome.runtime.sendMessagemethod to notify contentScript.jsto execute summary generation.
  - Add event listener chrome.runtime.onMessageto listen message resultfrom contentScript.jswhich will execute the outputSummarycallback function.
  - In callback function, display the summary in the divelement programmatically using Javascript.
- Add the line below to content_scriptsin the manifest file which will inject the content script contentScript.jsdeclaratively and execute the script automatically on a particular page.

```
{
 .
 .
 .
 "content_scripts":[
   {
    "matches":["https://www.youtube.com/wat
    ch?v=*"], "js": ["contentScript.js"]
   }
 ],
 .
.
 .
}
```

- In contentScript.js,

  - Add event listenerchrome.runtime.onMessagetolisten messagegeneratewhichwillexecutethegenerateSummarycallback function.

  - In callback function, extract the URL of the current tab and make a GET HTTP request usingXMLHTTPRequestWeb API to the backendtoreceive summarized text as aresponse.

  - Send an action messageresultwith summary payload usingchrome.runtime.sendMessageto notifypopup.js` to display the summarized text.

References

- Content Scripts
- Message Passing in Chrome
- How to use XMLHttpRequest to issue HTTP requests

Expected Outcome

The extension user interface should be able to display the summarized text upon request from the user.

Task 8

Finalizing

As the basic implementation is all done, for all the curious cats out there, these are some of the line items which can be implemented to spice up the existing functionality.

[Note: This is not a mandatory milestone.]

Requirements

- Try to do the following:
  - Can you add functionality to summarize very long transcripts using the extractive summarization technique (For e.g. using LSA technique)?
  - Can you add functionality to summarize transcripts from a non-English video and display it in the English language?
  - Can you add functionality to adjust the maximum length of the summarized text?
  - Can you add functionality to support transcript summarization from a video with no subtitles?

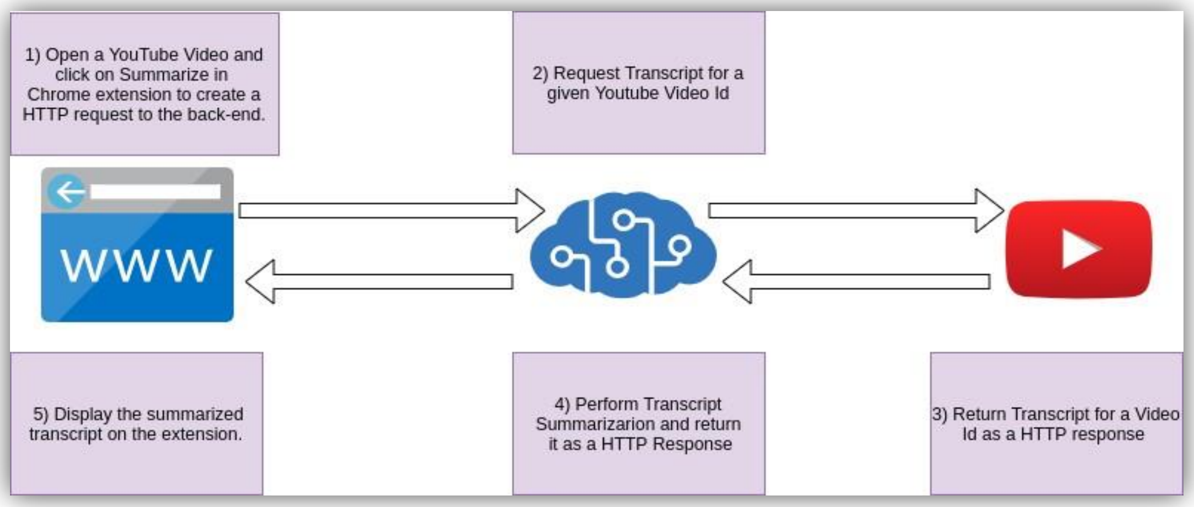
References

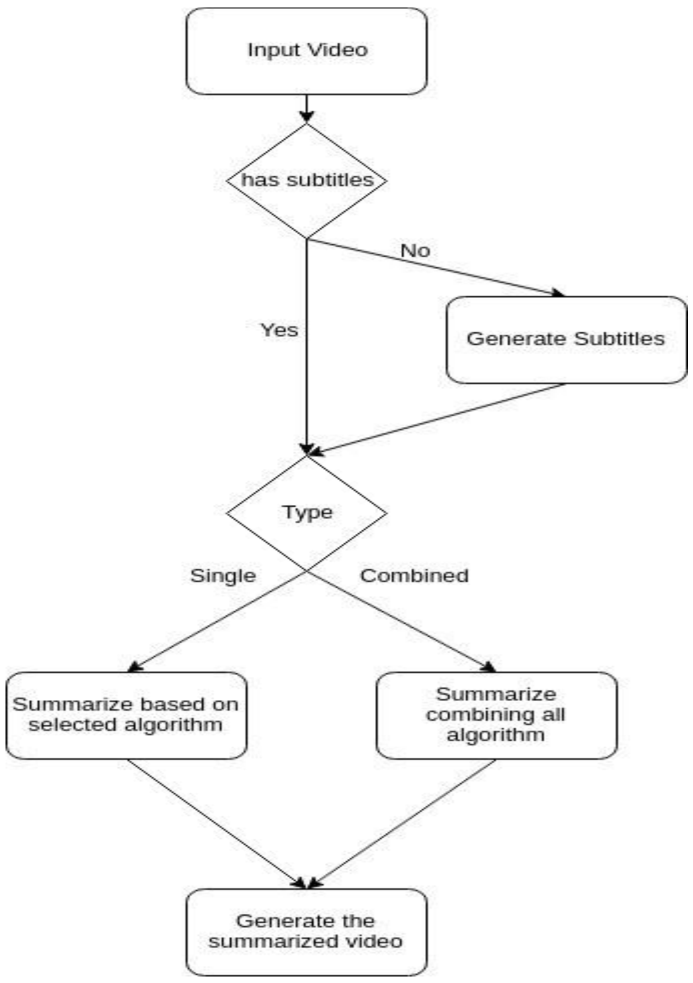
- Extractive Text Summarization Techniques With sumy
- Language Translator Using Google API in Python
- How to download YouTube video as audio using python
- Transcribing audio files using python

Expected Outcome

You should be able to add more features to your application.

PROJECT DESIGN


1) Open a YouTube Video and click on Summarize in Chrome extension to create a HTTP request to the back-end.
2) Request Transcript for a given Youtube Video Id
WWW
5) Display the summarized transcript on the extension.
4) Perform Transcript Summarizarion and return it as a HTTP Response
3) Return Transcript for a Video Id as a HTTP response


Input Video
has subtitles
No
Yes
Generate Subtitles
Type
Single
Combined
Summarize based on selected algorithm
Summarize combining all algorithm
Generate the summarized video

CONCLUSIONS

The increase in quality of video content on the net needs Associate in Nursing economical method of representing or managing the video. This will be done by representing the videos on the premise of their outline.

1. Applying language process on the subtitles. We’ve got a video with subtitles. We have a tendency to applied Associate in Nursing Automatic informatics primarily based LSA account rule on the subtitle to come up with the outline. Basically, we have a tendency to reborn the subtitles of the video into a text document then applied the account rule. Python library sumy provides the outline for a text document to the amount of sentences you specify as argument. There are several account algorithms that we will use with the assistance of this library. However we've got used the LSA rule.

2. Fitting the period that user provides. Exploitation the python library Sumy, it's potential to rank every sentence (or subtitles in our case). Every subtitle has bound period within the video. So as to suit the user period, we have a tendency to found the common period of every subtitle by dividing the full period of the video with the amount of subtitles. Using this average period we've got found the approximate variety of sentences that we want to supply the summarized video. This account technique works in such some way that the highest most hierarchic subtitles are taken into thought for the ultimate summarized video. If the full period of the summarized subtitles is additional, then it's potential to cut back the one that's least hierarchic and the other way around. During this method, it's potential to suit the video to the time provided by user.

3. Making the ultimate summarized video. Thus currently we have a tendency to get the outline of the subtitles and currently we've got to come up with the summarized video. We’ve got used the python module known as Moviepy. Exploitation the time stamps within the summarized subtitles we have a tendency to divide the video into many segments and eventually incorporate to form the ultimate summarized video. Thus by following the higher than steps we have a tendency to were able to generate the video account for the given video

REFERENCES

HuggingFace Transformer:
https://huggingface.co/docs/transformers/installation

Flask in Boilerplate:
https://atmamani.github.io/blog/building-restful-apis-with-flask-in-python/

API Documentation:
https://pypi.org/project/youtube-transcript-api/

Performing Text Summarization:
https://www.thepythoncode.com/article/text-summarization-using-huggingface-transformers-python

Building Chrome Extension:
https://medium.com/coding-in-simple-english/how-to-create-chrome-extension-7dd396e884ef

User Interface Design:
https://developer.chrome.com/docs/extensions/mv2/user_interface/

XMLHTTP Request:
https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest/Using_XMLHttpRequest

https://www.crio.do/projects/python-youtube-transcript/