TRANSCRIPT SUMMARIZER

MINI PROJECT REPORT

Submitted by

NAME OF THE CANDIDATE

in partial fulfilment for the award of the degree

of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

EXCEL ENGINEERING COLLEGE (AUTONOMOUS)
KOMARAPALAYM - 637303

APRIL/MAY-2023

TRANSCRIPT SUMMARIZER

A PROJECT REPORT

Submitted by

ABBA AM JOHN JUSTIN	730920104003
G.GANESH KUMAR	730920104030
K.NIKHIL KUMAR	730920104048
V.HEMANTH KUMAR	730920104035

in partial fulfillment for the award of the degree

of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

EXCEL ENGINEERING COLLEGE (AUTONOMOUS)

KOMARPALAYAM – 637303

ANNA UNIVERSITY :: CHENNAI 600 025

APRIL - 2023

EXCEL ENGINEERING COLLEGE (AUTONOMOUS)

KOMARAPALAYAM – 637 303

ANNA UNIVERSITY : CHENNAI – 600 025

BONAFIDE CERTIFICATE

Certified that this project report **"TRANSCRIPT SUMMARIZER"** is the bonafide work of "**ABBA AM JOHN JUSTIN (73090104003), G.GANESH KUMAR (730920104030), K.NIKHIL KUMAR (730920104048), V.HEMANTH KUMAR (730920104035)** " who carried out the project work under my supervision.

SIGNATURE

Dr.P.C.SENTHIL MAHESH M.E.,Ph.D.,

HEAD OF THE DEPARTMENT

Associate Professor,
Department of CSE,
Excel Engineering College,
(Autonomous),
Komarapalayam-637303.

SIGNATURE

Mrs,S.L.SWARNA M.Tech.,

SUPERVISOR

Assistant Professor,
Department of CSE,
Excel Engineering College,
(Autonomous),
Komarapalayam-637303.

Submitted for the University Examination held on_____

INTERNAL EXAMINER

EXTERNAL EXAMINER

ACKNOWLEDGEMENT

Behind every achievement lies an unfathomable sea of gratitude to those who actuated it, without them, it would never have into existence. To them, I lay words of gratitude imprinted within myself.

We wish our heartfelt thanks to my respected founder and Chairman of Excel Group Institutions **Prof. Dr. A. K. NATESAN, M.Com, M.B.A., M.Phil., Ph.D. FTA** and Vice Chairman **Dr. N. MATHAN KARTHICK M.B.B.S., M.H. Sc (Diabetology)** for following me to have the extensive use of the college facilities to do my mini-project effectively.

We express our sincere gratitude and heartfelt thanks to the respected Principal **Dr. K. BOMMANNARAJA Ph.D.,** for his encouragement and support to complete the mini-project.

We would like to express our profound interest and sincere gratitude to **Dr. P.C. SENTHIL MAHESH M.E., Ph.D.,** Head of the Department of Computer Science and Engineering for his encouragement and support to complete the mini-project.

We would like to give our sincere gratitude and heartfelt thanks to our supervisors **Mr. S.L.SWARNA M.Tech.,** Department of Computer Science and Engineering, who gave guidance and support throughout my work and made this as a successful mini-project.

We are privileged to express our deep sense of gratitude to the Project Coordinator **Mr. S PRAVEEN KUMAR M. Tech,** Assistant professor, Department of Computer Science and Engineering for guidance and support throughout my work and made this a successful mini-project.

Finally, we thank the almighty, all my staff members, parents, friends, and well-wishers for their moral support of the mini-project.

ABSTRACT

Integrated video data presentations may allow active video browsing. Such presentations provide the user with information about the content of a particular sequence being tested while maintaining an important message. We suggest how to automatically make video summaries for longer videos. Our video access method involves two tasks: first, splitting the video into smaller, compatible parts, and second, setting the levels into effects. Our proposed algorithm sections are based on an analysis of word frequency in speech transcripts. After that, the summary is made by selecting the parts with the highest scores depending on the length of time and these are illustrated. We created and conducted a user study to check the quality of the summaries made. Comparisons are made using our proposed algorithm and a random segment selection scheme based on mathematical analysis of user learning outcomes.

TABLE OF CONTENTS

TABLE OF FIGURES

CHAPTER 1

INTRODUCTION

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 the 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 is 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 producing an 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 me an opportunity to have hands-on experience with state-of-the-art NLP techniques for abstractive text summarization and implement an interesting idea suitable for intermediates and a refreshing hobby project for professionals.

CHAPTER 2
OBJECTIVE

OVERVIEW

In this project, you will be creating a Chrome Extension that will apply to the backend REST API where it will do NLP and respond with a summary of YouTube text.

1. Detail the specific features and functionality of the Chrome Extension, such as the user interface, user settings, and integration with the backend REST API.
2. Highlight the benefits of the project, such as faster and more accurate summarization of YouTube text, improved user productivity, and increased accessibility to video content for individuals with disabilities.
3. Outline the methodology used in the project, including the NLP algorithms and techniques used, data sources, and any ethical considerations or limitations of the approach.
4. Provide examples of use cases and scenarios in which the Chrome Extension can be utilized, such as in educational or professional settings, or for personal use.
5. Discuss any challenges encountered during the development and implementation of the Chrome Extension, and how they were addressed.
6. Present future directions and potential improvements for the project, such as expanding the scope of the project to support other languages or adding additional features to enhance the user experience.

CHAPTER 3

METHODOLOGY

This project basically aims at providing a clean and concise summary of the YouTube videos that the user doesn’t want to waste their time at. This project uses popular Python libraries like Flask, YouTube transcript API, Hugging-face libraries, Transformers, Speech recognition API, google translate API, Pytube FFmpeg, and many more which can be used in many real-world applications.

FLOWCHART

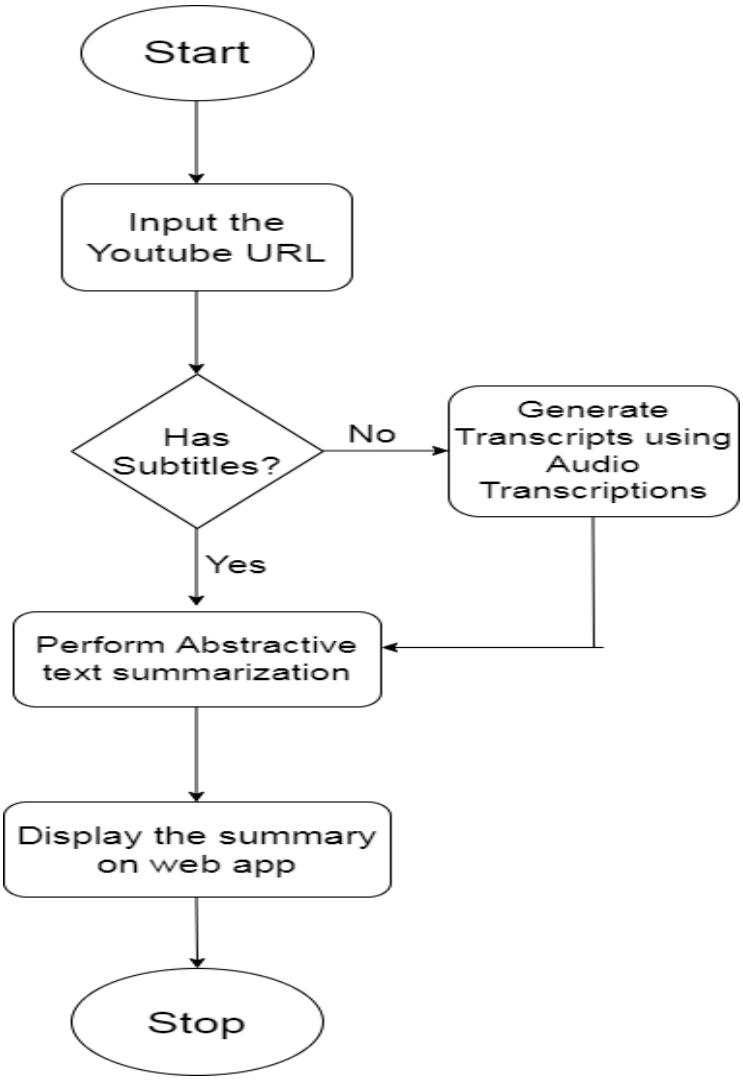


Figure-1

The project is divided into 3 modules:-

1. Input the valid YouTube video URL from the user.

The First step is to get the video link from the have a valid video id and it should be available on YouTube. These are some of the things that users should keep in mind before using this web application

Getting the transcripts from that video. After taking the video link from the user, the next part is to get the transcripts on video. Now it will check whether the given video has subtitles available or not. If the Given video has subtitles in it then we can simply use the Python library called YouTube_Transcript_Api which is used to extract the transcripts from the given video.

2. Now what if the given video doesn't have subtitles in it so in this case we will first convert the video into an audio format like .mp3 or .mp4 format using a Python library called polytube and then download that audio file. After downloading the audio file, there is a need to convert it into a .wav file using ffmpeg and then do its audio Transcription using Speech Recognition Library and Hugging Sound Library. It will basically convert that audio file into a text file that contains the Transcription of the video. This is how to get the Transcription of any video.

3. Passing the Generated transcripts to the text summarizer.

user which user wants to whole the project depends upon. This phase basically summarizes. The video should be recorded, it should

This is the project's main phase, including the text summarization. Now there are mainly two types of summarization techniques:

Extractive Summarization:- This is a text summarization technique that basically extracts the important patterns like phrases and sentences from the video, groups them together, and forms a concise summary of the YouTube video. It will not produce any new sentences. Some of them are:- TextRank, LexRank, LSA, Luhn etc.

Abstractive Summarization:- This is another text summarization technique that is a new start of the art method because it generates the sentences in a newly formed way. It will basically reproduce the sentence which is cleaner and more concise than the original sentence and in the most human-readable form. This is better than extractive summarization techniques. This can be easily implemented by using Bert Transformers or

pipeline API.

In this phase, implementation will be there of the abstractive summarization using pipeline API. Users can also change the language of the generated summary according to his/her requirements. For this purpose, googletransapi is used to do the language translation of the text. It supports 108 languages to translate.

Finally converting the whole project into a web application using Flask Framework and one can easily deploy it on cloud platforms like Heroku, AWS, etc.

CHAPTER 4
BACK-END PREPARATION

Getting started with the back-end

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

Requirements

- Create a backup application directory containing files named as app.py and needs.txt.
- Launch the app.py file with the basic Flask REST ful BoilerPlate with the tutorial link as mentioned in the reference section below.
- Create a new visible area with a pipe that will serve as a standalone (guide) where everything stays.
- Enable the newly created visual environment and apply the following dependencies using a pipe:

- Flask

- youtube_transcript_api

- converters [torch]

• Remove the cooling pipe and redirect the output to the requirements.txt file. This requirements.txt file is used to specify which Python packages are required to run a project.

Creating a Virtual Environment

- Create a project directory
- Change into the project directory
- Run python3 -m venv <name_of_virtualenv>

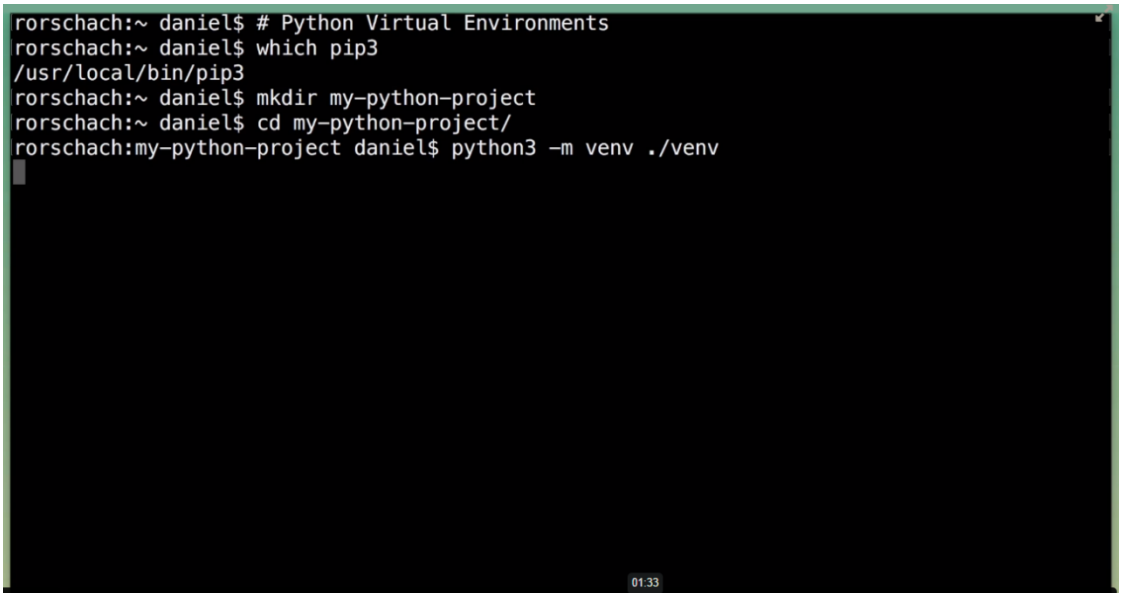


Figure 2.1

BOILERPLATE

Installation of flask

```python
from flask import Flask
from datetime

# define a variable to hold you app
app = Flask(__name__)

# define your resource endpoints
app.route('/')
def index_page():
    return "Hello world"

app.route('/time', methods=['GET'])
def get_time():
    return str(datetime.datetime.now())

# server the app when this file is run
if __name__ == '__main__':
    app.run()
```

Figure 2.2

TRANSFORMER (Hugging-Face)

Installation of Transformer from Hugging face Community.

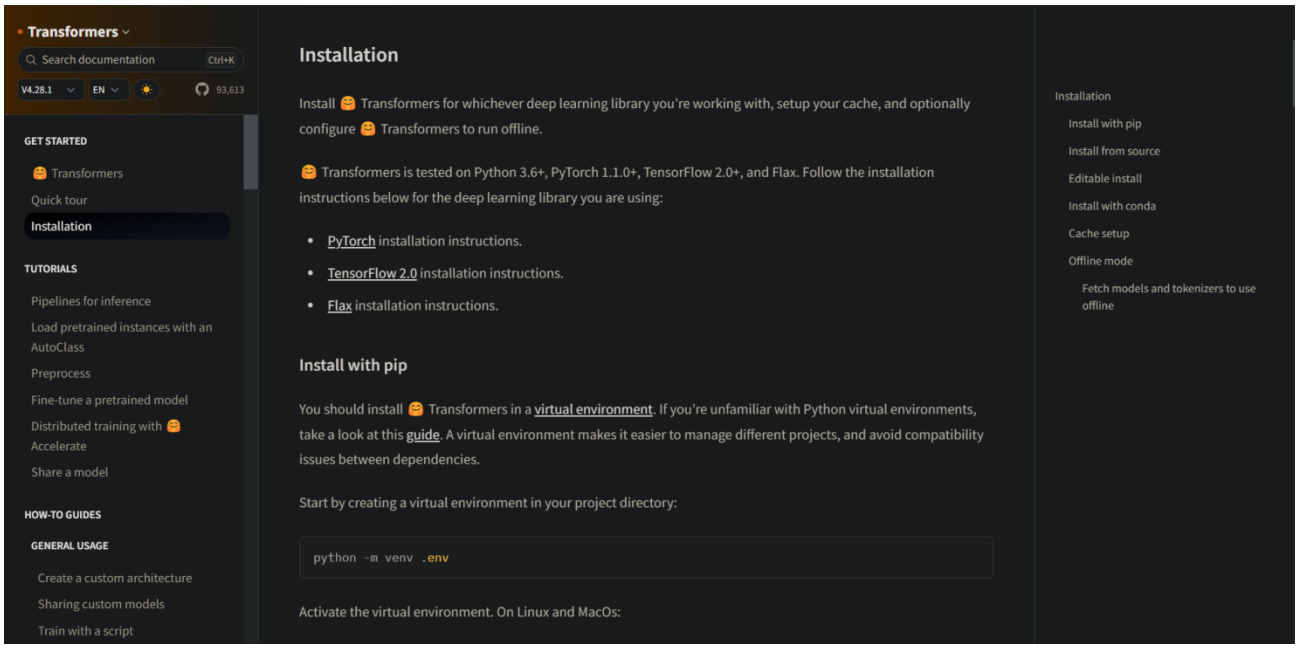


Figure2.3

Expected Outcome

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

CHAPTER 5
TRANSCRIPT SUMMARIZATION

Get transcript for a given video

Have you ever wondered how to find the contents of your YouTube video? In this milestone, we are.

You will use the python API which allows you to access text / video footage provided by YouTube. It also works with auto-generated subtitles, supports subtitle translation, and does not require a headless browser like other solutions designed for Selenium!

Requirements

In app.py, - Create a function that will accept YouTube video id as the input parameter and retrieve the full transmitted text as output. - Feedback from the Transcript API will return a list of dictionaries that look like this:

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


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

Figure 5.1

- Combine data from feedback to retrieve text in all cable formats that look like this:

  Hello, how are you ...

INSTALLATION OF API DOCUMENTATION

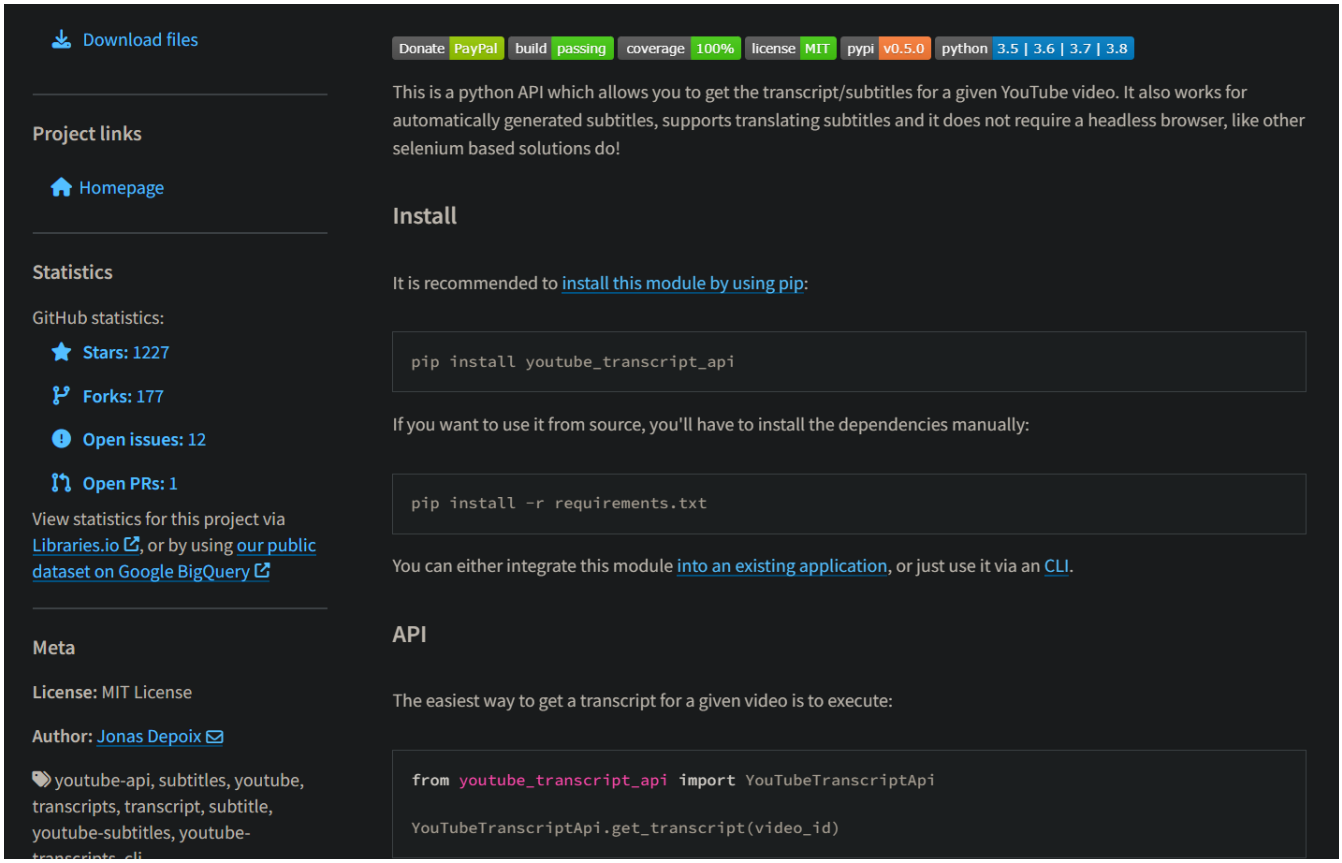


Figure 5.2

READ, WRITE, AND PARSE JSON IN Python

Figure 5.3

Expected Outcome

You should be able to download the script with the help of the work done which we will use later as the installation of the NLP processor feed into the pipeline.

CHAPTER 6

TEXT SUMMARIZATION

Perform Text Summarization

Text summarizing is the task of reducing text fragments into a concise summary that stores the content of key information and full meaning.

There are two different methods used to summarize the text:

- **Exclusive Summary**: This is where the model identifies key sentences and phrases from the original text and excludes only those.

- **Abstractive Abbreviation**: The model produces a completely different text shorter than the original, forming new sentences in a new way, like humans. In this project, we will use transformers in this way.

In this archive, we will use the HuggingFace library in Python to create an abstract text that was not found in the previous class.

Requirements

Polytube

Using pipeline API

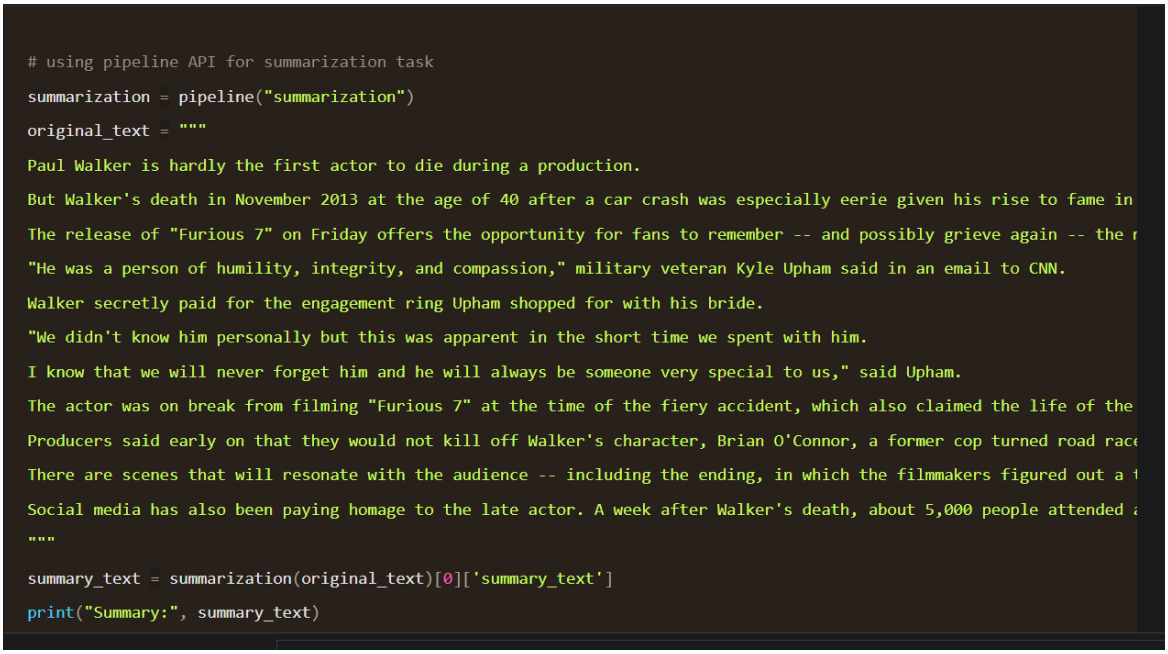


Figure 6.1

Transformer Documentation

Figure 6.2

Note

• The Transformer model used for the above project can only take input text size up to 1024 words. Therefore, text size with more than 1024 words can throw Exception in relation to the length of the text you have been transferred to.

Expected Outcome

You should be able to ensure that the model produces a completely new abstract text that is different from the original text.

CHAPTER 7

REST API ENDPOINT

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 a single endpoint, so our only resource will be the summarized text.

Requirements

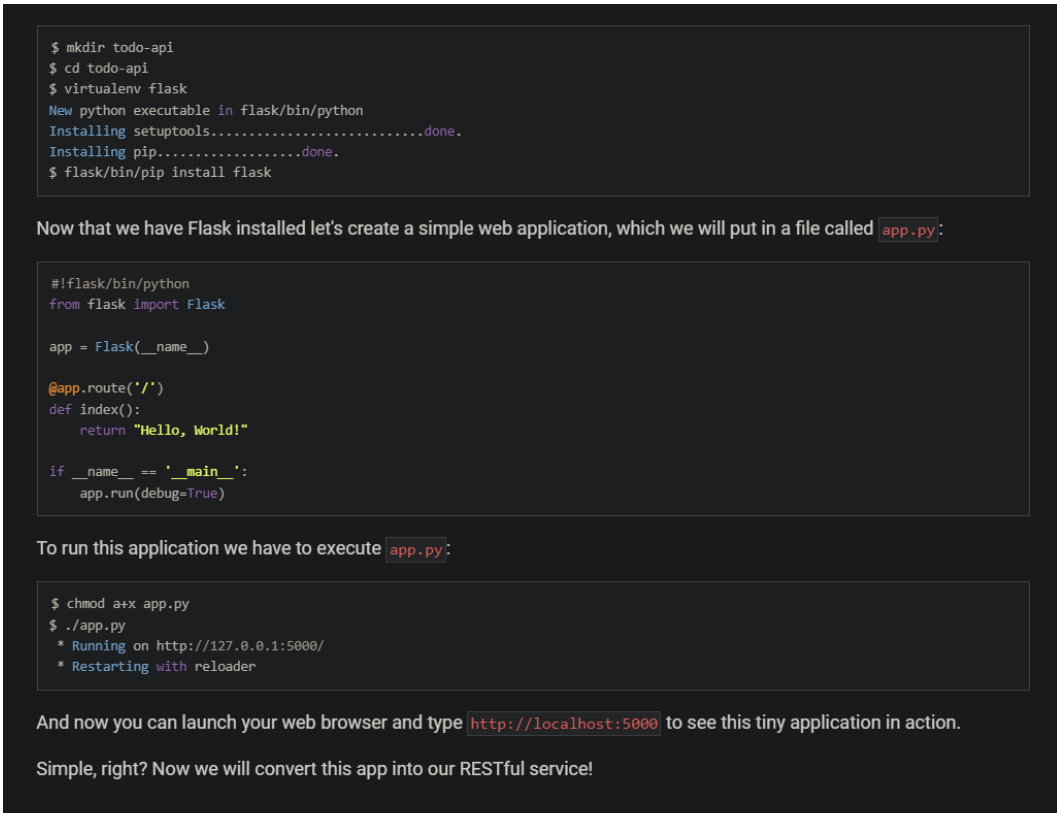
In app.py,

- Create a Flask API Route via GET HTTP Request with URI http: // [hostname] / api / summary? youtube_url = <url>.
- Extract the YouTube video id from the YouTube URL found in the query parameters. - Generate summaries by making notes the production function follows the execution of the written summary function.

- Return a summary of the HTTP status OK and manage the HTTP variant if applicable.
- Launch the Flask app and check the storage location in Postman to confirm the appropriate results.

Designing a RESTful API with Python and Flask

- **Client-Server**: There should be a separation between the server that offers a service, and the client that consumes it.
- **Stateless**: Each request from a client must contain all the information required by the server to carry out the request. In other words, the server cannot store the information provided by the client in one request and use it in another request.
- **Cacheable**: The server must indicate to the client if requests can be cached or not.
- **Layered System**: Communication between a client and a server should be standardized in such a way that allows intermediaries to respond to requests instead of the end server, without the client having to do anything different.
- **Uniform Interface**: The method of communication between a client and a server must be uniform.
- **Code on demand**: Servers can provide executable code or scripts for clients to execute in their context. This constraint is the only one that is optional.



```
$ mkdir todo-api
$ cd todo-api
$ virtualenv flask
New python executable in flask/bin/python
Installing setuptools.............................done.
Installing pip...................done.
$ flask/bin/pip install flask
```

Now that we have Flask installed let's create a simple web application, which we will put in a file called `app.py`:

```
#!flask/bin/python
from flask import Flask

app = Flask(__name__)

@app.route('/')
def index():
    return "Hello, World!"

if __name__ == '__main__':
    app.run(debug=True)
```

To run this application we have to execute `app.py`:

```
$ chmod a+x app.py
$ ./app.py
 * Running on http://127.0.0.1:5000/
 * Restarting with reloader
```

And now you can launch your web browser and type `http://localhost:5000` to see this tiny application in action.

Simple, right? Now we will convert this app into our RESTful service!

Figure 7.1

Parsing REST API Payload and Query Parameters With Flask.

For a very long time, Flask was my first choice of framework when it came to building micro-services (until Python 3.6 and async/await came to life). I have used Flask with a different number of extensions depending on the situation. flask-restful, flask-rest plus, web args .. etc. All of these extensions were great until I needed to parse the request payload or query parameters, it gets painful and the reason is they are all built on top of Marshmallow.

Marshmallow

Marshmallow is probably the oldest Python package for object serialization and parsing and probably most of the Flask extensions are built on top of it.

here is an example of marshmallow

```
from datetime import date
from pprint import pprint

from marshmallow import Schema, fields


class ArtistSchema(Schema):
    name = fields.Str()


class AlbumSchema(Schema):
    title = fields.Str()
    release_date = fields.Date()
    artist = fields.Nested(ArtistSchema())


bowie = dict(name="David Bowie")
album = dict(artist=bowie, title="Hunky Dory",
release_date=date(1971, 12, 17))

schema = AlbumSchema()
result = schema.dump(album)
pprint(result, indent=2)
# { 'artist': {'name': 'David Bowie'},
#   'release_date': '1971-12-17',
#   'title': 'Hunky Dory'
```

Figure 7.2

Expected Outcome

You should be able to create a final point to summarize YouTube video documents and test the response with different video URLs.

CHAPTER 8

CHROME EXTENSION

Getting Started with Chrome Extension

Extensions are small software programs that customize the browsing experience. Enables users to customize Chrome functionality and behavior preferences. See built on web

technologies such as HTML, CSS and JavaScript. In this epic, we are you will see how to create a recommended Chrome application guide and configure it to work with the required files.

Requirements

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

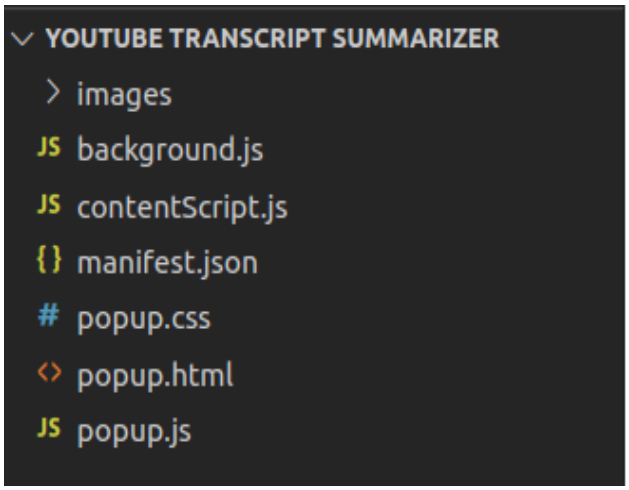


Figure8.1

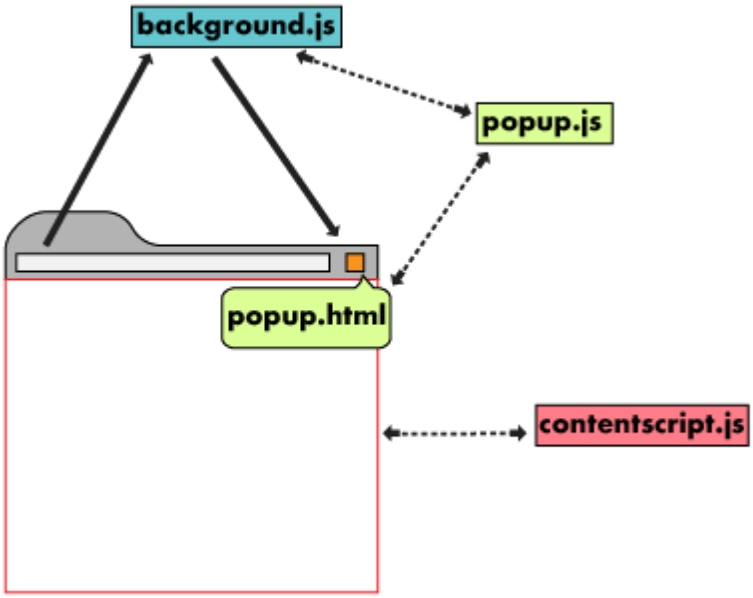


Figure8.2

- The diagram below shows a brief contribution of each chrome extension design file. (Photo source: Coding in Simple English - Medium)

- Paste the following code snippet in the manifest.json.

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


}
```

Figure 8.3

What did you guess? We already have enough to load our extension in the browser:

- Just go to chrome: // extensions and open the developer mode from the top right corner.
- Then click on Upload without downloading and select the folder containing the newly created expression file.
- When you have it, our extension is active.

Note

You will need to reload the extension every time we make changes to the extension.

Expected Outcome

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

CHAPTER 9

EXTENSION POPUP

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.

Requirements

- Insert a line at the bottom of the page action in the expression file that enables the user's evolutionary interface.

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

Figure 9.1

- In a popup.html file,

- Insert popup.css file to make styles available in HTML elements.

- Insert a popup.js file to enable user interaction and behavior with HTML objects.

- Insert a button object with an abbreviated name that when clicked will pull out a click event that the listener of the event will get to respond to.
- Enter a div item where the summary text will be displayed when received from the backend REST API Call.

• In a popup.css file,

- Provide the appropriate CSS style on the HTML and div elements button for a better user experience.

HTML & JAVASCRIPT

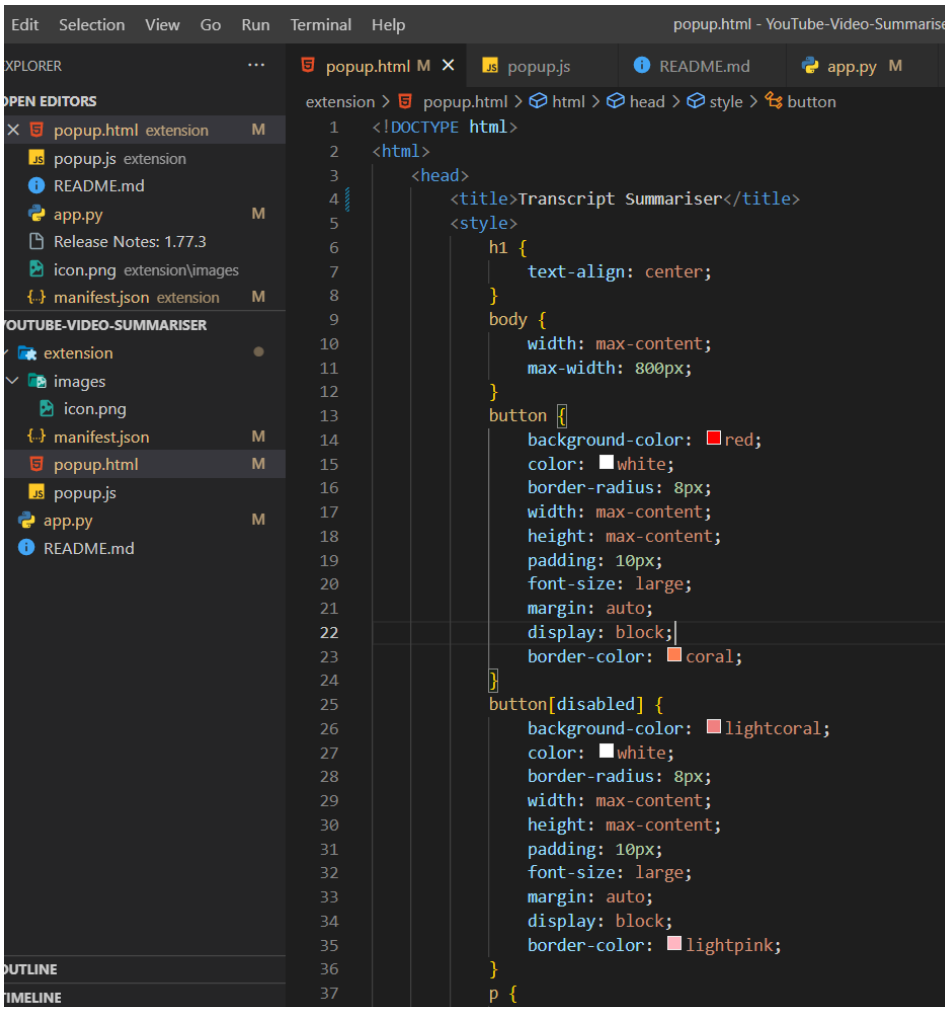


Figure 9.2

EXTENSION for Chrome

- Open the Chrome browser

- Open settings and choose the extension
- Turn on “Developer Mode”
- Load the Unpacked
- Use the Extension.

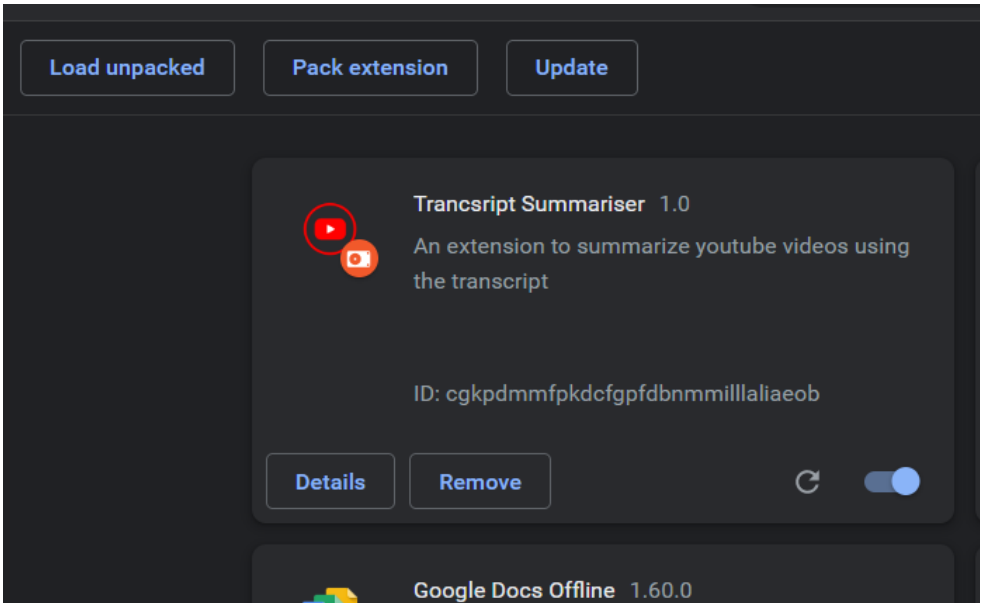


Figure 9.3

Expected Outcome

The visual interface of the user should be objective and minimal and should improve the browsing experience without interruption to it.

CHAPTER 10

SUMMARIZED TRANSCRIPT

Display Summarized Transcript

We have provided a basic UI to enable users to share and display a summary text but there are missing links that need to be fixed. In this archive, we will add functionality to allow the extension to connect to the recurring server using HTTP REST API calls.

Requirements

In popup.js,

- When the DOM is ready, paste the event listener by event type as "click" on the shortcut key and pass the second parameter as an anonymous call back function.

- For anonymous operation, send the action message generating using chrome. Run time.

Send Message method to introduce the contentScript.js to create a summary.

- Enter the event listener chrome. Run time On Message to listen to the message from contentScript.js that will execute the retrieval function to extract Summary.

- In the call back function, display a summary of the div item systematically using JavaScript.

Underline the content script in the expression file that will include excess script scripting scriptSj.js and create an automatic script on a particular page

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

```
 .
 .
 .
}
```

Figure 10.1

●In contentScript.js,

- Enter the event listener. Run time. On Message to listen a message generates that will issue a function of generational Summary call back.

- In reverse operation, extract the URL of the current tab and apply GET HTTP using the XML HTTP Request Web API back to get a summary text in response.

  - Send action messages with short uploads using Chrome run time send Message to notifypopup.js to display a summary text.

CONTENT SCRIPT

Content scripts live in an isolated world, allowing a content script to makes changes to its JavaScript environment without conflicting with the page or additional content scripts.

An extension may run in a web page with code similar to the example below.

```
<html>
  <button id="mybutton">click me</button>
  <script>
    var greeting = "hello, ";
    var button = document.getElementById("mybutton");
    button.person_name = "Bob";
    button.addEventListener("click", function() {
      alert(greeting + button.person_name + ".");
    }, false);
  </script>
</html>
```

Figure 10.2

MESSAGE PASSING IN CHROME

Since content scripts run in the context of a web page and not the extension, they often need some way of communicating with the rest of the extension. For example, an RSS reader extension might use content scripts to detect the presence of an RSS feed on a page, then notify the background page in order to display a page action icon for that page.

Communication between extensions and their content scripts works by using message passing. Either side can listen for messages sent from the other end, and respond on the same channel. A message can contain any valid JSON object (null, boolean, number, string, array, or object). There is a simple API for one-time requests and a more complex API that allows you to have long-lived connections for exchanging multiple messages with a shared

context. It is also possible to send a message to another extension if you know its ID, which is covered in the cross-extension messages section.

Using XML Http Request

To send an HTTP request, create an XML Http Request object, open a URL, and send the request. After the transaction completes, the object will contain useful information such as the response body and the HTTP status of the result.

```
function reqListener() {
  console.log(this.responseText);
}

const req = new XMLHttpRequest();
req.addEventListener("load", reqListener);
req.open("GET", "http://www.example.org/example.txt");
req.send();
```

Figure 10.3

TRANSCRIPT OUTCOME

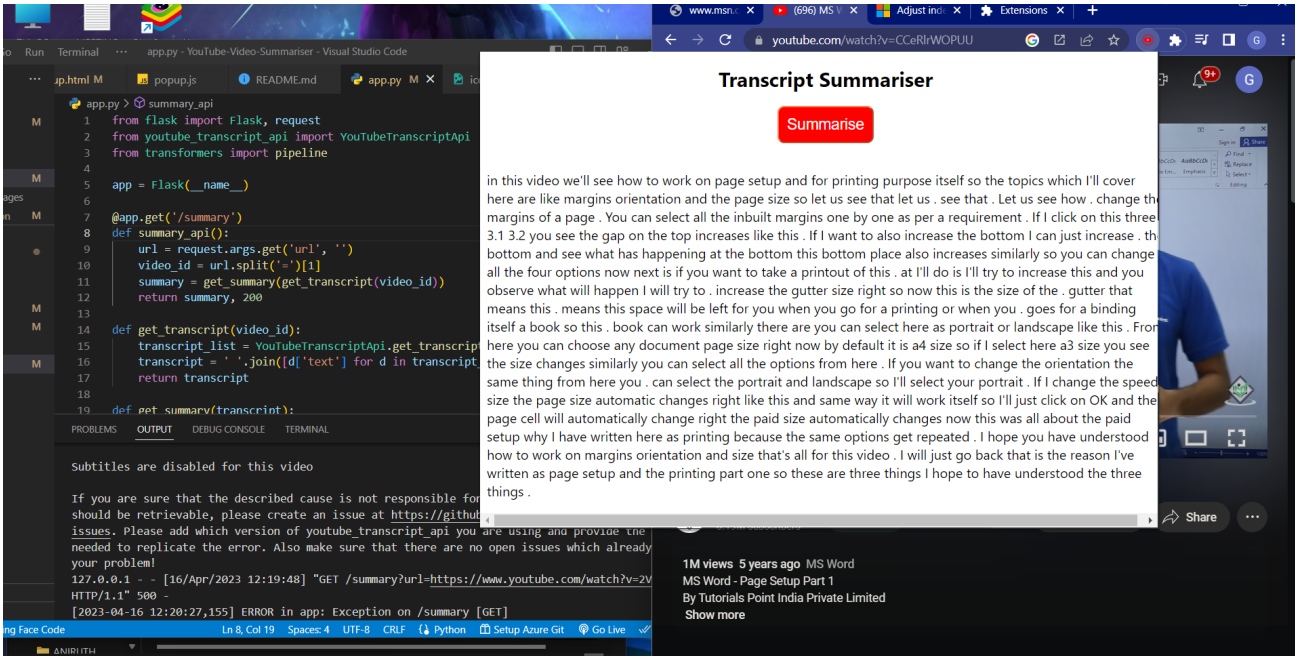


Figure 10.4

Expected Outcome

The visual interface of the user should be able to display a summary text at the request of the user.

Other Approaches We Came Around While The Exploring Ours

There are also some video browsing approaches which may be used to visualize video content compactly and hence may be considered video summarization techniques. Taskiran et al.8 cluster key frames extracted from shots using color, edge, and texture features and present them in a hierarchical fashion using a similarity pyramid. In their BMoViES system Vasconcelos and Lippman7 derive four semantic attributes from each shot and present these in a time line for the user to examine. In the CueVideo system, Srinivasan et al.3 provide a video browser with multiple synchronized views. It allows switching between different views, such as storyboards, salient animations, slide shows with fast or slow audio, and full video while preserving the corresponding point within the video between all different views. Ponceleon and Dieberger22 propose a grid, which they call the movie DNA, whose cells indicate the presence or absence of a feature of interest in a particular video segment. When the user moves the mouse over a cell, a window shows a representative

frame and other metadata about that particular cluster. A system to build a hierarchical representation of video content is discussed in Huang et al.23 where audio, video, and text content are fused to obtain an index table for broadcast news.

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 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. Overall, the project appears to be focused on developing a tool that can automate the process of summarizing YouTube video content using advanced techniques from the fields of natural language processing and machine learning. Such a tool could be useful for individuals who want to quickly and efficiently extract important information from videos without having to watch them in their entirety.

REFERENCES

1. Hank Liao, Erik McDermott, Andrew Senior“Large scale deep neural network acoustic modeling with semi-supervised training data for YouTube video transcription” December 2013.
2. Gaurav Sharma, Shaba Parveen Khan, Shivanshu

   Sharma, Syed Ubed Ali “Summarizer For Easy

   Video Assessment”

   Volume: 3 Issue:04-April-2021
3. Atluri Naga, Laggisetti Valli, JahnaviDuvru “Video

   Transcript Summarizer”

   Issue: 11 March 2022
4. Krishna Kulkarni, RushikeshPadaki “Video-Based Transcript Summarizer for Online Courses using

Natural Language Processing”

Issue: 18 December 2021

5. EvalampoisApostolidis, Eleni Adamantidou, Vasileios Mezaris “Video Summarization Using Deep Neural Networks: A Survey” Pages 1838-1863 Issue: 13 November 2021
6. G. PRIYANKA, M. PRASHA MEENA “Survey and Evaluation on Video Summarization Techniques” Issue: 28 May 2020
7. A. Workie, R. Sharma, Y. N. Chung “Digital Video Summarization Techniques” Volume 09 Issue: 01 January 2020
8. A. Dilawari, M. Usman Khan, “ASoVS: Abstractive Summarization of Video Sequences” Pages 29253-29263 Issue: 11 March 2019
9. Cüneyt M. Taskiran, Arnon Amir, Dulce B. Ponceleon, Edward J. Delp. “AutomatedVideo Summarization Using Speech Transcripts”.
10. AniquaDilawari, Muhammad usmanghani khan. “Abstractive Summarization of Video Sequences” IEEE Access, 2019.

ANNEXURE

HTML

```
<!DOCTYPE html>
<html>
   <head>
      <title>Transcript Summariser</title>
      <style>
         h1 {
            text-align: center;
         }
         body {
            width: max-content;
            max-width: 800px;
         }
         button {
```

```
            background-color: red;
            color: white;
            border-radius: 8px;
            width: max-content;
            height: max-content;
            padding: 10px;
            font-size: large;
            margin: auto;
            display: block;
            border-color: coral;
        }
        button[disabled] {
            background-color: lightcoral;
            color: white;
            border-radius: 8px;
            width: max-content;
            height: max-content;
            padding: 10px;
            font-size: large;
            margin: auto;
            display: block;
            border-color: lightpink;
        }
        p {
            font-size: medium;
        }
    </style>
  </head>
  <body>
    <h1>Transcript Summariser</h1>
    <button id="summarise" type="button">Summarise</button>
    <br/>
    <p id="output"></p>
    <script src="popup.js"></script>
  </body>
</html>
```

JAVASCRIPT

```
const btn = document.getElementById("summarise");
btn.addEventListener("click", function() {
  btn.disabled = true;
  btn.innerHTML = "Summarising...";
  chrome.tabs.query({currentWindow: true, active: true}, function(tabs){
    var url = tabs[0].url;
```

```
        var xhr = new XMLHttpRequest();
        xhr.open("GET", "http://127.0.0.1:5000/summary?url=" + url, true);
        xhr.onload = function() {
            var text = xhr.responseText;
            const p = document.getElementById("output");
            p.innerHTML = text;
            btn.disabled = false;
            btn.innerHTML = "Summarise";
        }
        xhr.send();
    });
});
```

PYTHON

```
from flask import Flask, request
from youtube_transcript_api import YouTubeTranscriptApi
from transformers import pipeline

app = Flask(__name__)

@app.get('/summary')
def summary_api():
    url = request.args.get('url', '')
    video_id = url.split('=')[1]
    summary = get_summary(get_transcript(video_id))
    return summary, 200

def get_transcript(video_id):
    transcript_list = YouTubeTranscriptApi.get_transcript(video_id)
    transcript = ' '.join([d['text'] for d in transcript_list])
    return transcript

def get_summary(transcript):
    summariser = pipeline('summarization')
    summary = ''
    for i in range(0, (len(transcript)//1000)+1):
        summary_text = summariser(transcript[i*1000:(i+1)*1000])[0]['summary_text']
        summary = summary + summary_text + ' '
    return summary

if __name__ == '__main__':
    app.run()
```

JSON

```
{
   "manifest_version": 3,
   "name": "Trancsript Summariser",
   "description": "An extension to summarize youtube videos using the transcript",
   "version": "1.0",
   "permissions": ["activeTab", "declarativeContent"],
   "host_permissions": ["http://127.0.0.1:5000/*"],

   "action": {
      "default_title": "Summarise this video",
      "default_icon": {
         "16": "images/icon.png",
         "32": "images/icon.png",
         "48": "images/icon.png",
         "128": "images/icon.png"
      },
      "default_popup": "popup.html"
   },

   "icons": {
      "16": "images/icon.png",
      "32": "images/icon.png",
      "48": "images/icon.png",
      "128": "images/icon.png"
   }
}
```