Crio.Do

YouTube Transcript Summarizer

Overview

Objective

In this project, you 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.

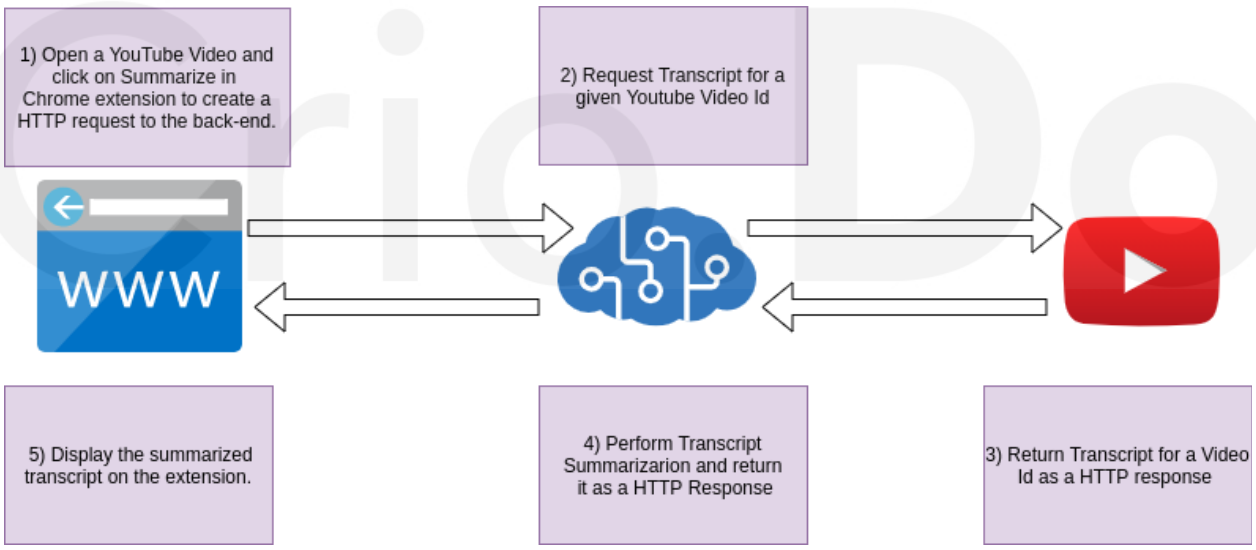
Project Context

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.

Project Stages

The project consists of the following stages:



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.

Task 1

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 back-end application directory containing files named as `app.py` and `requirements.txt`.
- Initialize `app.py` file with basic Flask RESTful BoilerPlate with the tutorial link as mentioned in the Reference Section below.

- Create a new virtual environment with `pip` installed which will act as an isolated location (a directory) where everything resides.
- Activate the newly formed virtual environment and install the following dependencies using `pip`:-
  - Flask
  - youtube_transcript_api
  - transformers[torch]
- Execute `pip freeze` and redirect the output to the `requirements.txt` file. This `requirements.txt` 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 transcript for a given video

Ever wondered how to get your YouTube video's transcripts? In this milestone, we are going to utilize a python API which allows you to get the 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 an input parameter and return parsed full transcript as output. - The response from the Transcript API will return a list of dictionaries looking somewhat like this:

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

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

- Parse the data from the response to return the transcript in whole string format looking somewhat like this:

```
Hey there how are you ...
```

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 outputs 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. In this project, we will use transformers for this approach.

In this milestone, we will use HuggingFace's transformers library in Python to perform abstractive text summarization on the transcript obtained from previous milestone.

Requirements

In `app.py`, - Create a function which will accept YouTube transcript as an input parameter and return summarized transcript as output. - Instantiate a tokenizer and a model from the checkpoint name. Summarization is usually done using an 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.

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 a single endpoint, so our only resource will be the 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 video id from the 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 exceptions 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

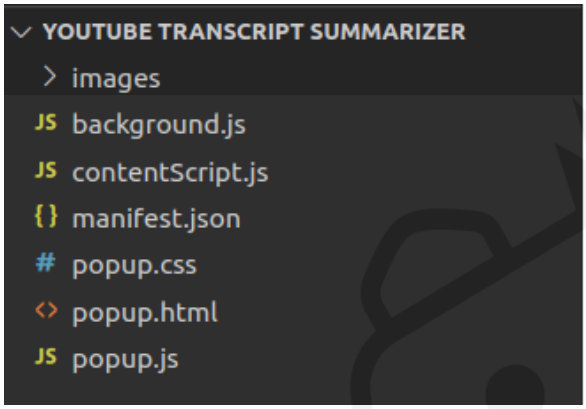
Getting Started with Chrome Extension

Extensions are small software programs that customize the browsing experience. They enable users to tailor Chrome functionality and behavior to individual preferences. They are built on web technologies such as HTML, CSS and JavaScript. In this milestone, we are

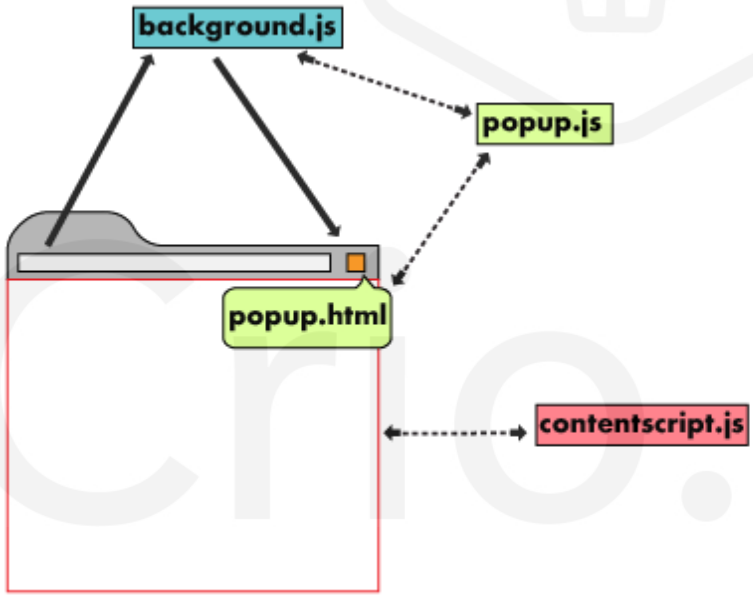
going to see how to create a recommended Chrome extension application directory and structure it to work with the required files.

Requirements

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



- The below diagram indicates the brief role of each of the files for building a chrome extension. (Image 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"],


}
```

- And guess what? We already have enough to load our extension in the browser:
  - Just go to `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 you have it, our extension is up and running.

References

- Check out Crio's HTML and CSS byte to get yourself fully equipped with HTML/CSS.
- The Ultimate Guide to Building a Chrome Extension
- How to Create Chrome Extensions

Note

- You'll need to reload the extension every time we make a change in the extension.

Expected Outcome

You 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.

Requirements

- Add the line below to `page_action` in the manifest file which enables the User Interface for a Popup.

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

- In the `popup.html` file,
  - Include the `popup.css` file to make the styles available to the HTML elements.
  - Include the `popup.js` file to enable user interaction and behavior with the HTML elements.
  - Add a `button` element named `Summarize` which when clicked will emit a click event which will be detected by an event listener to respond to it.
  - Add a `div` element where summarized text will be displayed when received from backend REST API Call.
- In `popup.css` file,
  - Provide appropriate CSS styling to the HTML elements `button` and `div` to 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 `Summarize` button and pass the second parameter as an anonymous callback function.
  - In anonymous function, send an action message `generate` using `chrome.runtime.sendMessage` method to notify `contentScript.js` to execute summary generation.
  - Add event listener `chrome.runtime.onMessage` to listen message `result` from `contentScript.js` which will execute the `outputSummary` callback function.
  - In callback function, display the summary in the `div` element programmatically using Javascript.
- Add the line below to `content_scripts` in the manifest file which will inject the content script `contentScript.js` declaratively and execute the script automatically on a particular page.

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

```
.
.
.
}
```

- In contentScript.js,

  - Add event listenerchrome.runtime.onMessageto listen messagegeneratewhich will execute thegenerateSummarycallback function.

  - In call back function, extract the URL of the current tab and make a GET HTTP request usingXMLHTTPRequestWeb API to the backend to receive summarized text as a response.

  - 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

Spice it up!

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.