**OVERVIEW:**

**Objective**

In this project, you will be a 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.

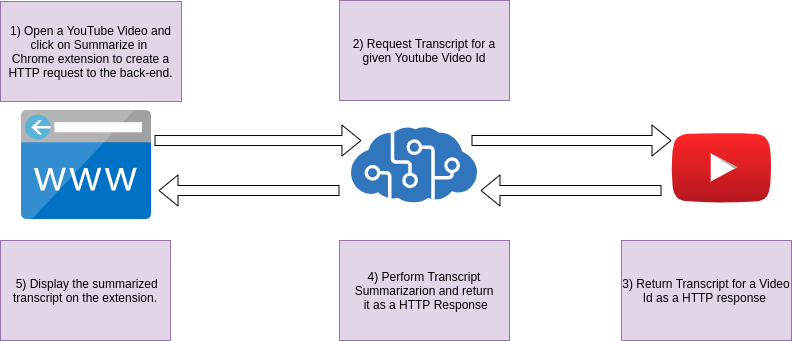
**Project Context**

Enormous number of video recordings are being created and shared on the Internet through out the day. It has become really difficult to spend time in 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 look out for the important patterns in the video and helps us to save time and efforts 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.

**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 the 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.

**1. Create and Activate a Virtual Environment:**

**For Windows:**

1. Open your command prompt and navigate to your project directory.
2. Run the following command to create a virtual environment:

bash

Copy code

python -m venv venv

1. Activate the virtual environment:

bash

Copy code

venv\Scripts\activate

**For macOS/Linux:**

1. Open your terminal and navigate to your project directory.
2. Run the following command to create a virtual environment:

bash

Copy code

python3 -m venv venv

1. Activate the virtual environment:

bash

Copy code

source venv/bin/activate

**2. Install Dependencies Using pip:**

Once the virtual environment is activated, you can install the required dependencies:

bash

Copy code

pip install Flask youtube\_transcript\_api transformers[torch]

This will install the following packages:

* **Flask**: A lightweight web framework for Python.
* **youtube\_transcript\_api**: A library to retrieve YouTube video transcripts.
* **transformers[torch]**: Hugging Face's transformers library with support for PyTorch models.

**3. Verify the Installation:**

After installing, you can verify that the dependencies were installed correctly by listing the installed packages:

bash

Copy code

pip freeze

This will display all installed packages and their versions in the virtual environment. You should see Flask, youtube\_transcript\_api, and transformers (with torch) in the list

To execute pip freeze and redirect the output to a requirements.txt file, you can use the following command after activating your virtual environment:

pip freeze > requirements.txt

**Explanation:**

* pip freeze: Lists all installed packages and their versions in the current environment.
* > requirements.txt: Redirects the output of pip freeze to a file called requirements.txt.

This requirements.txt file will contain all the packages and their exact versions, which can be used to replicate the environment on another machine by running:

pip install -r requirements.txt

This ensures that all required dependencies for the project are installed.

**References**

* [Creating a Virtual Environment in Python](https://realpython.com/lessons/creating-virtual-environment/)
* [Building RESTful APIs with Flask in Python BoilerPlate](https://atmamani.github.io/blog/building-restful-apis-with-flask-in-python/)
* [HuggingFace Transformer Python Installation](https://huggingface.co/transformers/installation.html)

**Expected Outcome**

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

**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 it 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 ...

**Steps:**

1. Install Flask and youtube\_transcript\_api (already mentioned in your previous steps).
2. Create a Flask app that receives a YouTube video ID as an input.
3. Use the youtube\_transcript\_api to fetch the transcript.
4. Parse the list of dictionaries from the response to combine the text field into one string.
5. Return the full transcript as a single string in the API response.

**Running the Application:**

1. Make sure you have Flask and youtube\_transcript\_api installed:

bash

Copy code

pip install Flask youtube\_transcript\_api

1. Run the Flask app:

bash

Copy code

python app.py

1. Access the endpoint using the browser or a tool like curl or Postman:

arduino

Copy code

http://127.0.0.1:5000/transcript?video\_id=YOUR\_VIDEO\_ID

Replace YOUR\_VIDEO\_ID with the actual YouTube video ID. The response will be a JSON object containing the parsed transcript:

json

Copy code

{

"transcript": "Hey there how are you ..."

}

**References**

* [YouTube Transcript API Documentation](https://pypi.org/project/youtube-transcript-api/)
* [Read, Write and Parse JSON using Python](https://www.geeksforgeeks.org/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.

**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](https://www.thepythoncode.com/article/text-summarization-using-huggingface-transformers-python)
* [Transformers official documentation](https://huggingface.co/transformers/task_summary.html)

**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.

**Explanation:**

1. **T5 Model for Summarization**:
   * We use T5Tokenizer and T5ForConditionalGeneration from Hugging Face's transformers library.
   * The T5 model can handle summarization tasks if provided with the appropriate prefix, summarize: .
2. **summarize\_transcript Function**:
   * Adds the summarize: prefix to the input transcript.
   * Tokenizes the input using T5Tokenizer.
   * Passes the tokenized input to the model's generate() method, which outputs the summarized text.
   * The model uses beam search (with 4 beams) to generate the summary, optimizing for more coherent results.
3. **Flask Route /summarize**:
   * Accepts a video\_id as a query parameter.
   * Fetches the full transcript using get\_transcript().
   * Summarizes the transcript using summarize\_transcript().
   * Returns the summarized version of the transcript.

**Installing Dependencies:**

Before running the app, install the necessary dependencies:

bash

Copy code

pip install Flask youtube\_transcript\_api transformers[torch]

**Example of a Summarized Transcript:**

If you pass a YouTube video ID and the transcript contains text like:

css

Copy code

Hey there how are you doing today, I hope you're having a great day ...

The summarized output might look something like:

vbnet

Copy code

Hope you're having a great day ...

**Running the Application:**

1. Start the Flask app by running:

bash

Copy code

python app.py

1. Access the endpoint using a browser, curl, or Postman:

arduino

Copy code

http://127.0.0.1:5000/summarize?video\_id=YOUR\_VIDEO\_ID

Replace YOUR\_VIDEO\_ID with the actual YouTube video ID. The response will return the summarized transcript in JSON format:

json

Copy code

{

"summary": "Summarized text of the transcript..."

}

**Notes:**

* The max\_length=150 in generate() controls the maximum length of the summary.
* You can increase or decrease max\_length and other parameters like num\_beams and length\_penalty to fine-tune the summarization output.

This setup provides a working solution for abstractive summarization of YouTube video transcripts using the T5 model! Let me know if you need further clarifications or enhancements.

**Expected Outcome**

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

**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 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](https://blog.miguelgrinberg.com/post/designing-a-restful-api-with-python-and-flask)
* [Parsing REST API Payload and Query Parameters With Flask](https://medium.com/swlh/parsing-rest-api-payload-and-query-parameters-with-flask-better-than-marshmallow-aa79c889e3ca)

**Steps:**

1. **Extract the YouTube video ID** from the URL.
2. **Generate the full transcript** using the YouTubeTranscriptApi.
3. **Summarize the transcript** using the transformers library.
4. **Return the summarized transcript** in the response with a status code of 200 OK.
5. **Handle exceptions** for invalid YouTube URLs or missing transcripts.

**Expected Outcome**

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