

Query Answering YouTube Video Seeker

A Mini Project Report Submitted by

Bhat Aditi Dinamani
(4NM20AI012)

Khushi S Bhimani
(4NM20AI021)

Krishna M S
(4NM20AI022)

UNDER THE GUIDANCE OF

Ms. Rakshitha .
Associate Professor

Department of Artificial Intelligence and Machine Learning Engineering

In partial fulfillment of the requirements for the

Natural Language Processing -
20AM603



NITTE
(Deemed to be University)

**NMAM INSTITUTE
OF TECHNOLOGY**

Nitte(DU) established under Section 3 of UGC Act 1956 | Accredited with 'A+' Grade by NAAC



08258 - 281039 – 281263, Fax: 08258 – 281265

May 2023



N.M.A.M. INSTITUTE OF TECHNOLOGY

(An Autonomous Institution affiliated to Visvesvaraya Technological University, Belagavi)

Nitte – 574 110, Karnataka, India

(ISO 9001:2015 Certified), Accredited with 'A' Grade by NAAC

☎ : 08258 - 281039 - 281263, Fax: 08258 - 281265

Department of Artificial Intelligence and Machine Learning Engineering

B.E AIML

CERTIFICATE

Certified that the mini project work entitled

“Query Answering YouTube Video Seeker”

is a bonafide work carried out by

Bhat Aditi Dinamani
(4NM20AI012)

Kfushi S Bhimani
(4NM20AI021)

Krishna M S
(4NM20AI022)

in partial fulfilment of the requirements for the award of

Bachelor of Engineering Degree in Artificial Intelligence and Machine Learning Engineering

prescribed by Visvesvaraya Technological University, Belgaum

during the year 2022-2023.

It is certified that all corrections/suggestions indicated for Internal Assessment have been

incorporated in the report deposited in the departmental library.

The mini project report has been approved as it satisfies the academic requirements in respect of the

mini project work prescribed for the Bachelor of Engineering Degree.

Signature of Guide

Signature of HOD

Evaluation

Name of the Examiners

Signature with Date

1. _____

2. _____

ACKNOWLEDGEMENT

We believe that our mini project will be complete only after we thank the people who have contributed to make this mini project successful.

First and foremost, our sincere thanks to our beloved principal, **Dr. Niranjan N. Chiplunkar** for giving us an opportunity to carry out our mini project work at our college and providing us with all the needed facilities.

I acknowledge the support and valuable inputs given by, **Dr. Sharada U Shenoy** the Head of the Department, Artificial Intelligence and Machine Learning Engineering, NMAMIT, Nitte

We express our deep sense of gratitude and indebtedness to our guide **Ms. Disha D. N, Assistant Professor** Artificial Intelligence and Machine Learning Engineering, for her inspiring guidance, constant encouragement, support and suggestions for improvement during the course for our mini project.

We also thank all those who have supported us throughout the entire duration of our mini project.

Finally, we thank the staff members of the Department of Artificial Intelligence and Machine Learning Engineering and all our friends for their honest opinions and suggestions throughout the course of our mini project.

Bhat Aditi Dinamani

Khushi S Bhimani

Krishna M S

TABLE OF CONTENTS

Sl. no.	Title	Page no.
	CHAPTER 1:	
1.1	Abstract	5
1.2	Introduction	6
1.3	System Requirements	7
1.3.1	Hardware Requirement	7
1.3.2	Software Requirements	7
	CHAPTER 2:	
2.1	Problem Statement	8
2.2	Proposed Solution	9
2.3	Solution Composition	11
2.4	Output	16
	CHAPTER 3:	
3.1	Conclusion	19
3.2	Application and Future Works	21
3.3	Bibliography	22

CHAPTER 1

1.1 ABSTRACT

This report delves into a comprehensive study conducted on a Natural Language Processing (NLP) project that addresses the task of locating the precise timestamp in a YouTube video where a particular topic is mentioned. The primary objective of this project is to develop a robust pipeline utilizing tokenization and a question answering approach. The dataset utilized in this study comprises transcripts extracted from YouTube videos, with varying lengths depending on the specific video under consideration. The report extensively discusses the essential pre-processing steps, various tokenization techniques employed, and the implementation details of the question answering model. By accurately identifying the relevant timestamps, the proposed solution aims to surmount the challenge of efficiently searching for specific topics within vast video content.

This project aims to provide a convenient tool for extracting and analyzing transcriptions of YouTube videos, as well as answering user-specific questions about the content. The tool utilizes the YouTube Transcript API to extract the transcript from a provided YouTube video link. The extracted transcript is then processed to concatenate all the text into a single string. Using the Hugging Face Transformers library, a question-answering pipeline is created to find answers within the concatenated transcript text. The project prompts the user to input a YouTube link and a question related to the video's content. It then outputs the extracted transcript, the answer to the question, and the corresponding timestamp in the video. Additionally, it generates a YouTube link with the specified timestamp, allowing users to directly navigate to the relevant part of the video. This tool provides a convenient way to extract information from YouTube videos and navigate to specific timestamps of interest.

1.2 INTRODUCTION

We've all been there before – scrolling through countless search results, hoping to find a solution to our problem, only to stumble upon a lengthy video that might contain the answer. Frustratingly, we're left with two options: watch the entire video or skip ahead blindly, hoping to find the relevant part.

That's where our project comes in. With our innovative solution, all you need is the link to the YouTube video you're interested in. We'll analyze the video and provide you with prior information, letting you know if your desired content is contained within it. Not only that, but we'll even take you directly to the precise timestamp where the information you're looking for can be found.

No more wasting time watching lengthy videos or playing the guessing game. Our project aims to streamline your search process, providing you with the efficiency and convenience you deserve.

The purpose of this project report is to provide a comprehensive overview of a YouTube Video Transcription and Question-Answering Tool. In today's digital age, video content, particularly on platforms like YouTube, has gained immense popularity and relevance. However, extracting meaningful information from videos can often be a time-consuming task. Therefore, the aim of this project was to develop a tool that simplifies the process of accessing video transcriptions and obtaining specific information within them.

Furthermore, the report discusses the significance and potential applications of the tool. By streamlining the process of accessing video transcriptions and extracting information, the tool can be beneficial for researchers, content creators, and individuals seeking specific information within YouTube videos. The report also highlights the tool's ability to generate a timestamped link, allowing users to navigate directly to relevant sections of the video.

Overall, this project report serves as a guide to understanding the functionality, and potential benefits of the YouTube Video Transcription and Question-Answering Tool.

1.3 SYSTEM REQUIREMENTS

1.3.1 SOFTWARE REQUIREMENTS :

- Windows 7 or above / Linux.
- Python 2.7 or above.
- Jupyter Notebook.
- Internet connection

1.3.2 HARDWARE REQUIREMENTS :

- Graphics Processing Unit (GPU).
- Intel Core i3 processor or above

CHAPTER 2

2.1 PROBLEM STATEMENT

In today's digital age, users frequently encounter the frustrating situation of finding a potentially helpful solution to their problem within a lengthy video. They are often faced with the daunting task of watching the entire video or blindly skipping ahead, hoping to stumble upon the relevant content. This inefficient and time-consuming process hinders users' ability to quickly access the information they need.

The challenge lies in developing a solution that can analyze YouTube videos and provide users with prior knowledge about the presence of specific content within the video. Additionally, this solution should enable users to navigate directly to the precise timestamp where the desired information is located. By addressing this problem, users will save valuable time and effort, enhancing their overall experience when seeking solutions through video content.

2.2 PROPOSED SOLUTION

The proposed solution is to design and implement a system that can effectively analyze YouTube videos, predict the presence of desired content, and provide users with the ability to access the relevant timestamps directly. The successful development of this solution will alleviate the frustration experienced by users, revolutionizing the way they interact with video content and significantly improving their search efficiency.

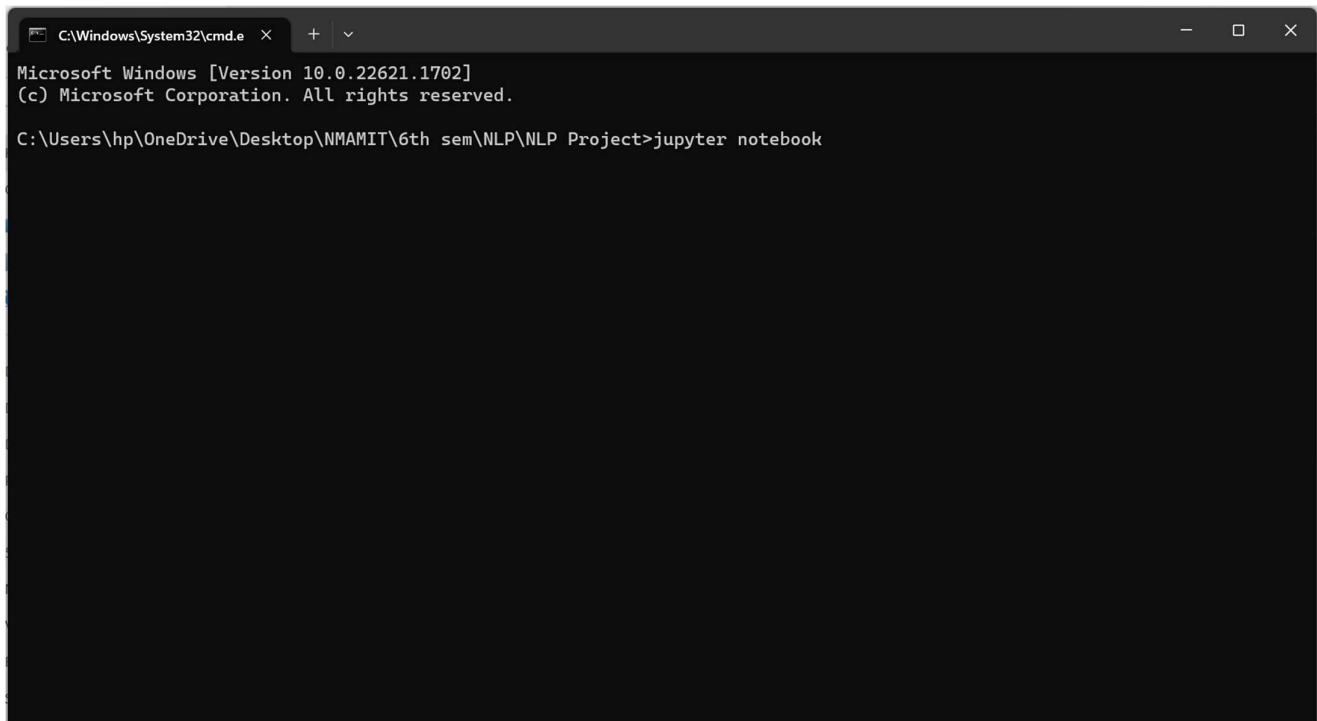
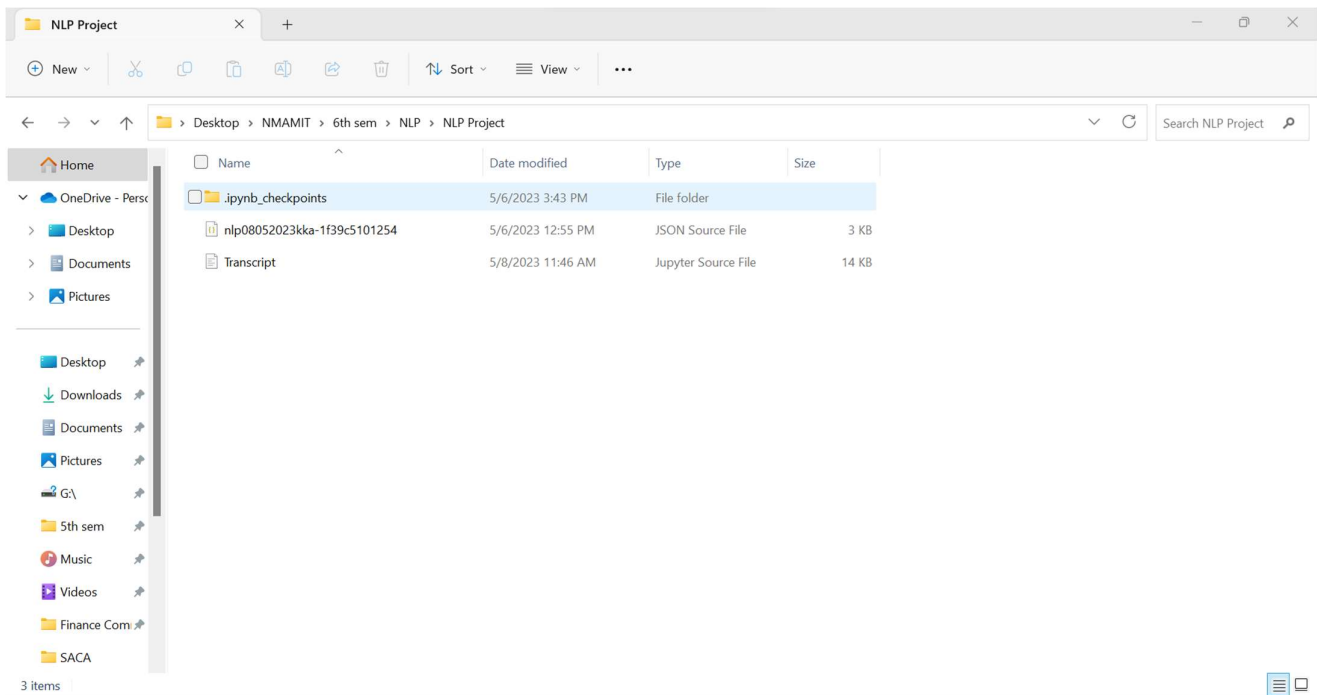
The code aims to provide a solution for extracting the transcript of a YouTube video, answering user questions about the video content, and generating a timestamped link to the video based on the provided answer.

To utilize the code, the following steps should be followed:

1. The user is prompted to enter a YouTube link. The link should be in the format:
`https://www.youtube.com/watch?v=[VIDEO_ID]`.
2. The video ID is extracted from the YouTube link.
3. The code retrieves the transcript of the YouTube video using the YouTubeTranscriptApi library.
4. The transcript is printed to the console.
5. The code creates a list to store the text from the transcript.
6. It extracts the 'text' field from each caption in the transcript and appends it to the 'text' list.

7. The 'question-answering' pipeline from the transformers library is imported.
8. The code prompts the user to input a question.
9. The question-answering pipeline is used to find the answer to the question in the concatenated text from the transcript.
10. The result, which contains the question and the corresponding answer, is printed to the console.
11. The code iterates over the captions in the transcript.
12. It checks if the answer found in step 9 is present in the text of the current caption.
13. If a match is found, the start time of the caption is printed.
14. The time in seconds is extracted from the caption.
15. The extracted time is converted into hours, minutes, and seconds.
16. The duration is formatted as "hh:mm:ss" text.
17. The formatted duration is printed to the console.
18. The desired timestamp is set based on the duration.
19. The YouTube link with the specified timestamp is generated.
20. The generated link is printed to the console.

2.3 SOLUTION COMPOSITION



Home Page - Select or create a notebook x +

localhost:8888/tree

jupyter

Quit Logout

Files Running Clusters

Select items to perform actions on them.

Upload New

<input type="checkbox"/>	0	/	Name	Last Modified	File size
<input type="checkbox"/>			Transcript.ipynb	10 days ago	14.2 kB
<input type="checkbox"/>			nlp08052023kka-1f39c5101254.json	12 days ago	2.31 kB

Home Page - Select or create a notebook x Transcript - Jupyter Notebook x +

localhost:8888/notebooks/Transcript.ipynb

jupyter Transcript (unsaved changes)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted Python 3 (ipykernel)

+ - Undo Redo Copy Paste Run Stop Code

```
In [ ]: from youtube_transcript_api import YouTubeTranscriptApi as YTA
```

```
In [ ]: link=input("YouTube Link: ")
```

```
In [ ]: videoid = link.split("v=")[1]
```

```
In [ ]: cap=YTA.get_transcript(videoid)
print(cap)
```

```
In [ ]: text=[]
for i in range(len(cap)):
    text.append(cap[i]['text'])
```

```
In [ ]: text
```

```
In [ ]: from transformers import pipeline
```

```
In [ ]: question_answerer = pipeline("question-answering")
```

```
In [ ]: sent=""
for a in text:
    sent+=a
sent
```

```
In [ ]: quest= input("Question: ")
```

```

In [ ]: result = question_answerer(question=quest, context=sent)

In [ ]: result

In [ ]: for i in range(len(cap)):
        if result['answer'] in cap[i]['text']:
            print(cap[i]['start'])
            time=int(cap[i]['start'])
            break
        break

In [ ]: # calculate the hours, minutes, and seconds
hours = int(time// 3600)
minutes = int((time % 3600) // 60)
seconds = int(time % 60)

# format the duration as "hh:mm:ss" text
duration_text = f"{hours:02d}h{minutes:02d}m{seconds:02d}s"

# print the duration text
print(duration_text)

In [ ]: timestamp =duration_text # replace with the desired timestamp in the format "XmYs" (X = minutes, Y = seconds)

# generate the YouTube Link with the specified timestamp
youtube_link = f"https://www.youtube.com/watch?v={videoid}&t={timestamp}"

# print the generated link
print(youtube_link)

```

Code

```

from youtube_transcript_api import YouTubeTranscriptApi as YTA

# Prompt the user to input a YouTube link
link = input("YouTube Link: ")

# Extract the video ID from the link
videoid = link.split("v=")[1]

# Get the transcript of the YouTube video using the extracted video ID
cap = YTA.get_transcript(videoid)

# Print the transcript
print(cap)

# Create an empty list to store the text from the transcript
text = []

```

```

# Extract the 'text' field from each caption and append it to the 'text'
list
for i in range(len(cap)):
    text.append(cap[i]['text'])

# Print the 'text' list
print(text)

# Import the required pipeline from the transformers library
from transformers import pipeline

# Create a question-answering pipeline
question_answerer = pipeline("question-answering")

# Concatenate all the text from the transcript into a single string
sent = ""
for a in text:
    sent += a

# Prompt the user to input a question
quest = input("Question: ")

# Use the question-answering pipeline to find the answer in the
concatenated text
result = question_answerer(question=quest, context=sent)

# Print the result
print(result)

# Iterate over the transcript captions
for i in range(len(cap)):
    # Check if the answer is present in the current caption's text
    if result['answer'] in cap[i]['text']:
        # Print the start time of the caption
        print(cap[i]['start'])
        # Extract the time in seconds from the caption
        time = int(cap[i]['start'])
        break # Exit the loop after finding the first matching caption

# Calculate the hours, minutes, and seconds from the extracted time in
seconds
hours = int(time // 3600)
minutes = int((time % 3600) // 60)
seconds = int(time % 60)

# Format the duration as "hh:mm:ss" text
duration_text = f"{hours:02d}h{minutes:02d}m{seconds:02d}s"

```

```
# Print the duration text
print(duration_text)

# Set the desired timestamp
timestamp = duration_text # replace with the desired timestamp in the
format "XmYs" (X = minutes, Y = seconds)

# Generate the YouTube link with the specified timestamp
youtube_link = f"https://www.youtube.com/watch?v={videoid}&t={timestamp}"

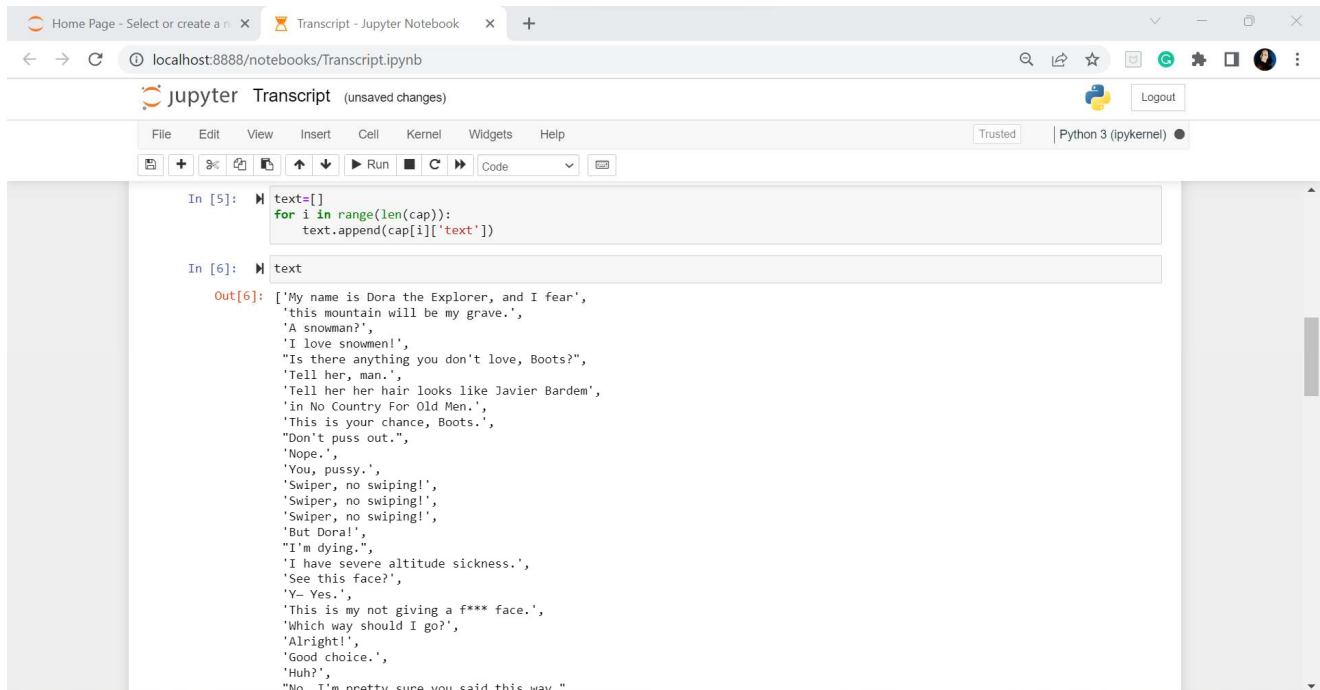
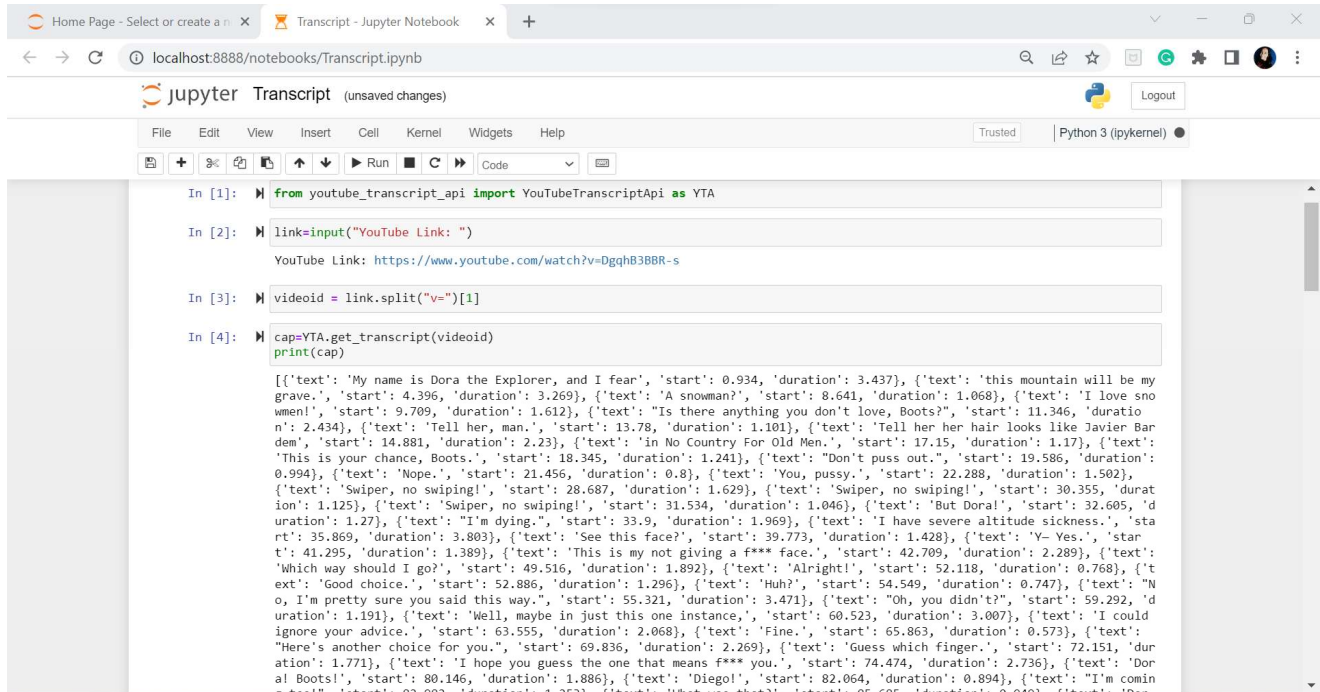
# Print the generated link
print(youtube_link)
```

The provided code utilizes the `youtube_transcript_api` library to retrieve the transcript of a YouTube video. The library provides a convenient way to access the transcript data associated with a specific video ID. By extracting the video ID from the YouTube link provided by the user, the code calls the `get_transcript` function from the `YouTubeTranscriptApi` class to retrieve the transcript.

Once the transcript is obtained, it is stored in the variable `cap`. The code then extracts the text from each caption in the transcript and appends it to the text list. This allows for easy manipulation and analysis of the transcript content. To perform question-answering, the code utilizes the `transformers` library, which provides a powerful set of tools for natural language processing (NLP) tasks. Specifically, the code imports the `pipeline` function from `transformers`. The pipeline function allows for easy access to pre-trained models for various NLP tasks, including question-answering.

After concatenating all the text from the transcript into a single string, the code prompts the user to input a question. It then uses the question-answering pipeline, initialized with the question-answering task, to find the answer within the concatenated text. The `question_answerer` pipeline object takes the question as input and searches for the answer within the provided context (i.e., the concatenated transcript text). The result, containing the question and the corresponding answer, is stored in the `result` variable.

2.4 OUTPUT




```

In [7]: from transformers import pipeline

In [8]: question_answerer = pipeline("question-answering")

No model was supplied, defaulted to distilbert-base-cased-distilled-squad and revision 626af31 (https://huggingface.co/distilbert-base-cased-distilled-squad).
Using a pipeline without specifying a model name and revision in production is not recommended.
All model checkpoint layers were used when initializing TFDistilBertForQuestionAnswering.

All the layers of TFDistilBertForQuestionAnswering were initialized from the model checkpoint at distilbert-base-cased-distilled-squad.
If your task is similar to the task the model of the checkpoint was trained on, you can already use TFDistilBertForQuestionAnswering for predictions without further training.

In [9]: sent=""
        for a in text:
            sent+=a
        sent

Out[9]: "My name is Dora the Explorer, and I fear this mountain will be my grave. A snowman? I love snowmen! Is there anything you don't love, Boots? Tell her, man. Tell her her hair looks like Javier Bardem. No Country for Old Men. This is your chance, Boots. Don't puss out. Nope. You, pussy. Swiper, no swiping! Swiper, no swiping! Swiper, no swiping! But Dora! I'm dying. I have severe altitude sickness. See this face? Y- Yes. This is my not giving a f*** face. Which way should I go? Alright! Good choice. Huh? No, I'm pretty sure you said this way. Oh, you didn't? Well, maybe in just this one instance, I could ignore your advice. Fine. Here's another choice for you. Guess which finger. I hope you guess the one that means f*** you. Dora! Boots! Diego! I'm coming too! What was that? Dora! Where is your safety harness? Pay back's a bitch. Oh. I'm coming, Diego! I rescued some condors once. Are you here to return the favor? Why didn't we learn from the Crocodile Hunter? Dora! Ah, I can't hang on! It's okay, Dora. It's time for me to go, Diego, Go! To heaven! Diego! We did it. I'm an explorer. Food is food. Help me. Gotcha something. You, bitch. Reaper, no reaping. Reaper, no reaping. Reaper -Doom! Ha ha!"

```

```

In [10]: quest = input("Question: ")
         Question: Dora is a what?

In [11]: result = question_answerer(question=quest, context=sent)

In [12]: result

Out[12]: {'score': 0.7363806366920471, 'start': 1038, 'end': 1046, 'answer': 'explorer'}

In [13]: for i in range(len(cap)):
         if result['answer'] in cap[i]['text']:
             print(cap[i]['start'])
             time = int(cap[i]['start'])
             break
         break

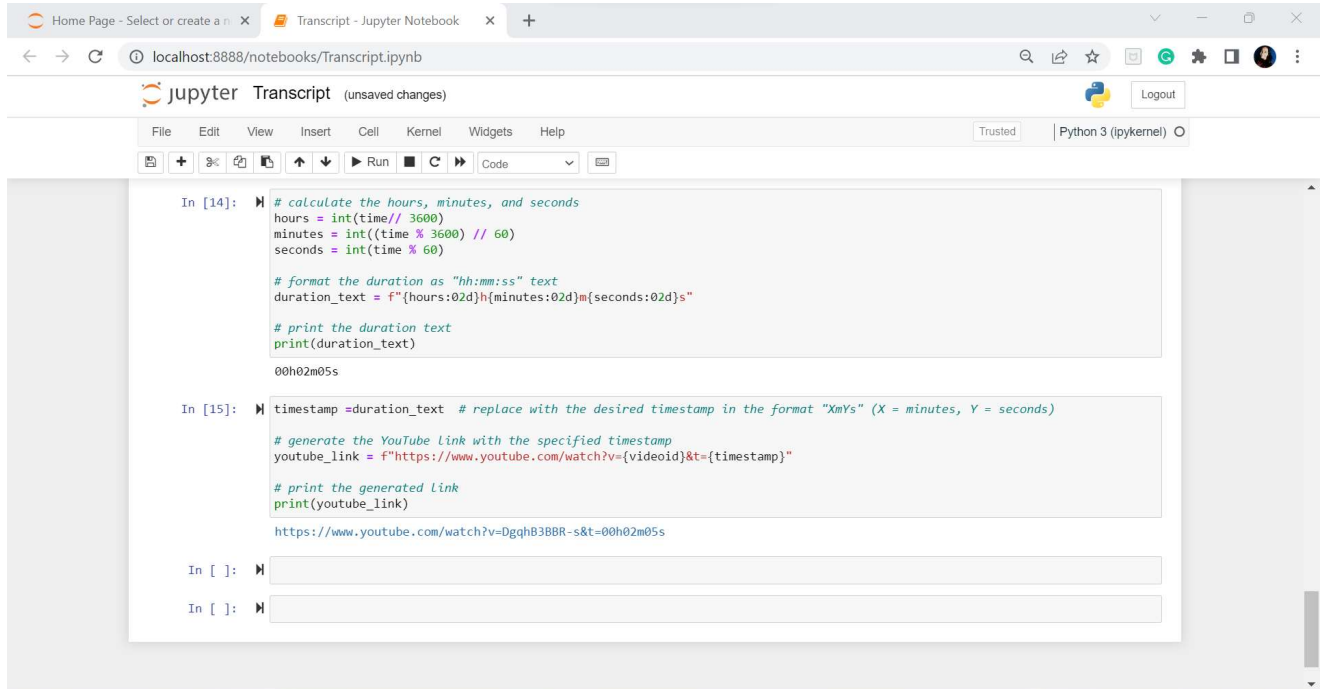
125.758

In [14]: # calculate the hours, minutes, and seconds
         hours = int(time // 3600)
         minutes = int((time % 3600) // 60)
         seconds = int(time % 60)

         # format the duration as "hh:mm:ss" text
         duration_text = f"{hours:02d}h{minutes:02d}m{seconds:02d}s"

         # print the duration text
         print(duration_text)

```



The screenshot shows a Jupyter Notebook titled "Transcript" running on a local host. The notebook contains two code cells. The first cell calculates the hours, minutes, and seconds from a given time and formats them into a string like "00h02m05s". The second cell takes this string, replaces it with a desired timestamp, and generates a YouTube link. The output of the first cell is "00h02m05s", and the output of the second cell is "https://www.youtube.com/watch?v=DgqhB3B8R-s&t=00h02m05s".

```
In [14]: # calculate the hours, minutes, and seconds
hours = int(time // 3600)
minutes = int((time % 3600) // 60)
seconds = int(time % 60)

# format the duration as "hh:mm:ss" text
duration_text = f"{hours:02d}h{minutes:02d}m{seconds:02d}s"

# print the duration text
print(duration_text)

00h02m05s

In [15]: timestamp = duration_text # replace with the desired timestamp in the format "XmYs" (X = minutes, Y = seconds)

# generate the YouTube link with the specified timestamp
youtube_link = f"https://www.youtube.com/watch?v={video_id}&t={timestamp}"

# print the generated link
print(youtube_link)

https://www.youtube.com/watch?v=DgqhB3B8R-s&t=00h02m05s

In [ ]: 
In [ ]:
```

CHAPTER 3

3.1 CONCLUSION

In conclusion, the code presented in this project provides a valuable solution for extracting transcripts from YouTube videos, answering user questions about the video content, and generating timestamped links for easy access to specific sections of the videos. By utilizing the `youtube_transcript_api` library, the code retrieves the transcript associated with a given YouTube video ID. The transformers library, specifically the question-answering pipeline, is employed to accurately answer user questions based on the transcript content.

The code's potential applications are diverse and impactful. It can be employed in video content analysis, allowing for in-depth exploration, keyword identification, and pattern recognition. Furthermore, the code enhances the interactive video browsing experience by enabling users to navigate directly to relevant moments of interest within lengthy videos. In educational settings, the code facilitates effective learning by providing precise answers to questions and offering timestamps for further comprehension. Additionally, the code can streamline transcription services by automating the generation of accurate, time-stamped transcripts for YouTube videos.

By incorporating this code into various applications, users can unlock new possibilities

for video analysis, user engagement, education, and transcription services. The provided solution demonstrates the power of leveraging existing libraries and technologies to enhance the accessibility, usability, and insights derived from YouTube video content.

Overall, this project showcases the potential of utilizing Python, along with the `youtube_transcript_api` and `transformers` libraries, to extract valuable information, facilitate interaction, and improve user experiences in the realm of YouTube video analysis and accessibility.

3.2 APPLICATION AND FUTURE WOKS

The potential applications of this code are numerous. Here are a few examples:

Video Content Analysis: The code can be used to analyze and extract meaningful information from video transcripts. It enables researchers, content creators, and marketers to perform in-depth analysis of video content, identify keywords, topics, and patterns, and gain insights for various purposes, such as content optimization, sentiment analysis, and market research.

Interactive Video Browsing: By providing the ability to ask questions about the video content and obtain precise timestamps for relevant sections, this code can enhance the browsing experience for viewers. It allows users to quickly navigate to specific topics or moments of interest within a lengthy video, improving user engagement and satisfaction.

Educational Video Platforms: This code can be integrated into educational video platforms to facilitate learning and comprehension. Students can ask questions related to the video content, and the system can provide accurate answers along with timestamps to refer to specific sections for further understanding. It can also be utilized for creating interactive quizzes based on the video content.

Transcription Services: The code can be leveraged in transcription services to automate the process of generating accurate and time-stamped transcripts for YouTube videos. It reduces the manual effort required for transcription and ensures the availability of accurate transcripts for accessibility, indexing, and search purposes.

These are just a few examples of how the code can be applied. Its versatility in extracting transcripts, answering questions, and generating timestamped links opens up possibilities for a wide range of applications in video analysis, user interaction, education, and transcription services.

3.3 BIBLIOGRAPHY

- <https://pypi.org/project/youtube-transcript-api/>