DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Rajiv Gandhi University of Knowledge Technologies – SKLM,

Etcherla, Andhra Pradesh – 521202

VIDEO TRANSCRIPT SUMMARIZATION USING NLP

A MINI PROJECT REPORT

Submitted in partial fulfillment of requirements for the award of the degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

Under the esteemed guidance of

Mr. G.Siva Rama Sastry MTech(PhD)
Project Guide,
Asst.Professor,
Computer Science and Engineering,
RGUKT SKLM.

Team Members:Team-77
Ch.Lohitha-S190441
K.Sownya-S190774
D.Bhuvaneshwari-S190864
V.NagaSuryaDurgaPrasad-S190784

ACKNOWLEDGEMENT

We would like to express our sincere gratitude and appreciation to all those who have contributed to the completion of this mini project on video Transcript summarization using NLP First and foremost, we extend our heartfelt thanks to **Mr. G.Siva Rama Sastry sir** whose guidance, support, and expertise were invaluable throughout the project. Their insights and feedback significantly enriched our understanding and guided us in the right direction.

We express our gratitude to ***Lakshmi Bala Mam (*HOD of CSE)** and other faculty members for being a source of inspiration and constant encouragement which helped us in completing the project successfully.

An endeavor over a long period can also be successful by constant effort and encouragement. We wish to take this opportunity to express our deep gratitude to all the people who have extended their cooperation in various ways during our project work. It is our pleasure to acknowledge the help of all those respected individuals.

INDEX

ABSTRACT

The increasing number of online videos makes it challenging to watch lengthy videos in their entirety, often resulting in wasted time if the necessary information is not found. By leveraging NLP, specifically Hugging Face transformers, this project automatically summarizes video transcripts, identifying key patterns and providing concise content summaries. Natural Language Processing (NLP) uses computers to examine voice and language, find contextual patterns, and generate insights from text and audio.

The project integrates multiple technologies, including the YouTube Transcript API for fetching YouTube video transcripts, Speech Recognition for converting speech to text, Google Translate API for translating text, Pydub for audio conversion, MoviePy for handling video files and extracting audio, and Streamlit for building the web interface.

This application processes local video files and YouTube videos. Users can upload a video, which is then processed to extract audio using MoviePy. The audio is segmented using Pydub, and Google Speech Recognition transcribes the segments into text. For summarization, the BART model from Hugging Face's Transformers library is used, The Google Translate API translates the summaries into languages such as Kannada, Telugu, Hindi, and Tamil.

The Streamlit interface is user-friendly, offering options to upload local videos or enter YouTube URLs. It provides real-time processing and displays transcripts, summaries, and translations. Users can customize the summary length and select the target language for translation, enhancing their overall experience. We have used the following keywords: YouTube, Text Summarization, Streamlit, Speech Recognition, Translator, NLP

CHAPTER I INTRODUCTION

1.1 Background and Context of the Project

In modern times, there is a vast quantity of videos being produced and shared on Youtube continuously.Globally,YouTube ranks as the second-highest frequented website.YouTube offers a diverse selection of content, varying from short films and music videos to feature films, documentaries, corporate sponsored movie trailers, live streams, vlogs, and other material produced by famous YouTubers. Every day, video content on YouTube is being watched collectively by its users for more than a total of one billion hours.

In 2020, there were approximately 2.3 billion people who used YouTube and the number of users has been quickly growing each year. At a rate of 300 hours of video uploaded per minute, YouTube constantly receives an immense amount of content.According to research conducted by Google, almost 33% of viewers on YouTube in India use their mobile devices to watch videos and spend more than 48 hours on the platform every month. youtube is the primary source for each and every student where they can learn new concept and can do the self study.But Watching such lengthy videos has become challenging because it is possible to waste time without finding the desired information as our efforts may be unproductive if we fail to retrieve the relevant information we seek.Searching for videos that contain the relevant content can be a tedious and exasperating process. Many videos posted online involve a speaker discussing a subject at length, yet it can prove challenging to locate the main message of the presentation without viewing the entire video. Python offers different packages that can be extremely useful. Accessing YouTube content, such as transcripts of videos, has now become more convenient with the assistance of the API in the Python library. We are able to view the video content directly and provide users with a summary by utilizing this benefit. One way to achieve this is through the application of Hugging Face transformer, a method for summarizing text. The generated summary is a result of using

the hugging face transformer package. Typically, written descriptions are used to encapsulate the content of YouTube videos rather than automation. our model proposes the usage of a transformer package for summarizing the transcripts of the video, thereby providing a meaningful and important summary of the video. Our main concern is to summarize the data, by using the pre-trained summarization techniques.

1.2 LITERATURE SURVEY:

In 2021, "Natural Language Processing (NLP) based Text Summarization - A Survey" was published by Ishitva Awasthi,Kuntal Gupta, Prajbot Singh Bhojal, Anand, Piyush kumar. The techniques used are Extractive and abstract methods for summarizing texts. The advantages are: Based on linguistic and statistical characteristics, the implications of sentences are calculated. The disadvantage is each type of summarization technique is useful in different situations. One cannot say which technique is more promising [1].

"Review of automatic text summarization techniques & methods" is developed by AdhikaPramita, SupriadiRustad, Abdul Shukur, Affandy. It was published in 2020. They have used Text summarization, Systematic review techniques. The drawback is that the Fuzzy based approach is weak in semantic problems. [2].

"Study on Abstractive Text Summarization Techniques" is developed by Parth Rajesh Dedhia, Hardik Pradeep, Meghana Naik. It was published in 2020. They have used Seq2Seq, Encoder-Decoder, and Pointer Mechanism. The drawback is that thecurrent model does not work if multiple documents are passed to the model[3].

"Abstractive Summarization of video sequences" is developed by AniqaDilawari, Muhammad Usman Ghani Khan. They have used multi-line video description, RCNN deep neural network model. The drawback is It focuses only on the conciseness of thesummary. Memory efficiency and time constraints are not under consideration[4]

1.3 Objectives and Goals of the Project

Objectives:

User Input:

- User inputs the Local Video File and YouTube video link and selects the desired summary settings.

Transcript Extraction:

- The application extracts the transcript from the YouTube video using the youtube transcript API,similarly extracts the transcript from the local videos using speech recognition.

Language Processing:

- Detects the language of the transcript.

Summary Generation:

- Generates a summary .

Optional Translation:

- Translates the summary to the selected target language according to user choice.

Display Results:

- Displays the generated summary.

Goals:

1. Efficient Information Extraction:

Enhance the extraction of key information from video content, enabling users to quickly grasp essential points without watching the entire video.

2. Multilingual Support:

Provide robust support for summarization in multiple languages to cater to a diverse user base, enhancing accessibility.

3. Improved Content Accessibility:

Contribute to improved accessibility of video content by offering quick and reliable summaries, saving users time and effort.

4. User-Friendly Application:

Develop a user-friendly application that allows users to input YouTube links and receive concise summaries in their preferred language.

5. Research Advancement:

Advance research in the field of NLP and video summarization by applying innovative techniques and improving existing methodologies.

6. Scalable Solution:

Create a scalable solution capable of handling a large volume of video content, making it practical for widespread use.

7. Enhance User Experience:

Improve the overall user experience by providing quick access to summarized content, making it easier to consume and understand large amounts of video information.

1.1 Limitations

Data Quality and Quantity:

- **Limitation:** The accuracy of video summarization models depends on the quality and quantity of available data. Insufficient or biased datasets can lead to poor model performance.
- **Impact:** Limited data may cause models to miss important content or provide inaccurate summaries.

Model Interpretability:

- **Limitation:** Complex NLP models can be difficult for non-technical stakeholders to understand.
- **Impact:** Lack of interpretability may hinder acceptance and trust in the summarization process.

Generalization to Diverse Content:

- **Limitation:** Models trained on specific datasets may not generalize well to diverse video content.
- **Impact:** Models might perform poorly on new, unseen data, limiting their applicability across various content types.

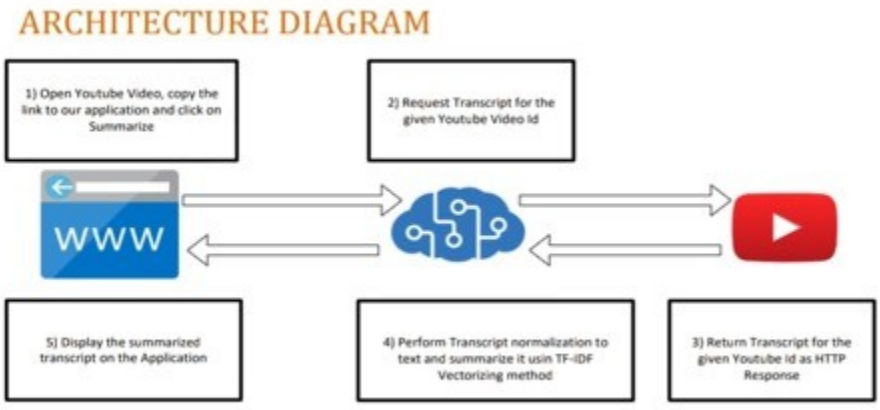
Language Support:

- **Limitation:** Summarization effectiveness varies across languages, especially if training data is predominantly in one language.
- **Impact:** Less accurate summaries for underrepresented languages reduce the tool's accessibility.

CHAPTER-2
METHODOLOGY

2.1 Methodology

In this project we get transcripts/subtitles for a given YouTube video Id using a Python API, perform text summarization on obtained transcripts using HuggingFace transformers, using a Streamlit web applications to expose the summarization service to display summarized text to the user.



figureInitially we open a YouTube video and click the button summarize in the chrome extension. This will create a HTTP request to the back-end. Then the request to access the transcripts will be made with YouTube video ID which was taken from the URL. The response will be a transcript of that video in json format. After getting the transcripts in the text format the system performs Transcript Summarization. Finally, it displays the summarized transcript on the extension.

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 our APIs .Streamlit is designed for quickly building interactive web apps for data science and machine learning and requires lessboiler plate code. We create a back-end application using Streamlit. Streamlit initializes and runs the app with 'streamlit run <script_name>.py'.

GET TRANSCRIPTS

In this module, we are going to utilize a Google Speech Recognition to process local video and extract text,using BART model to Summarize text,using Google Translate API which allows you to translate the text to a specified language. In app.py, we create a function which will accept YouTube transcript as an input parameter and return summarized transcript as output. Then instantiate a tokenizer and a model from the checkpoint name.

Summarization is done with the help of encoder-decoder models, such as Bart or T5. Then we define the transcript that should be summarized and add the BART specific prefix "summarize:". Finally use the PreTrainedModel.generate() method to generate the summary.

Display Summarized Text

A user interface is implemented to enable users to interact and display the summarized text but there are some missing links which must be addressed. In this step, we will add functionality to allow the extension to interact with the backend server using HTTP REST API Calls.

2.2 TOOLS AND TECHNOLOGIES USED:

1.Streamlit: A Python library for building interactive web applications.

2.YouTube Transcript API: Retrieves transcripts from YouTube videos.

3.Google Translate API: Translates text between languages.

4.Moviepy: It is a Python library used for video editing. It provides a wide range of tools for handling and manipulating video files.

5.**Speech_recognition**:It can capture and recognize speech from various sources convert it to text

6.**Transformers (BartTokenizer and BartForConditionalGeneration)**:
BartTokenizer is used to tokenize text, and BartForConditionalGeneration is used for tasks like text summarization, translation, and other sequence-to-sequence tasks.

These technologies together enable the application to extract, analyze, summarize, translate, and display information from YouTube videos interactively.

REQUIREMENT ANALYSIS:

A. Recommended Operating System

1) Window 7

2) Linux: Ubuntu 16.04-17.10

B. Hardware

1) Processor: Minimum 1 GHz; Recommended 2 GHz or more

2) Ethernet connection (LAN) OR a wireless adapter (Wi-Fi)

3) Hard Drive: Minimum 32 GB; Recommended 64 GB or more

4) Memory (RAM): Minimum 1 GB; Recommended 4 GB or above

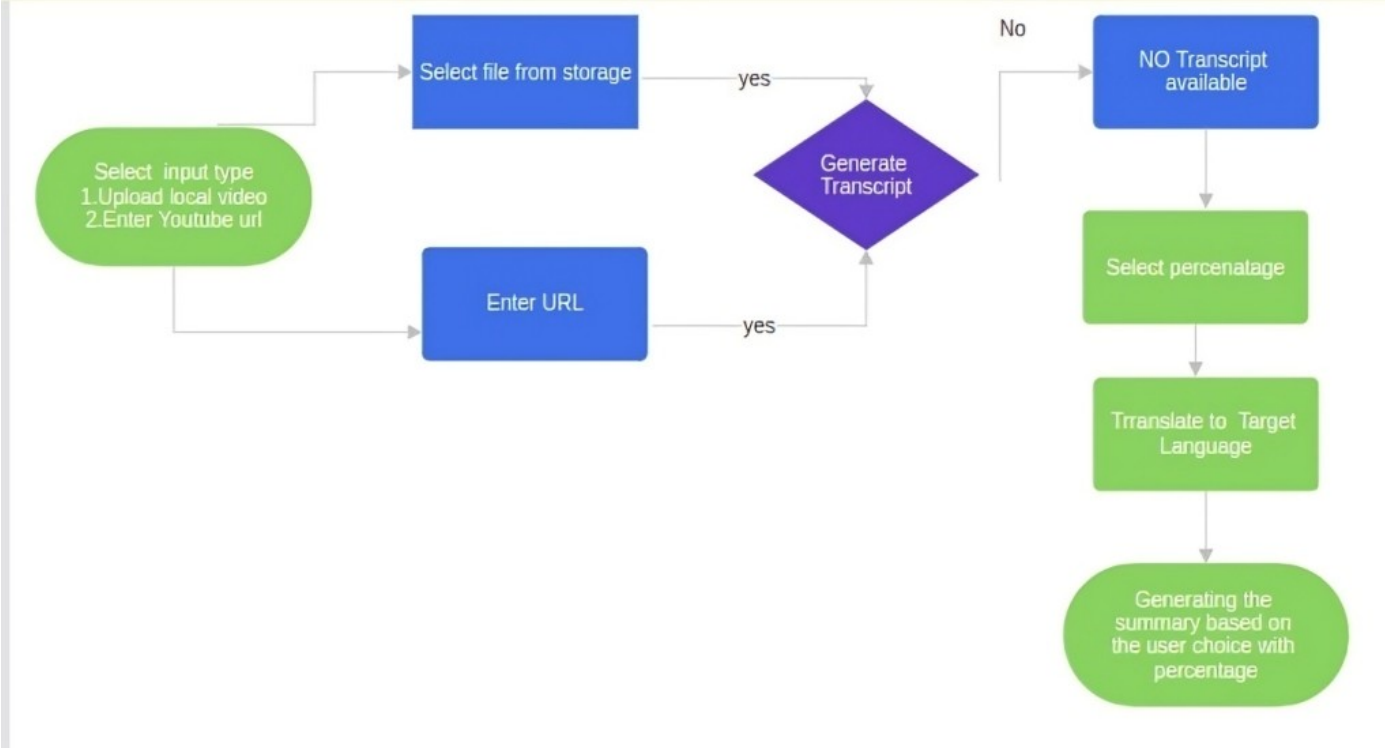
C. Software

1) Python

2) Visual Studio

3) Streamlit

FLOWCHART:



CHAPTER 3 IMPLEMENTATION

3.1 OVERALL CODE:

```
import os
import streamlit as st
import moviepy.editor as mp
from pydub import AudioSegment
from googletrans import Translator
from youtube_transcript_api import YouTubeTranscriptApi
import speech_recognition as sr
from transformers import BartTokenizer, BartForConditionalGeneration

# Set TOKENIZERS_PARALLELISM to avoid parallelism issues
os.environ["TOKENIZERS_PARALLELISM"] = "false"

# Function to process local video and extract text using Google Speech Recognition
def process_local_video(video_path):
    clip = mp.VideoFileClip(video_path)
    audio_path = "audio1.wav"
    clip.audio.write_audiofile(audio_path)  # Generate audio file

    # Loading audio file
    audio_file = AudioSegment.from_wav(audio_path)

    # Define the duration of each segment in milliseconds
    segment_duration = 30000  # 30 seconds for more efficient processing

    result_text = ""

    # Process each segment
    recognizer = sr.Recognizer()
    for i in range(0, len(audio_file), segment_duration):
        # Calculate start and end times for the segment
        start_time = i
        end_time = min(i + segment_duration, len(audio_file))  # Ensure end time doesn't
exceed the audio duration

        # Extract the segment
```

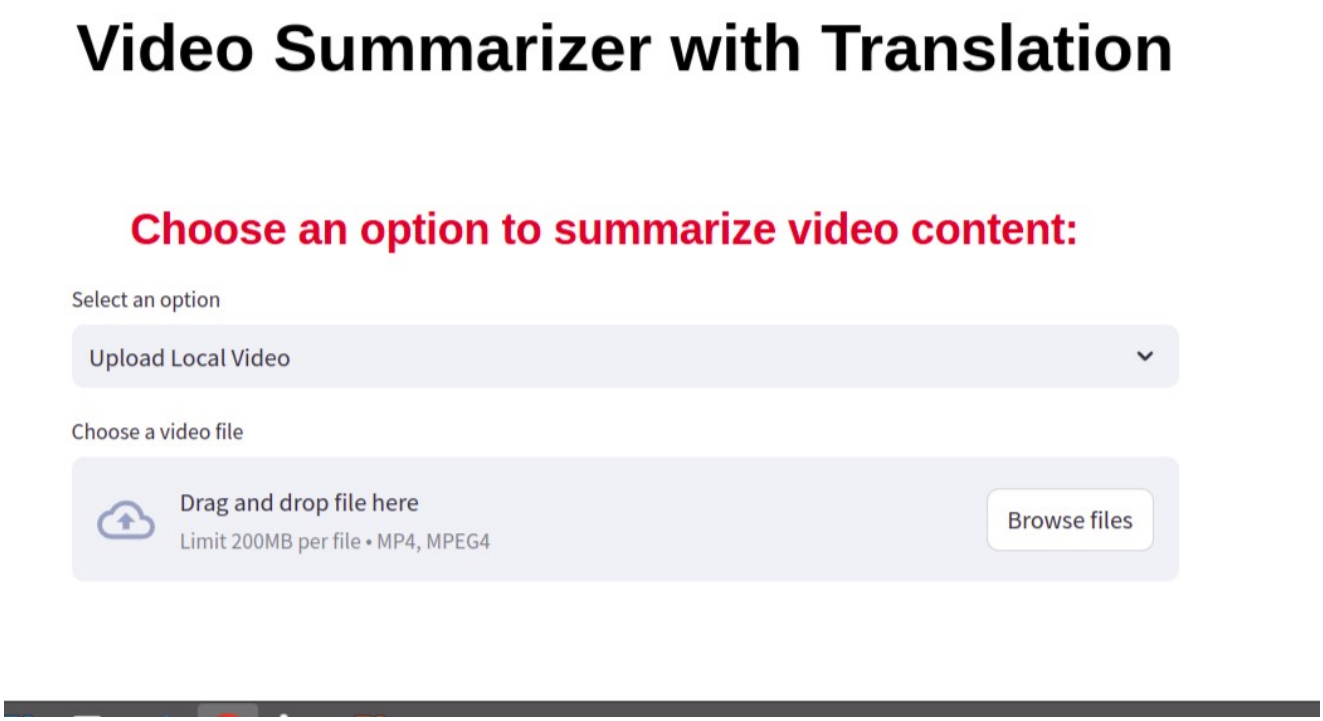
```
        segment = audio_file[start_time:end_time]

        # Export the segment as a temporary WAV file
        segment_path = "temp_segment.wav"
        segment.export(segment_path, format="wav")

        # Recognize speech from the segment
        with sr.AudioFile(segment_path) as source:
            audio_data = recognizer.record(source)  # Read the entire audio file
            try:
                text = recognizer.recognize_google(audio_data)  # Use Google Speech
Recognition
                result_text += text + " "
            except sr.UnknownValueError:
                print("Speech recognition could not understand audio")
            except sr.RequestError as e:
                print(f"Could not request results from Google Speech Recognition service;
{e}")

    return result_text

# Function to summarize text using BART model
def summarize_text(text, summary_percentage):
    model_name = "facebook/bart-large-cnn"
    tokenizer = BartTokenizer.from_pretrained(model_name)
    model = BartForConditionalGeneration.from_pretrained(model_name)

    # Tokenize the input text
    inputs = tokenizer([text], max_length=1024, return_tensors='pt', truncation=True)

    # Generate summary
    max_length = int(len(text.split()) * (summary_percentage / 100))
    min_length = int(max_length * 0.6)  # Ensure min_length is roughly 60% of max_length
    summary_ids = model.generate(inputs['input_ids'], max_length=min_length,
min_length=min_length, length_penalty=2.0, num_beams=4, early_stopping=True)
    summary = tokenizer.decode(summary_ids[0], skip_special_tokens=True)

    return summary

# Function to translate text to a specified language using Google Translate API
def translate_text(text, target_language):
    translator = Translator()
    language_codes = {
        "Telugu": "te",
        "Kannada": "kn",
        "Hindi": "hi",
        "Tamil": "ta"
```

```
    }
    translated = translator.translate(text, src='en',
dest=language_codes[target_language])
    return translated.text

# Streamlit frontend
st.markdown(
    """
    <style>
    .main {
        background-color: lightpink;
    }
    .title {
        color: blue;
        text-align: center;
        font-family: 'Arial Black', sans-serif;
    }
    .subtitle {
        color: green;
        text-align: center;
        font-family: 'Arial', sans-serif;
    }
    .instruction {
        color: blue;
        font-family: 'Arial', sans-serif;
    }
    </style>
    """,
    unsafe_allow_html=True
)

st.markdown("<h1 class='title'>Video Summarizer with Translation</h1>",
unsafe_allow_html=True)
st.markdown("<h3 class='subtitle'>Choose an option to summarize video content:</h3>",
unsafe_allow_html=True)

option = st.selectbox("Select an option", ["Upload Local Video", "Enter YouTube URL"])

if option == "Upload Local Video":
    uploaded_file = st.file_uploader("Choose a video file", type=["mp4"])
    if uploaded_file is not None:
        # Save the uploaded file
        with open("uploaded_video.mp4", "wb") as f:
            f.write(uploaded_file.read())

        st.markdown("<p class='instruction'>Processing the video...</p>",
unsafe_allow_html=True)
```

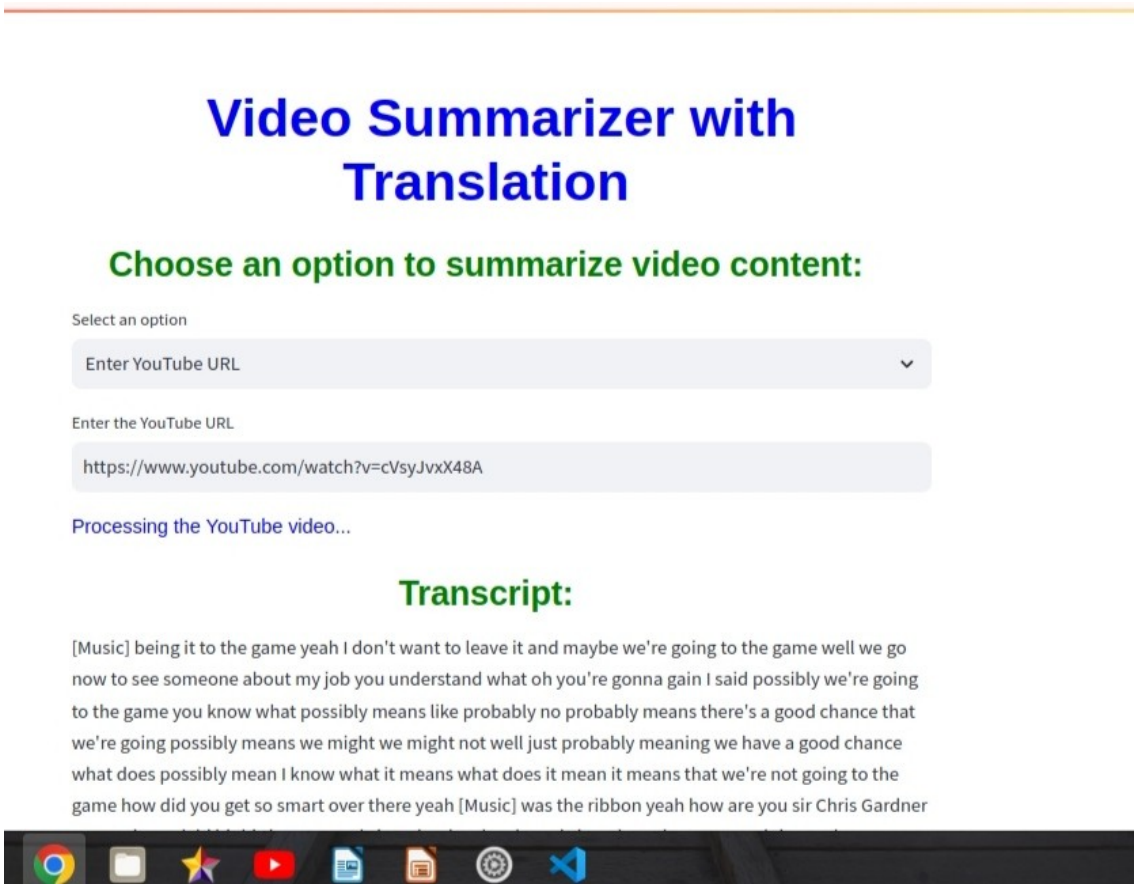
```
        result_text = process_local_video("uploaded_video.mp4")

        st.markdown("<h3 class='subtitle'>Transcript:</h3>", unsafe_allow_html=True)
        st.write(result_text)

        summary_percentage = st.select_slider(
            "Select summary length as percentage of the original text:",
            options=[10, 20, 30, 40, 50, 60, 70, 80, 90, 100],
            value=30
        )

        st.markdown(f"<h3 class='subtitle'>English Summary ({summary_percentage}%):</h3>",
unsafe_allow_html=True)
        summary = summarize_text(result_text, summary_percentage)
        st.write(summary)

        target_language = st.selectbox(
            "Select target language for translation:",
            options=["Kannada", "Telugu", "Hindi", "Tamil"]
        )

        st.markdown(f"<h3 class='subtitle'>{target_language} Summary ({summary_percentage}
%):</h3>", unsafe_allow_html=True)
        translated_summary = translate_text(summary, target_language)
        st.write(translated_summary)
```

3.3 OUTPUTS:

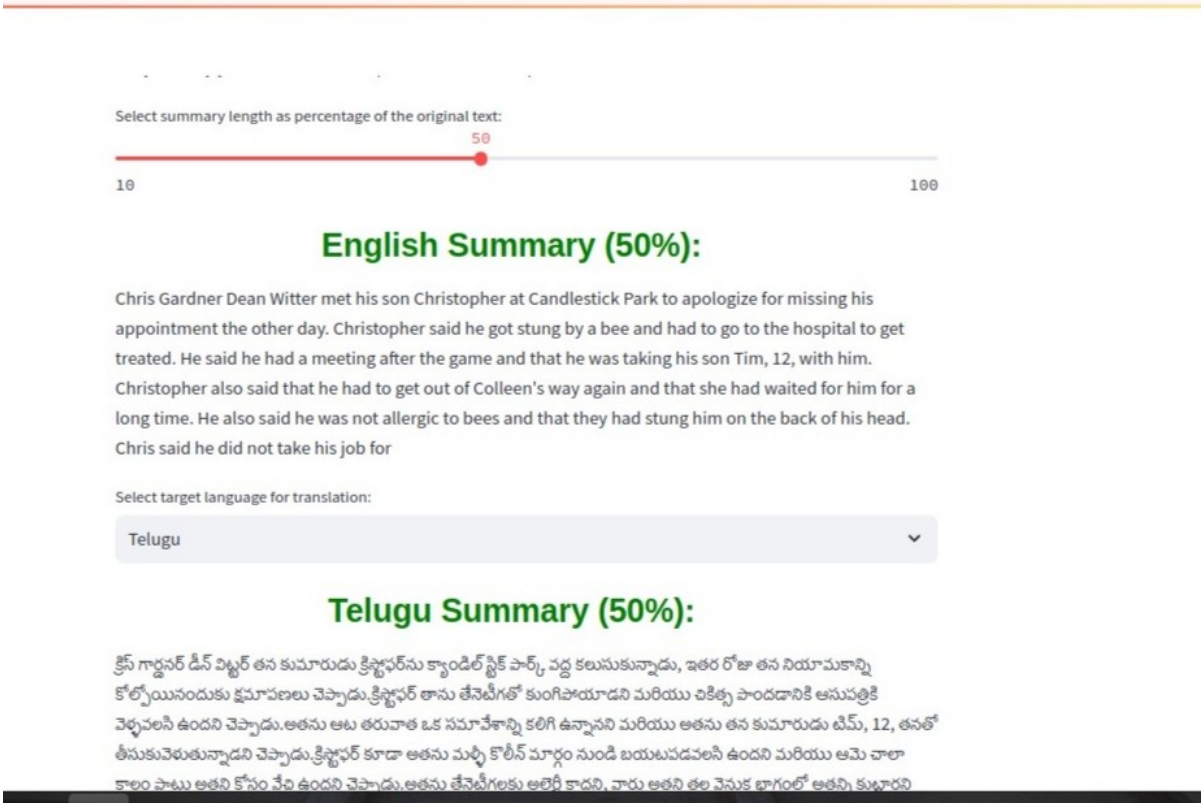
USER INTERFACE:



SELECT AND PASTE YOUTUBE URL:



SELECTED LANGUAGE TRANSCRIPT SUMMARY:



CONCLUSION

4.1Summarizer Project:

This project has proposed a YouTube Transcript Summarizer. The system takes the input YouTube video from the Chrome extension of the Google Chrome browser when the user clicks on the summarize button on the chrome extension web page, and access the transcripts of that video with the help of python API. The accessed transcripts are then summarized with the transformers package. Then the summarized text is shown to the user in the chrome extension web page.

This project helps the users a lot by saving their valuable time and resources. This helps us to get the gist of the video without watching the whole video. It also helps the user to identify the unusual and unhealthy content so that it may not disturb their viewing experience.

This project also ensures great user interface experience in finding out the summarized text as chrome extensions have been used. This helps in getting the summarized text without copying the URL and pasting at terminals or by any third-party applications.

4.2 Future Scope of YouTube Video Summarizer:

1.Enhanced Language Support: Expand language options for transcription and translation.

2.Improved Summarization Accuracy: Integrate advanced AI models for better summaries.

3.Customization Options: Allow users to select sections for summarization.

4.Multi-Video Summarization: Summarize playlists or multiple videos.

5.Visual Summaries: Generate visual summaries alongside text.

6.User Feedback Integration: Improve algorithms based on user feedback.

7.Mobile App Development: Create a mobile version for on-the-go use.

8.Integration with Other Platforms: Support additional video platforms.

9.Advanced Analytics: Offer insights like sentiment analysis and keyword extraction.

4.3 REFERENCES:

https://ieeexplore.ieee.org/document/9683609

https://ieeexplore.ieee.org/document/10128375