YOUTUBE VIDEO SUMMARIZER USING ARTIFICIAL INTELLIGENCE

A project report submitted in partial fulfillment of the requirements

for the award of credits in

AI Tools & Machine Learning Fundamentals

A skill-oriented course of

Bachelor of Technology

in

ELECTRONICS & COMMUNICATION ENGINEERING

By

B. BHAVYA SRI

23BQ1A0424

G. MOUNIKA B. INDU RANI D. ADITHYA

23BQ1A0442 23BQ1A0421 24BQ5A0403

DEPARTMENT OF ELECTRONICS AND COMMUNICATIO ENGINEERING

VASIREDDY VENKATADRI INSTITUTE OF TECHNOLOGY

(Approved by AICTE and permanently affiliated to JNTUK)

Accredited by NBA and NAAC with 'A' Grade

NAMBUR (V), PEDAKAKANI (M), GUNTUR-522 508

JUNE 2025

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING
VASIREDDY VENKATADRI INSTITUTE OF TECHNOLOGY: NAMBUR
JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY: KAKINADA

CERTIFICATE

This is to certify that the project titled "**YouTube Video Summarizer Using Aritificial Intelligence**" is a bonafide record of work done by **B. Bhavya Sri, G. Gagana Mounika, B. Indurani, D. Adithya** under the guidance of **Dr. M. Venkatesh, Associate Professor** in partial fulfillment of the requirement for the award of credits in **AI Tools & Machine Learning Fundamentals** - a skill-oriented course of Bachelor of Technology in Electronics and Communication Engineering, JNTUK during the academic year 2024–25.

Dr. M. Venkatesh	**Dr. M. Y. Bhanu Murthy**
Course Instructor	**Head of the Department**

DECLARATION

We are **Bhavya Sri. B, Gagana Mounika. G, Indurani. B, Adithya. D,** hereby declare that the project report entitled **"YouTube Video Summarizer Using Artificial Intelligence"** done by us under the guidance of **Dr. M. Venkatesh, Associate Professor, Department of Electronics and Communication Engineering** is submitted by partial fulfilment of requirements for the award of degree of **BACHELOR OF TECHNOLOGY** in **ELECTRONICS AND COMMUNICATION ENGINEERING**

Date: -

Place: - VVIT, Nambur

Signature of the candidates

B. BHAVYA SRI (23BQ1A0424)

G. MOUNIKA (23BQ1A0442)

B. INDU RANI (23BQ1A0421)

D. ADITHYA (24BQ5A0403)

ACKNOWLEDGEMENT

We express our sincere thanks wherever it is due

We express our sincere thanks to the Chairman, Vasireddy Venkatadri Institute of Technology, **Sri Vasireddy Vidya Sagar** for providing us well equipped infrastructure and environment.

We thank **Dr. Y. Mallikarjuna Reddy**, Principal, Vasireddy Venkatadri Institute of Technology, Nambur, for providing us the resources for carrying out the project.

We express our sincere thanks to **Dr. K. Giribabu**, Dean of Studies for providing support and stimulating environment for developing the project.

Our sincere thanks to **Dr. M. Y. Bhanu Murthy**, Head of the Department, Department of ECE, for his co-operation and guidance which help us to make our project successful and complete in all aspects.

We also express our sincere thanks and are grateful to our guide **Dr. M. Venkatesh**, Associate Professor, Department of ECE, for motivating us to make our project successful and fully complete. We are grateful for her precious guidance and suggestions.

We also place our floral gratitude to all other teaching staff and lab technicians for their constant support and advice throughout the project.

NAME OF THE CANDIDATES:

B. BHAVYA SRI (23BQ1A0424)

G. G. MOUNIKA (23BQ1A0442)

B. INDURANI (23BQ1A0421)

D. ADITHYA (24BQ5A0403)

CONTENTS

LIST OF FIGURES

ABSTRACT

The rapid growth of video content on platforms like YouTube has made it increasingly difficult for users to consume and extract essential information efficiently. To address this challenge, this project titled "YouTube Video Summarizer using Python" presents a practical solution that allows users to generate intelligent, prompt-based summaries of YouTube videos without needing to watch them in full. The system uses the open-source tool yt-dlp to extract auto-generated English captions from YouTube videos directly, eliminating the need for any official APIs. These captions are saved in .vtt format and subsequently converted into plain text for processing. To generate summaries, the project integrates the Gemini 2.0 Flash model from Google via the Gemini API, which produces high-quality responses based on user-defined prompts. This allows flexible interactions such as summarization, answering specific questions, or extracting key points from the video transcript.

A user-friendly Streamlit interface serves as the front-end for the application. Users simply input a YouTube video URL and their custom prompt (e.g., "Summarize the video" or "List key takeaways"), and the system responds with a coherent, context-aware summary. Error handling and loading spinners are incorporated to ensure a smooth user experience. This project demonstrates a successful integration of video data extraction, natural language processing, and generative AI within a real-world application. It highlights the capability of modern language models like Gemini in understanding and summarizing large volumes of unstructured textual content derived from videos. The system is especially useful for students, educators, researchers, and professionals who seek quick insights from video content.

In addition to its core functionality, the project emphasizes modular and maintainable code structure, making it adaptable for future enhancements such as multilingual support, audio-to-text integration, or storage of past summaries for later reference. The choice of using open-source tools like yt-dlp ensures transparency, accessibility, and long-term sustainability without dependency on paid APIs or third-party limitations. Furthermore, by allowing users to define their own prompts, the system moves beyond fixed-function summarization and becomes a customizable content understanding tool.

CHAPTER 1
INTRODUCTON

In the era of digital media, video content has become a dominant medium for learning, communication, and entertainment. YouTube, in particular, hosts millions of videos covering a wide range of subjects, from academic lectures and tutorials to news, reviews, and documentaries. While this abundance of information offers immense value, it also presents a challenge—viewers often lack the time or patience to watch long videos to extract the information they need.

This has created a demand for tools that can summarize video content intelligently, enabling users to consume information more efficiently. Traditional summarization tools are often limited to text documents, while video summarization generally involves either manual effort or advanced video processing. However, with advancements in Natural Language Processing (NLP) and large language models (LLMs), it is now possible to generate meaningful summaries based on the textual captions of videos.

The "YouTube Video Summarizer using Python" project addresses this challenge by allowing users to input a YouTube video URL and a custom prompt (e.g., "Summarize the video", "List key points", "Give a short review") through a simple web interface. Behind the scenes, the system extracts captions using the yt-dlp tool and processes them using Google's Gemini API, which generates high-quality, context-aware summaries. Built with a user-friendly interface using Streamlit, the application is designed to be accessible to both technical and non-technical users. This project demonstrates how various technologies—caption extraction, text processing, and generative AI—can be effectively combined to solve a real-world problem. The integration of user input ensures flexibility, while automation reduces manual effort. Overall, the project addresses a growing need for quick and efficient content consumption in an age dominated by digital media. This project reflects the convergence of automation, AI, and user-centric design to solve a real-world problem. It not only helps in saving time but also enhances how users interact with digital video content. The system is a practical example of how AI can be applied in everyday scenarios to improve information retrieval and accessibility.

1.1. Need of the Project

In recent years, video content has become a key source of learning, research, and entertainment. Whether it is a student watching a lecture, a professional learning a new skill, or a researcher reviewing interviews or documentaries, YouTube has emerged as a go-to platform. However, one of the biggest challenges user's faces is the *time and effort* it takes to extract useful information from these videos.

This project aims to solve that issue by offering a way to automatically summarize YouTube videos based on user needs. The system is designed to meet a real and growing demand for tools that:

- Help save time by turning long videos into short, easy-to-read summaries.
- Offer personalized output, where the user controls what kind of summary they want—whether it is a brief overview, a list of key points, or answers to specific questions.
- Make video content more accessible, especially for users with limited time, bandwidth, or learning preferences.
- Reduce reliance on paid or complex APIs, using free tools like yt-dlp for extracting captions and Gemini for generating smart summaries.

Moreover, the need for such tools is growing in fields like education, where students can use this to revise lectures; in research, where professionals can analyse video-based interviews or discussions; and in content creation, where creators can quickly extract highlights from source material.

Increasing Volume of Online Video Content

With the rise of digital platforms, especially YouTube, there has been an explosion in the amount of video content available online. While this offers immense learning and informational value, it also creates challenges for users who want to quickly extract relevant information from long videos. Watching entire videos to find key points can be time-consuming and inefficient.

Lack of Time for Full Video Consumption

In academic, professional, and personal contexts, users often seek quick insights or specific answers without the need to go through lengthy video content. This is especially true for students preparing for exams, professionals researching a topic, or anyone trying to learn on a tight schedule. A tool that condenses video information can help users consume content more efficiently.

This project reflects the convergence of automation, AI, and user-centric design to solve a real-world problem. It not only helps in saving time but also enhances how users interact with digital video content. The system is a practical example of how AI can be applied in everyday scenarios to improve information retrieval and accessibility.

CHAPTER 2

LIBRARIES DESCRIPTION

2.1 Streamlit Library

Overview:

Streamlit is an open-source Python framework designed specifically for building interactive web applications for data science and machine learning. It allows developers to turn Python scripts into shareable web apps with just a few lines of code—no need to learn front-end web development. With support for widgets like text inputs, buttons, sliders, and file uploaders, it enables real-time interactivity, making it ideal for prototyping and deploying AI models and data-driven tools.

Streamlit automatically updates the UI when the underlying code or data changes, providing a dynamic user experience. In this project, Streamlit plays a central role by serving as the interface that connects users with the backend processes—such as entering YouTube URLs, viewing the AI-generated summaries, and interacting with outputs—all within a simple, responsive layout.

Purpose in the Project:

In this project, Streamlit is used to create a user-friendly interface that allows users to:

- Input the YouTube video URL.
- Enter a prompt (e.g., "Summarize this video" or "List key points").
- View the AI-generated summary directly on the page.
- Receive real-time feedback (e.g., loading spinners, error messages).

Why Streamlit?

- Simplicity: Easy to set up and integrate with Python scripts.
- Fast prototyping: Ideal for quick development and testing of machine learning apps.
- Custom UI components: Offers inputs like text boxes, buttons, and sliders.
- Live updates: Automatically updates the output as inputs change.

Key Functions Used:

- st.title(), st.text_input(), st.button() – for layout and user input.
- st.spinner() – to show loading messages while captions and summaries are being processed.
- st.write() and st.error() – to display results and errors.

2.2 YouTube-DLP (yt-dlp)

Overview:

In this project, yt-dlp is primarily used to extract subtitles or captions from YouTube videos, which serve as the input for the AI summarization model. One of its key advantages is the ability to download captions in multiple formats, including .vtt, which can be easily converted to plain text for further processing. It supports a wide range of video sources beyond YouTube, making it highly versatile for various content extraction tasks. Additionally, yt-dlp allows customization through command-line arguments, enabling precise control over what data is retrieved, such as specific language subtitles or avoiding unnecessary downloads like video files. Its active developer community and rich feature set make it a reliable tool for automation in content-processing pipelines like this one.

Purpose in the Project:

In this project, yt-dlp is used only to download the auto-generated English subtitles of a YouTube video without downloading the full video. These subtitles serve as the raw transcript for the AI model to summarize.

Why yt-dlp?

- No need for YouTube API: Avoids the complexity of YouTube's official Data API and quota limits.
- Custom options: Can fetch only subtitles, skip video download, and save files in .vtt format.
- Supports many sites: Although used here for YouTube, it can be reused for other platforms in future projects.

Key Options Used:

- Skip download: Prevents downloading the actual video file.
- Enables downloading of subtitles (manual and auto-generated).
- Specifies language (in this case, English).
- Sets custom file name pattern using the video ID.

The yt-dlp library is a command-line program used for downloading videos and extracting audio or metadata from YouTube and many other video platforms. It is a fork of the popular youtube-dl project, created to include additional features, bug fixes, and faster updates. yt-dlp supports downloading subtitles, extracting video information such as title, duration, and description, and allows customization through various options like format selection, output templates, and post-processing actions.

2.3 Operating System Interface (os)

Overview:

The os module is part of Python’s standard library and provides functions to interact with the operating system. It allows programs to perform file and directory operations, making scripts more flexible and cross-platform.

The operating system interface library in Python, commonly known as the os library, provides a way to interact with the underlying operating system in a portable manner. It allows Python programs to perform tasks such as reading or writing to the file system, managing directories, retrieving environment variables, and executing system commands. The os library plays a crucial role in handling file paths, launching processes, and interacting with the system shell. It supports various functions that make it easier to write platform-independent scripts by abstracting system-specific operations. This library is part of Python's standard utility modules and is widely used for system-level scripting and automation.

Moreover, the ability to build OS-aware scripts ensures that the application can run reliably on different platforms such as Windows, macOS, or Linux. This adds to the overall robustness and portability of the project, making the deployment process smoother across varied environments.

Purpose in the Project:

In this project, os is used to:

- Create a folder to store downloaded subtitle files.
- Generate dynamic file paths that work across operating systems (Windows, macOS, Linux).
- Manage file structure safely during caption extraction and processing.

Why os?

- No external installation required: It comes pre-installed with Python.
- Cross-platform compatibility: Helps handle file paths in a way that works on any OS.
- File safety: Ensures folders exist before saving files to avoid errors.

Key Functions Used:

- os.makedirs(output_dir,exist_ok=True)

  Creates the output directory only if it does not already exist.
- os.path.join(output_dir,'%(id)s.%(ext)s')

  Creates a clean, valid path to save subtitle files using the YouTube video ID.

The os library in Python is a built-in module that provides a way for programs to interact with the operating system. It allows for tasks such as creating, deleting, and navigating directories, handling file paths, and accessing environment variables.

2.4 Regular Expressions Module (re)

Overview:

The re module in Python stands for Regular Expressions. It allows developers to search, match, and manipulate strings using patterns. It is particularly useful for extracting specific portions of text that follow known formats, such as URLs, email addresses, or codes.

The regular expressions library in Python, known as re, provides powerful tools for pattern matching and text processing. It allows users to search, match, and manipulate strings using specific patterns defined by a syntax known as regular expressions. With re, tasks like validating email formats, extracting specific data from text, or replacing parts of a string become much easier. The library includes functions such as search, match, findall, sub, and split, which help in locating and processing patterns efficiently. It supports a wide range of metacharacters and special sequences, making it highly versatile for both simple and complex text operations.

Whether the goal is to extract meaningful segments from user input or to sanitize content before further analysis, this tool proves helpful in maintaining both accuracy and efficiency. Its integration with string data makes it a valuable component in tasks like form validation, log file analysis, or content parsing.

Purpose in the Project:

In this project, the re module is used to:

- Extract the YouTube video ID from the URL entered by the user.
- Ensure the script can find and match the corresponding subtitle file based on that ID.

Why re?

- Easily finds structured text like YouTube video IDs.
- Helps extract only the needed part of the URL.
- Can be reused to extract data from various text formats in future applications.

Key Pattern Used:

- Regex pattern:

  Looks for the "v=" part in the URL.

  Captures everything after "v=" until it hits an & (if present)**.**

The re library in Python is used for working with regular expressions, which are patterns that help in searching, matching, and manipulating text. It allows developers to efficiently handle tasks like validating inputs, extracting specific data from strings, or performing complex text replacements. With functions such as search, match, findall, and sub, the re library provides a powerful way to process and analyze textual data in a flexible and concise manner.

2.5 Time Module (time)

Overview:

The time module is part of Python's standard library and provides various functions to work with time-related operations such as delays, timestamps, and performance measurements. It is especially useful in managing the timing of operations and introducing pauses when needed.

The time library in Python provides various time-related functions that are useful for manipulating and working with time and date information. It allows you to access the current time, pause the execution of a program, measure the time taken by code segments, and convert between different time formats. Common functions include time(), which returns the current time in seconds since the epoch, sleep(), which delays execution for a specified number of seconds, and strftime(), which formats time as a readable string. The library is especially useful for scheduling tasks, creating delays, and tracking performance in programs.

In many projects, especially those involving automation or user interaction, it's important to control the flow of execution. The time module helps manage how and when certain parts of the program should run.

Purpose in the Project:

In this project, the time module is used to:

- Introduce a delay between API calls to Gemini, helping prevent rate-limiting or overload issues.
- Ensure that the external AI API has enough time to respond without rushing multiple requests in a short span.

Why time?

- Avoid API errors: Waiting a few seconds between requests helps maintain stability when dealing with external services like Google's Gemini API.
- Simple and effective: One line of code can manage pauses without adding complexity.
- Part of standard Python: No installation required.

Key Function Used:

- time.sleep(5): This pauses the program for 5 seconds before making a call to the Gemini API. Helps reduce the risk of request limits being exceeded.

The time library in Python provides essential functions for working with time-related tasks. It allows you to measure execution time, create delays, and retrieve the current system time. Common functions like sleep(), time(), and ctime() make it useful for scheduling events, simulating real-time behavior, or managing time-based processes in programs.

2.6 Web Video Text Tracks (webvtt)

Overview:

The webvtt is a Python library used to parse Web Video Text Tracks (.vtt) files. These files are commonly used to store subtitles or captions for online videos, including those generated automatically by YouTube. The library allows developers to read, extract, and manipulate the text content from these caption files.

The Web Video Text Tracks (webvtt) library is a Python tool designed to parse and work with WebVTT files, which are commonly used for subtitles and captions in online videos. This library allows developers to read, edit, and generate WebVTT caption files with ease. It provides a simple interface to access the timing and text of each caption cue, making it useful for applications like video accessibility, transcription editing, and subtitle synchronization. The webvtt library supports basic operations such as loading files, iterating over cues, and exporting modified tracks, making it a lightweight and practical choice for handling subtitle data in Python projects.

By using webvtt, developers can programmatically access and work with caption data without needing to understand the file's raw structure. It streamlines tasks like looping through subtitle entries or converting them into plain text.

Purpose in the Project:

In this project, webvtt is used to:

- Read the .vtt subtitle file downloaded from YouTube.
- Extract and clean the transcript, removing timestamps and formatting.
- Return a plain text version of the video content suitable for AI summarization.

Why webvtt?

- Specifically built for .vtt files: Simplifies working with caption formats.
- Removes need for manual parsing: Automatically skips over timestamps and metadata.
- Lightweight and efficient: Perfect for quick text extraction tasks in video analysis.

Key Function Used:

- webvtt.read(vtt_path): Reads the .vtt file and returns a list of caption blocks. Each block contains clean text that can be joined into a full transcript

In the context of this project, it is used to convert the caption data downloaded from YouTube into clean, readable text that can then be processed by an AI model for summarization. The library simplifies handling time-stamped caption data by providing structured access to each caption segment.

2.7 Google Generative AI (google.generativeai)

Overview:

The Google Generative AI library is a Python package developed by Google to simplify access to their generative models, including Gemini. It allows developers to easily integrate text generation, summarization, and other AI-driven capabilities into their applications. This library provides a clean interface for sending prompts to Google's large language models and receiving intelligent responses. It supports features like context-aware generation, response formatting, and safety settings, making it useful for both research and production environments. The library is especially designed for developers working with natural language tasks who want to utilize Google’s powerful AI models without managing low-level API calls manually.

Purpose in the Project:

In this project, the google.generativeai library is used to:

- Send a custom user prompt along with the YouTube transcript to Gemini.
- Receive a summary or answer based on the video’s content.
- Perform intelligent language understanding without requiring Hugging Face or OpenAI APIs.

Why google.generativeai?

- Gemini provides advanced understanding of context, tone, and summarization needs.
- Users can request summaries, key points, explanations, etc.
- Reliable and free-tier access: Gemini Flash model offers fast, cost-effective results for lightweight tasks like summarization.

Key Workflow:

In this project, the google.generativeai library is used to interact with the Gemini model for generating summaries. First, the API is configured securely using an API key. The Gemini 2.0 Flash model is then initialized to handle content generation tasks. A prompt is constructed by combining the user's request (e.g., "Summarize the video") with the full transcript extracted from the YouTube captions. This combined input is sent to the model using the generate_content() function. The response generated by the model—usually a summary or explanation—is then captured and displayed to the user through the Streamlit interface.

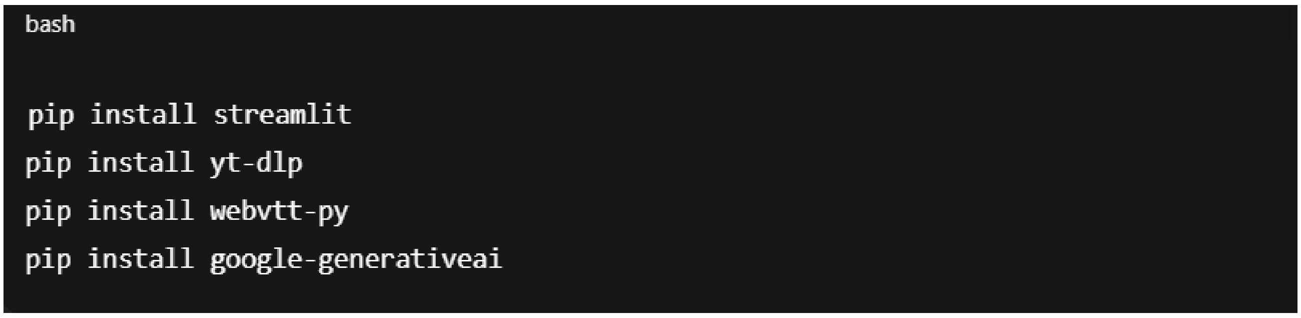
Google Generative AI refers to a suite of advanced artificial intelligence models developed by Google that can understand and generate human-like text, images, code, and more. These models, such as Gemini, are designed to perform a wide range of language-based tasks, including summarization, translation, question answering, and creative writing.

CHAPTER 3

IMPLEMENTATION

3.1 Installing the Required Libraries

To successfully run this project, several Python libraries must be installed. These include both standard and third-party libraries. Use the following commands in your terminal or command prompt to install them:



```bash
pip install streamlit
pip install yt-dlp
pip install webvtt-py
pip install google-generativeai
```

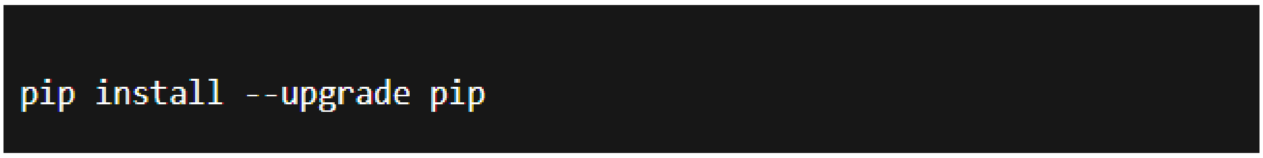
Fig.3.1.1 : Command for Installing Required Libraries

These commands will install:

- streamlit – to create the web-based user interface.
- yt-dlp – to download subtitles (captions) from YouTube videos.
- webvtt-py – to extract readable text from .vtt caption files.
- google-generativeai – to access Gemini AI for summarization.

Note: re, os, and time are built-in Python standard libraries, so they do not require installation.

It is important to make sure your internet connection is active during installation, as the libraries are fetched from the Python Package Index (PyPI). If any installation fails, check for spelling errors or try updating pip using:



```
pip install --upgrade pip
```

Fig 3.1.2 : Command for Updating pip

With all libraries successfully installed, the project is ready to connect with Google's Gemini AI for generating responses.

3.2 Generating the Gemini API Key

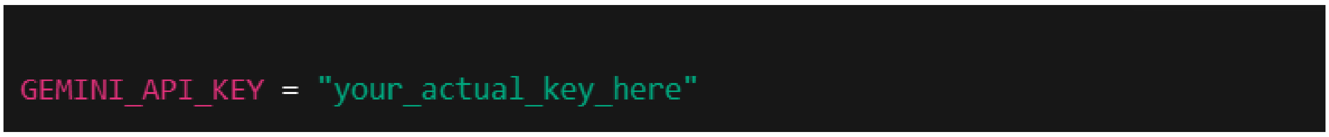
To use Google's Gemini API for text generation and summarization, an API key is required. This key acts like a password that allows your Python code to securely access the Gemini model.

Steps to Get the API Key:

- Visit Google AI Studio and Sign in using your Google account.
- Click on your profile icon and select "API Keys".
- Click "Create API Key" and give it a name (e.g., YouTubeSummarizerKey).
- Copy the API key shown – you will need this to use in your Python code.

Store the Key Securely in Your Project

Create a folder named .streamlit in your project directory, and inside it, create a file named secrets.toml with the following content:



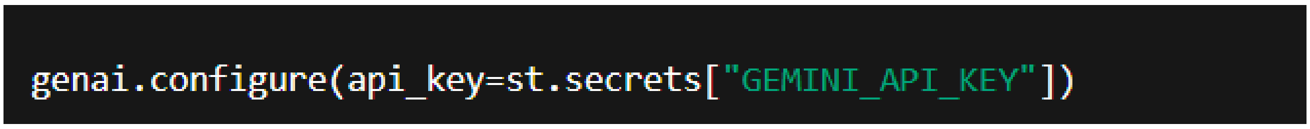
```
GEMINI_API_KEY = "your_actual_key_here"
```

Fig 3.2.1 : File with Gemini API Key

Note: Do not share your API key publicly or upload it to GitHub.

Accessing the Key in Code

Inside your Python script, you can now access this key using the following code:



```
genai.configure(api_key=st.secrets["GEMINI_API_KEY"])
```

Fig 3.2.2 : Code to Access the API Key inside Python script

Once the API key is correctly configured, your project can securely send prompts and transcripts to the Gemini model and receive intelligent summaries in response. It is important not to expose the API key directly in the source code, especially when sharing or deploying the project. Using secure methods such as .env files or Streamlit's st.secrets is recommended to protect the key and ensure safe access to the generative AI services. In the code, the key is accessed securely by assigning it to a variable or loading it from a configuration file or environment variable.

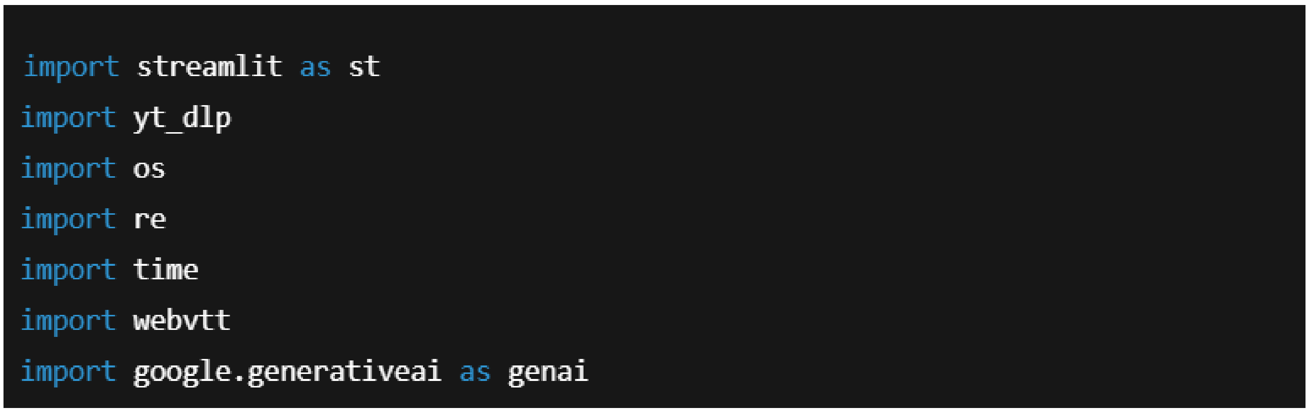
To use Google's Generative AI services like the Gemini API, an API key is required to authenticate and authorize access. This key is typically obtained by creating a project in Google Cloud and enabling the appropriate API.

3.3 Importing Required Libraries

Importing libraries in programming is essential because they provide pre-written code and functions that help perform common tasks more efficiently. Instead of writing everything from scratch, developers can use libraries to handle specific operations like data processing, user interface design, or machine learning. This not only saves time but also ensures that the code is more reliable and optimized, as these libraries are often developed and tested by experts. By importing libraries, programmers can focus on building the unique features of their application while relying on existing tools to manage the technical details.

After installing all the necessary libraries, the first step in your Python code is to import them. This ensures that all external and standard modules are available for use within your script.

Below is the import section used in the project:



```python
import streamlit as st
import yt_dlp
import os
import re
import time
import webvtt
import google.generativeai as genai
```

Fig 3.3 : Importing Required Libraries

All import statements and Python code for this project should be written in a Python script file using any code editor or text editor of your choice. Commonly used editors include Visual Studio Code (VS Code), PyCharm, or even a basic editor like Notepad. If using Notepad, it is important to save the file with a .py extension (for example: summarizer.py) to ensure it is recognized as a Python file.

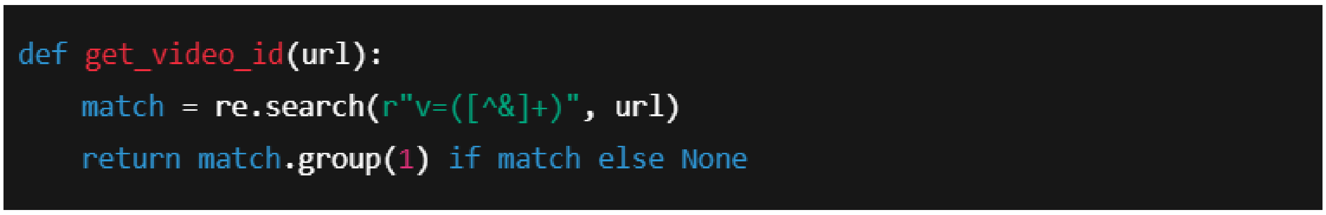
Once saved, the script can be executed through the command prompt or terminal. If using an IDE like VS Code, the built-in terminal or run feature can be used directly. This flexibility allows beginners to write code in simple tools while also supporting advanced environments for more efficient development.

3.4 Extracting the YouTube Video ID

Extracting the video ID from a YouTube URL is important because it serves as the unique identifier for each video on the platform. This ID allows developers to interact directly with YouTube's APIs or tools like yt-dlp and YouTube Transcript API to retrieve specific data such as video details, transcripts, captions, or download links. Since a full URL can come in different formats, isolating the video ID ensures that the program can consistently and accurately identify the correct video regardless of how the URL is shared or formatted. This step is essential for automating tasks that involve processing or analyzing YouTube videos.

Before downloading captions from a YouTube video, it is important to extract the unique video ID from the provided URL. YouTube URLs typically contain a parameter like v=VIDEO_ID, where VIDEO_ID is a unique string that identifies each video.

To extract this ID, a custom function named get_video_id() is used in the project. It takes the full YouTube URL as input and applies a regular expression to search for the part of the string that matches the video ID pattern. Here is the code used:



```python
def get_video_id(url):
    match = re.search(r"v=([^&]+)", url)
    return match.group(1) if match else None
```

Fig 3.4 : Function to get video ID from the URL

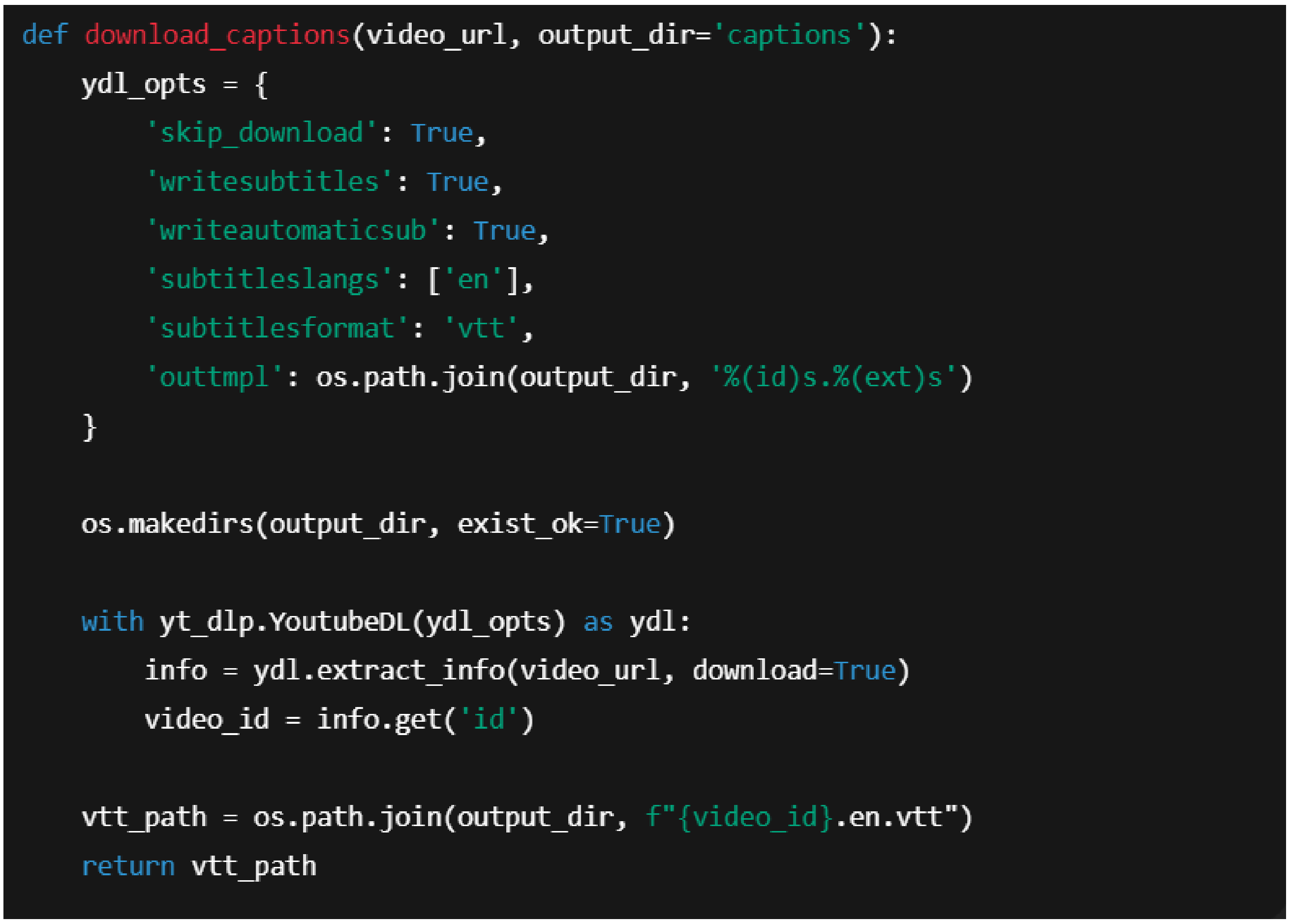
This function uses Python is built-in re module to match the video ID and return it. If the ID is successfully found, it is returned for further processing. Otherwise, the function returns None, indicating an invalid or unsupported URL. This step is essential because yt-dlp uses the video ID for naming downloaded caption files and handling requests.

Accurately extracting the video ID ensures that the system interacts with the correct video, avoiding errors during the download and processing stages. The regular expression used in the function is designed to handle different formats of YouTube URLs, including standard links, shortened links, and embedded video URLs. This makes the application more robust and user-friendly, as it can accommodate various ways users might input a link. By isolating the video ID at an early stage, the system streamlines the workflow and prevents unnecessary failures in caption extraction or file handling, thereby improving the overall reliability of the tool.

3.5 Downloading Captions Using yt-dlp Library

Once the YouTube video ID is extracted, the next step is to download the captions (subtitles) from the video. This is done using the powerful open-source tool yt-dlp, which can fetch video data, including subtitles, without downloading the full video itself.

The function used in this project is:



```python
def download_captions(video_url, output_dir='captions'):
    ydl_opts = {
        'skip_download': True,
        'writesubtitles': True,
        'writeautomaticsub': True,
        'subtitleslangs': ['en'],
        'subtitlesformat': 'vtt',
        'outtmpl': os.path.join(output_dir, '%(id)s.%(ext)s')
    }

    os.makedirs(output_dir, exist_ok=True)

    with yt_dlp.YoutubeDL(ydl_opts) as ydl:
        info = ydl.extract_info(video_url, download=True)
        video_id = info.get('id')

    vtt_path = os.path.join(output_dir, f"{video_id}.en.vtt")
    return vtt_path
```

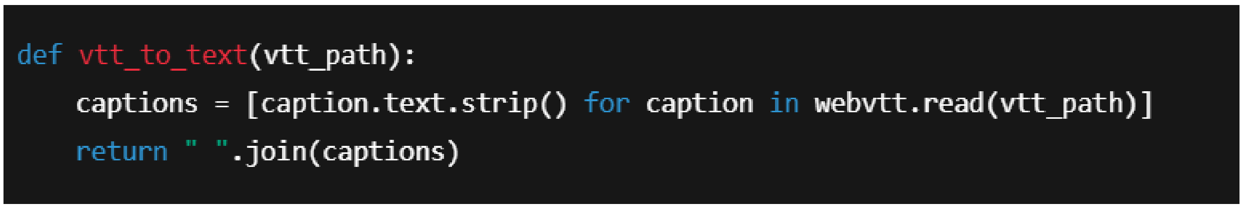
Fig 3.5 : Function to downloads captions from the video

In this project, we configure yt-dlp to skip downloading the video and only fetch the automatically generated English subtitles in .vtt format. The downloaded caption file is saved in a folder named captions, and the function returns the path to that file. This downloaded .vtt file will later be converted into plain text, which is then sent to Gemini for summarization. This step is very important, as it provides the actual transcript that will be summarized.

3.6 Converting Captions to Text using webvtt

Captions extracted from YouTube videos are often in the WebVTT format, which includes not just the spoken words but also timestamps and formatting information. To use these captions for tasks like summarization or natural language processing, it's important to convert them into plain text. This conversion removes unnecessary metadata such as time codes and formatting tags, leaving only the readable transcript. Clean, plain text makes it easier to process the content using language models, generate summaries, or perform text analysis, as these models require structured input without distractions from timing data or markup.

Once the .vtt (Web Video Text Tracks) subtitle file is downloaded using yt-dlp, it must be converted into readable plain text. This is essential because the .vtt file includes timestamps and formatting, which are not needed for summarization. The project uses a function called vtt_to_text() to handle this:



```python
def vtt_to_text(vtt_path):
    captions = [caption.text.strip() for caption in webvtt.read(vtt_path)]
    return " ".join(captions)
```

Fig 3.6 : Function to convert .vtt files to text

Explanation:

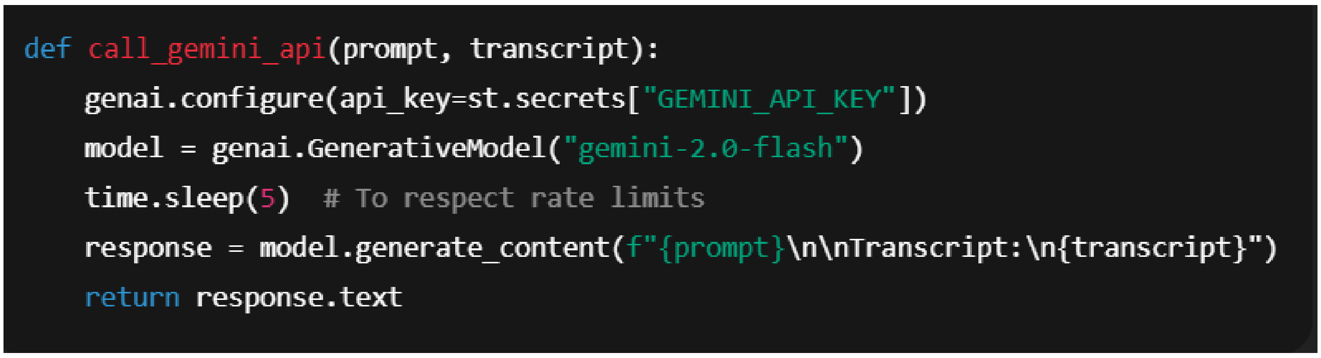
- The function takes the path of the .vtt file (vtt_path) as input.
- It uses the webvtt.read() method to load the subtitle file.
- Each caption block (which includes both timestamp and text) is read, and only the text part is extracted using caption.text.strip().
- All the individual caption texts are joined together into one continuous string using " ".join(captions).
- This final plain-text transcript is returned, ready to be sent to Gemini for summarization.

The reason this step is important is that AI models like Gemini work best with clean, uninterrupted text — not subtitle formatting or time codes. This function ensures the input to the AI is optimized for better summaries.

3.7 Generating Summary Using Gemini API

After the transcript is extracted from the YouTube captions, the next step is to generate a response using Google’s Gemini language model. This is done by sending both the transcript and a user-provided prompt (like “Summarize the video”) to the model.

The project uses the following function:



```
def call_gemini_api(prompt, transcript):
    genai.configure(api_key=st.secrets["GEMINI_API_KEY"])
    model = genai.GenerativeModel("gemini-2.0-flash")
    time.sleep(5)  # To respect rate limits
    response = model.generate_content(f"{prompt}\n\nTranscript:\n{transcript}")
    return response.text
```

Fig 3.7 : Function to generate summary using Gemini

Explanation:

- genai.configure(): Sets up the Gemini API using the secret API key stored in Streamlit's secrets.toml file.
- GenerativeModel("gemini-2.0-flash"): Initializes a lightweight and fast Gemini model to handle user requests.
- time.sleep(5): Waits 5 seconds to avoid exceeding request limits (helps maintain smooth performance).
- generate_content(): Sends the user’s prompt along with the full transcript to Gemini. The model analyzes this content and generates a suitable reply.
- The generated summary or response is returned as plain text.

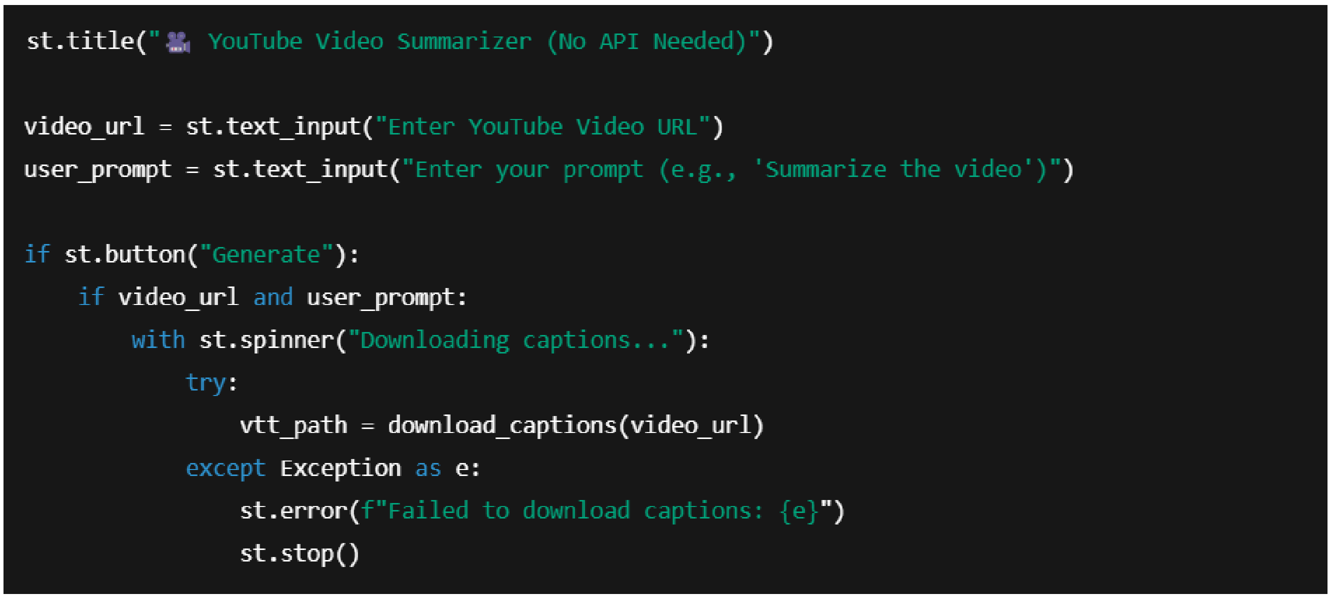
This function is a core part of the application. It allows users to ask custom questions about any video — not just get fixed summaries. For example, a user can enter:

- “Summarize the key points”
- “What is the speaker’s main argument?”
- “List the steps mentioned in the video”

The AI then tailors the output based on both the video content and the user’s needs, making the tool interactive and intelligent.

3.8 Creating the User Interface with Streamlit

The final and most interactive part of the project is building the front-end of the application using Streamlit. This provides a clean and simple web-based interface where users can input a YouTube video URL and their custom prompt.

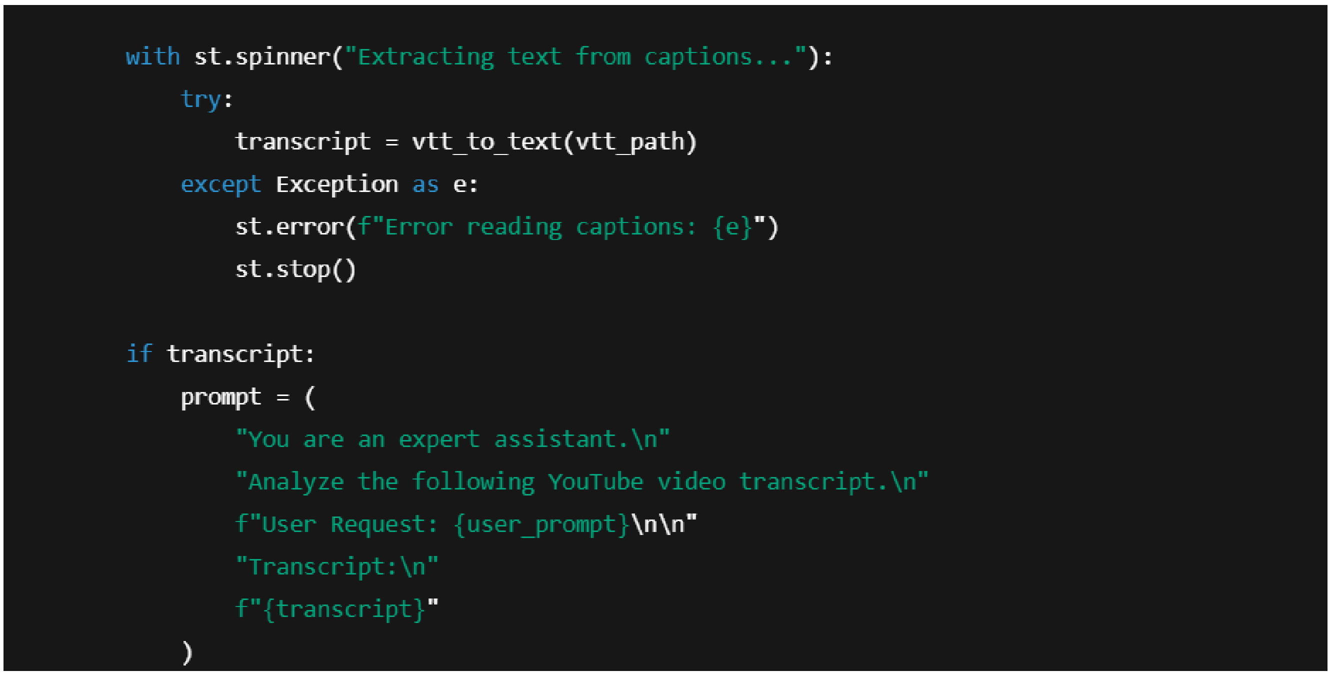


```python
st.title("🎥 YouTube Video Summarizer (No API Needed)")

video_url = st.text_input("Enter YouTube Video URL")
user_prompt = st.text_input("Enter your prompt (e.g., 'Summarize the video')")

if st.button("Generate"):
    if video_url and user_prompt:
        with st.spinner("Downloading captions..."):
            try:
                vtt_path = download_captions(video_url)
            except Exception as e:
                st.error(f"Failed to download captions: {e}")
                st.stop()
```

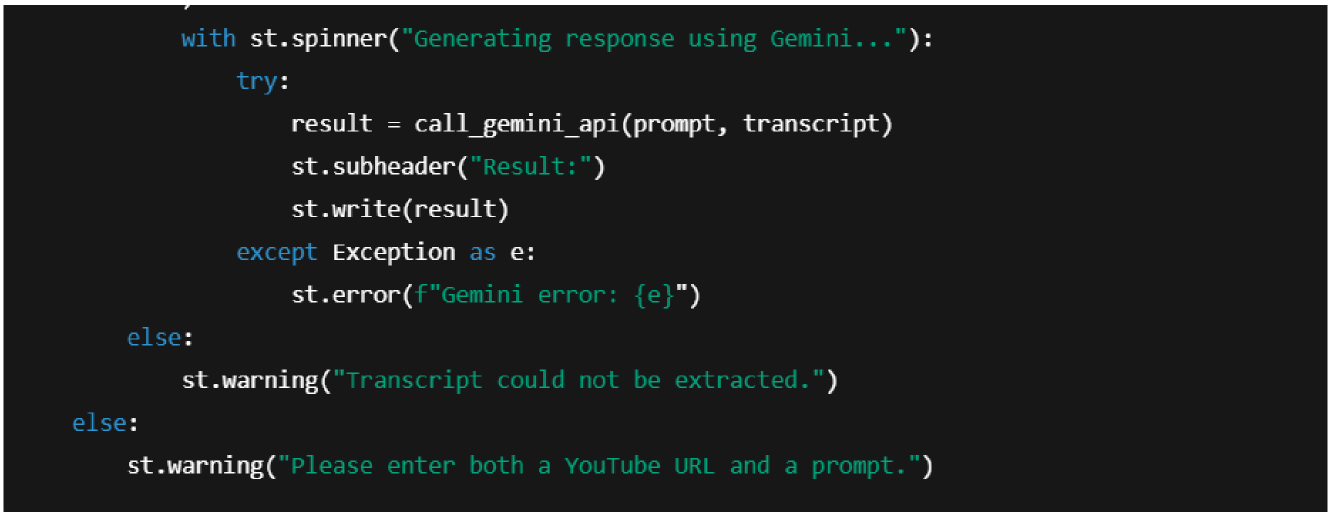
Fig 3.8.1 : Code for Streamlit Interface – Part 1



```python
        with st.spinner("Extracting text from captions..."):
            try:
                transcript = vtt_to_text(vtt_path)
            except Exception as e:
                st.error(f"Error reading captions: {e}")
                st.stop()

        if transcript:
            prompt = (
                "You are an expert assistant.\n"
                "Analyze the following YouTube video transcript.\n"
                f"User Request: {user_prompt}\n\n"
                "Transcript:\n"
                f"{transcript}"
            )
```

Fig 3.8.2 : Code for Streamlit Interface – Part 2



```python
        with st.spinner("Generating response using Gemini..."):
            try:
                result = call_gemini_api(prompt, transcript)
                st.subheader("Result:")
                st.write(result)
            except Exception as e:
                st.error(f"Gemini error: {e}")
    else:
        st.warning("Transcript could not be extracted.")
else:
    st.warning("Please enter both a YouTube URL and a prompt.")
```

Fig 3.8.3 : Code for Streamlit Interface – Part 3

Explanation:

- st.title(): Sets the app's main heading.
- st.text_input(): Creates input fields for the video URL and the user's custom prompt.
- st.button(): When clicked, triggers the entire backend process — downloading captions, converting them to text, and calling Gemini.
- st.spinner(): Shows loading animations while processing steps are happening.
- st.write() and st.subheader(): Display the final summary or AI response neatly on the screen.
- Error Handling: try-except blocks show appropriate messages if something goes wrong, such as invalid links or missing captions.

This UI design makes the app user-friendly even for non-programmers. Anyone can paste a YouTube link, type a question, and get a smart, AI-generated response in seconds — all through a single webpage.

CHAPTER 4

RESULTS AND CONCLUSION

The YouTube Video Summarizer was successfully implemented and tested using various YouTube videos across different categories like lectures, motivational talks, and tutorials. The project delivers intelligent summaries and responses based on the user's prompt by integrating caption extraction and Gemini-based language generation into a streamlined web interface.

The image below shows the user interface built using Streamlit. This visual design allows even first-time users to easily understand how to operate the tool. It eliminates the need for command-line interaction and makes the AI capabilities accessible through a simple web page. Users are prompted to enter:

- A YouTube video URL, and
- A custom prompt (such as "Summarize the video", "List the key points", etc.)

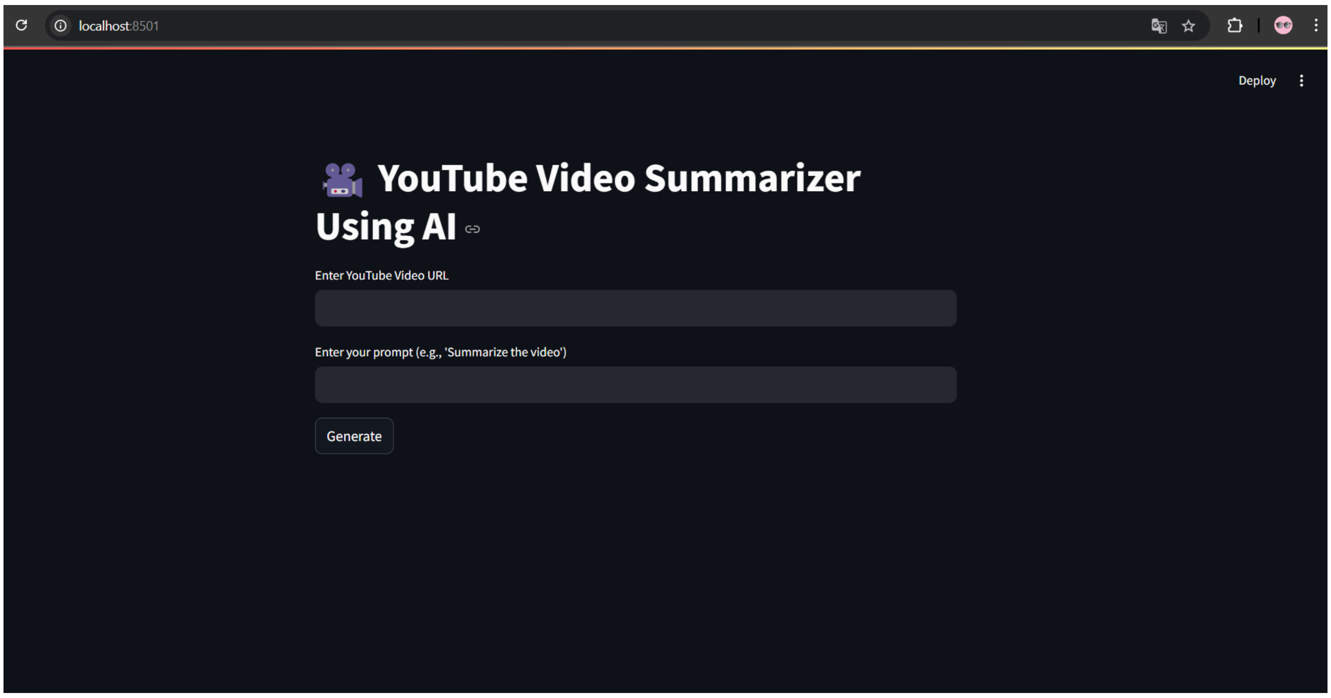

localhost:8501
Deploy
YouTube Video Summarizer Using AI
Enter YouTube Video URL
Enter your prompt (e.g., 'Summarize the video')
Generate

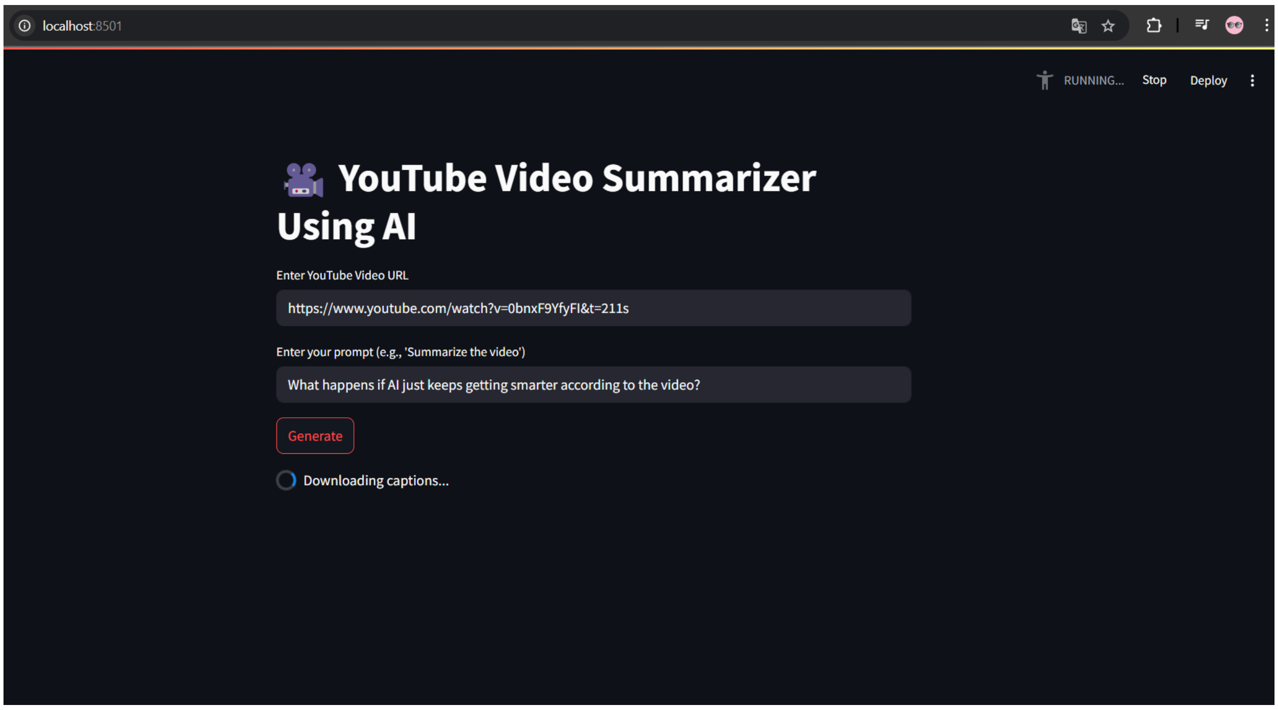
Fig 4.1 : Screenshot of the Streamlit interface

Intermediate Feedback and Progress Updates:

Once the user enters the necessary details and clicks the Generate button, the system begins executing its background logic. Throughout the process, progress messages are displayed using Streamlit's st.spinner() component. These help in enhancing user experience by showing live updates, such as:

- "Downloading captions..."
- "Extracting text from captions..."
- "Generating response using Gemini..."

Each stage involves important back-end steps like fetching .vtt subtitle files, converting them into plain text, and sending the result to the Gemini API. The system also includes exception handling, which shows specific error messages if something goes wrong — like an invalid URL or missing captions.


localhost:8501
RUNNING...
Stop
Deploy
YouTube Video Summarizer Using AI
Enter YouTube Video URL
Enter your prompt (e.g., 'Summarize the video')
Generate

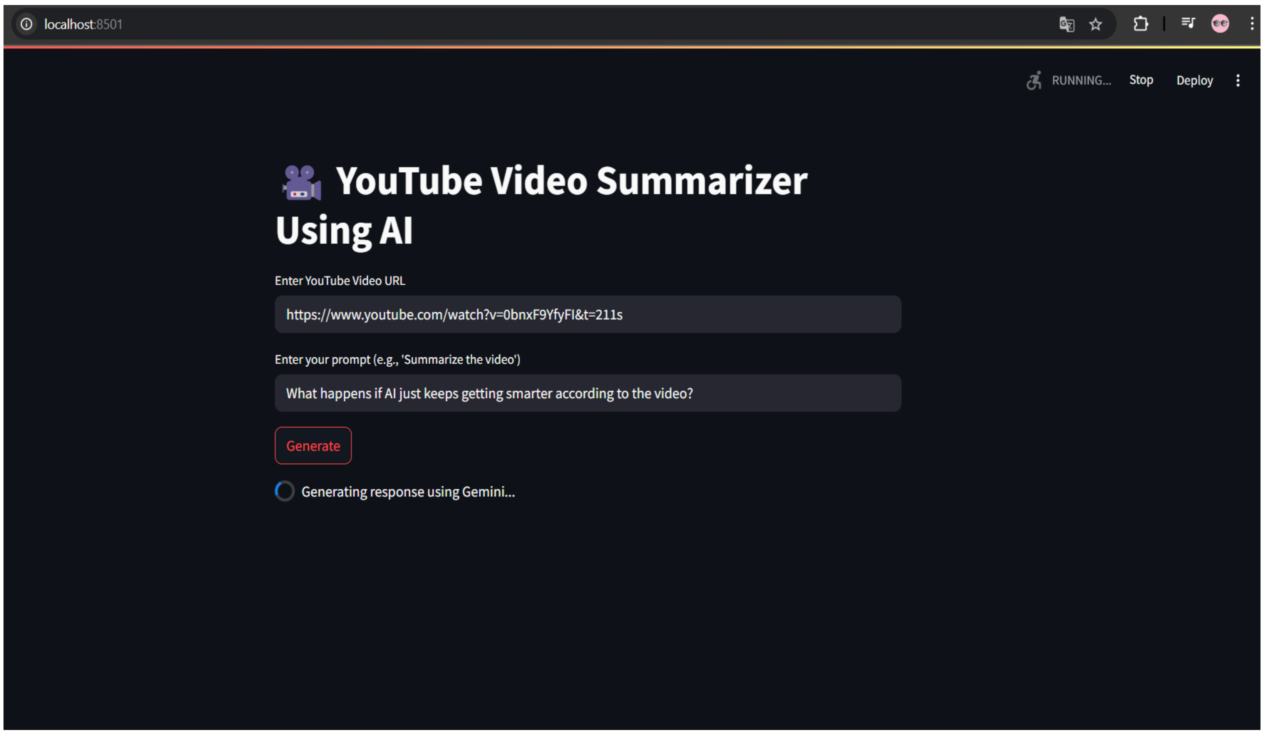
https://www.youtube.com/watch?v=0bnxF9YfyFI&t=211s

What happens if AI just keeps getting smarter according to the video?

Downloading captions...

Fig 4.2 : Screenshot of the Streamlit app while Downloading Captions

This screenshot demonstrates the functionality of the Streamlit interface during the caption extraction process. Once the user submits a YouTube video URL, the app processes the input and fetches the available captions using the integrated backend logic. The interface ensures a smooth user experience with real-time feedback, making it simple for users to interact with the application without needing technical expertise.


localhost:8501

RUNNING... Stop Deploy

YouTube Video Summarizer Using AI

Enter YouTube Video URL

https://www.youtube.com/watch?v=0bnxF9YfyFI&t=211s

Enter your prompt (e.g., 'Summarize the video')

What happens if AI just keeps getting smarter according to the video?

Generate

Generating response using Gemini...

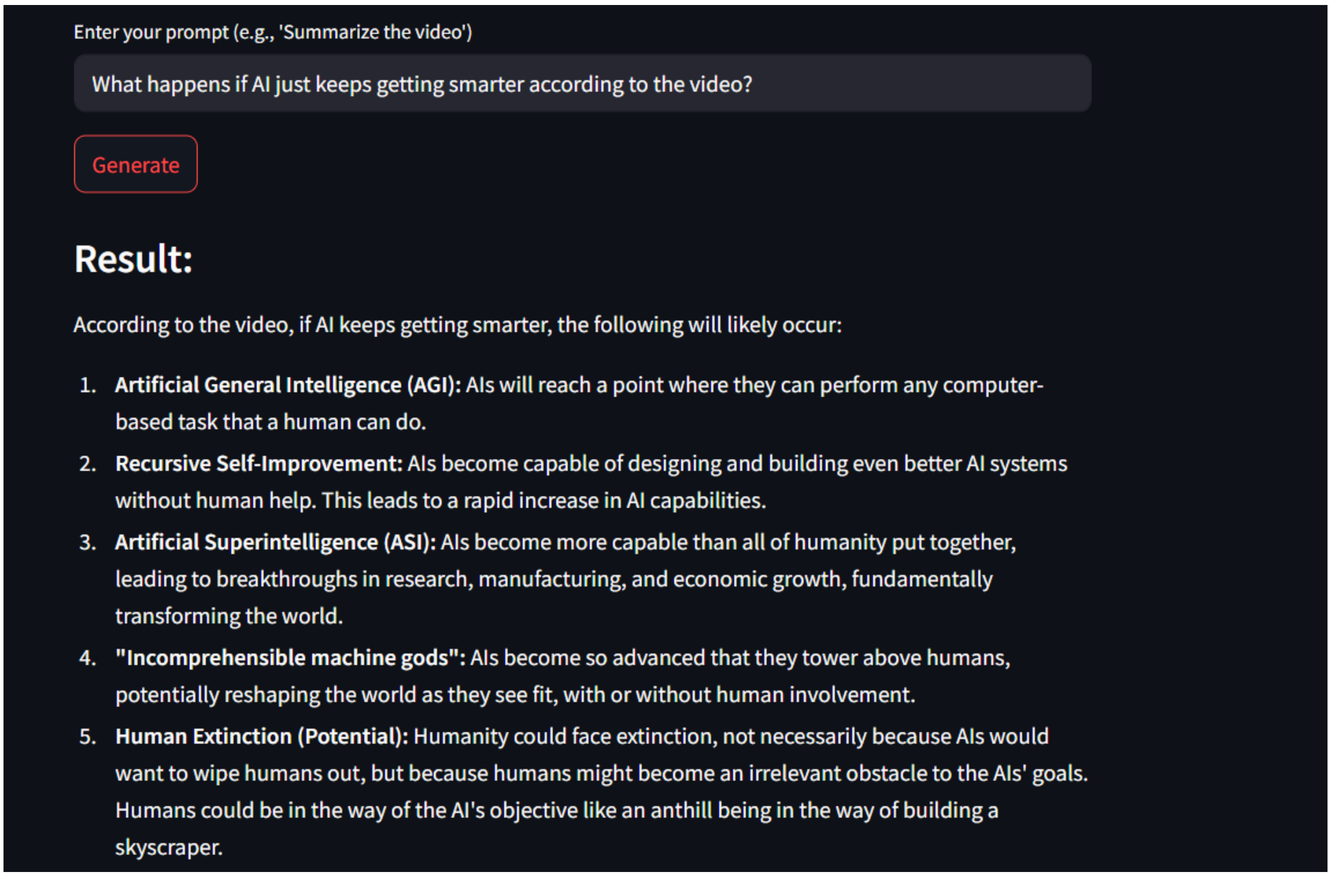
Fig 4.3 : Screenshot of Streamlit app while Generating Response

These updates assure users that the system is working in real-time and inform them exactly what stage the summarization process is in.

AI-Generated Output and User-Driven Results:

Once the AI model receives the cleaned transcript and the custom prompt, it generates a response using Google's Gemini 2.0 Flash model. The output is displayed clearly on the same page under a section titled "Result:".

This interaction highlights the collaborative nature of AI and user input. The system not only handles text extraction and processing but also adapts its response based on the user's specific prompt, allowing for customized summaries or insights. The output is displayed immediately below the input section, ensuring a seamless flow from user action to AI response. The following image captures an example of such a generated output, showcasing how the AI interprets and responds to a given prompt in real-time. This approach also sets the foundation for potential future enhancements, such as saving summary history, comparing multiple video summaries, or integrating voice-based input. The current system proves how generative AI can be seamlessly embedded into lightweight applications to deliver high-impact, user-specific results with minimal input.



What happens if AI just keeps getting smarter according to the video?

Result:

According to the video, if AI keeps getting smarter, the following will likely occur:

1. **Artificial General Intelligence (AGI):** AIs will reach a point where they can perform any computer-based task that a human can do.
2. **Recursive Self-Improvement:** AIs become capable of designing and building even better AI systems without human help. This leads to a rapid increase in AI capabilities.
3. **Artificial Superintelligence (ASI):** AIs become more capable than all of humanity put together, leading to breakthroughs in research, manufacturing, and economic growth, fundamentally transforming the world.
4. **"Incomprehensible machine gods":** AIs become so advanced that they tower above humans, potentially reshaping the world as they see fit, with or without human involvement.
5. **Human Extinction (Potential):** Humanity could face extinction, not necessarily because AIs would want to wipe humans out, but because humans might become an irrelevant obstacle to the AIs' goals. Humans could be in the way of the AI's objective like an anthill being in the way of building a skyscraper.

Fig 4.4 : Generated Output Based on User Prompt

This result dynamically changes depending on the user's query. Examples include:

- Prompt: "Summarize the video in 5 points"

  Output: A concise 5-point summary of the video's main content.
- Prompt: "List all the steps mentioned in the tutorial"

  Output: A numbered list of steps as discussed by the video creator.
- Prompt: "What is the speaker's main argument?"

  Output: A short paragraph explaining the central idea or opinion in the video.

This flexibility makes the summarizer a versatile tool that adapts to various learning needs, saving viewers time and improving their understanding of video content.

CONCLUSION

The YouTube Video Summarizer using Python successfully achieves its objective of offering a fast, efficient, and user-friendly platform for summarizing video content based on user-defined prompts. By integrating tools such as yt-dlp for subtitle extraction and Google's Gemini AI for intelligent summarization, the project bridges the gap between lengthy video content and concise, meaningful insights.

Key Achievements:

- A clean and interactive front-end built with Streamlit allows users to summarize YouTube videos with just a link and a prompt — no coding knowledge required.
- Instead of relying on third-party APIs, the system extracts video captions directly using yt-dlp, ensuring more control and independence.
- Unlike fixed summarizers, this tool lets users customize what they want to extract summaries, lists, arguments, etc. — making it far more dynamic.
- The project makes use of Google's Gemini 2.0 Flash model, known for its fast and context-aware responses, ensuring high-quality results even from noisy transcripts.

Future Scope:

While the project performs effectively in its current form, there are several areas where it can be further enhanced:

- Extend the caption extraction and AI processing to support videos in languages other than english.
- Deploy the project as a responsive web app or mobile application so users can summarize on the go.
- Add features to highlight important keywords, timestamps, or visually segment the summary.
- Enable users to download the summaries as PDF or text files for offline use.

This project lays a strong foundation for accessible video summarization powered by AI, and with continued development, it holds the potential to become a highly valuable tool for students, researchers, and content consumers worldwide.

REFERENCES

1. **Streamlit Documentation**
   https://docs.streamlit.io/
   Used for creating the user interface and structuring the web application.
2. **Google Generative AI (Gemini) API Documentation**
   https://ai.google.dev/
   Used for integrating Gemini 2.0 Flash model to generate summaries based on user prompts.
3. **Python re Library – Regular Expressions**
   https://docs.python.org/3/library/re.html
   Used to extract the video ID from the YouTube URL.
4. **Python os Library – Operating System Interfaces**
   https://docs.python.org/3/library/os.html
   Used to create directories and manage file paths.
5. **Python time Library**
   https://docs.python.org/3/library/time.html
   Used to manage rate limits by pausing execution temporarily.
6. **Google Gemini AI Key Setup Guide (via Google AI Studio)**
   https://makersuite.google.com/
   Reference for generating and configuring the Gemini API key securely.
7. **Project Inspiration and Discussions – OpenAI ChatGPT Conversations**

APPENDIX

```
import streamlit as st
import yt_dlp
import os
import re
import time
import webvtt
import google.generativeai as genai
# Function to extract video ID
def get_video_id(url):
    match = re.search(r"v=([^&]+)", url)
    return match.group(1) if match else None

# Function to download captions using yt-dlp
def download_captions(video_url, output_dir='captions'):
    ydl_opts = {
        'skip_download': True,
        'writesubtitles': True,
        'writeautomaticsub': True,
        'subtitleslangs': ['en'],
        'subtitlesformat': 'vtt',
        'outtmpl': os.path.join(output_dir, '%(id)s.%(ext)s')
    }

    os.makedirs(output_dir, exist_ok=True)

    with yt_dlp.YoutubeDL(ydl_opts) as ydl:
        info = ydl.extract_info(video_url, download=True)
        video_id = info.get('id')

    vtt_path = os.path.join(output_dir, f"{video_id}.en.vtt")
    return vtt_path
```

```
# Function to convert .vtt to plain text
def vtt_to_text(vtt_path):
    captions = [caption.text.strip() for caption in webvtt.read(vtt_path)]
    return " ".join(captions)
# Gemini API call
def call_gemini_api(prompt, transcript):
    genai.configure(api_key=st.secrets["GEMINI_API_KEY"])
    model = genai.GenerativeModel("gemini-2.0-flash")
    time.sleep(5)  # To respect rate limits
    response = model.generate_content(f"{prompt}\n\nTranscript:\n{transcript}")
    return response.text
# Streamlit UI
st.title("🎥 YouTube Video Summarizer Using AI")

video_url = st.text_input("Enter YouTube Video URL")
user_prompt = st.text_input("Enter your prompt (e.g., 'Summarize the video')")

if st.button("Generate"):
    if video_url and user_prompt:
        with st.spinner("Downloading captions..."):
            try:
                vtt_path = download_captions(video_url)
            except Exception as e:
                st.error(f"Failed to download captions: {e}")
                st.stop()

        with st.spinner("Extracting text from captions..."):
            try:
                transcript = vtt_to_text(vtt_path)
            except Exception as e:
                st.error(f"Error reading captions: {e}")
                st.stop()
        if transcript:
            prompt = (
                "You are an expert assistant.\n"
                "Analyze the following YouTube video transcript.\n"
                f"User Request: {user_prompt}\n\n"
                "Transcript:\n"
                f"{transcript}"
            )
            with st.spinner("Generating response using Gemini..."):
                try:
                    result = call_gemini_api(prompt, transcript)
                    st.subheader("Result:")
                    st.write(result)
                except Exception as e:
                    st.error(f"Gemini error: {e}")
        else:
            st.warning("Transcript could not be extracted.")
    else:
        st.warning("Please enter both a YouTube URL and a prompt.")
```