INDUSTRY ORIENTED MINI PROJECT

Report

On

AI-POWERED SYSTEM FOR TRANSLATING EDUCATIONAL CONTENT AND GENERATING REGIONALLY ACCURATE SUBTITLES

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

BACHELOR OF TECHNOLOGY

In

INFORMATION TECHNOLOGY

By

Abdul Muqtadir – 22261A1201

Under the guidance of

Dr. D. Vijaya Lakshmi

Professor & HOD, Department of IT

DEPARTMENT OF INFORMATION TECHNOLOGY

MAHATMA GANDHI INSTITUTE OF TECHNOLOGY

(AUTONOMOUS)

(Affiliated to JNTUH, Hyderabad; Eight UG Programs Accredited by NBA; Accredited by NAAC with 'A++' Grade)

Gandipet, Hyderabad, Telangana, Chaitanya Bharati (P.O), Ranga Reddy District, Hyderabad– 500075, Telangana

2024-2025

CERTIFICATE

This is to certify that the **Industry Oriented Mini Project** entitled **AI-POWERED SYSTEM FOR TRANSLATING EDUCATIONAL CONTENT AND GENERATING REGIONALLY ACCURATE SUBTITLES** submitted by **Abdul Muqtadir (22261A1201),** in partial fulfilment of the requirements for the Award of the Degree of Bachelor of Technology in Information Technology as specialization is a record of the bona fide work carried out under the supervision **of Dr. D. VIJAYA LAKSHMI**, and this has not been submitted to any other University or Institute for the award of any degree or diploma.

Internal Supervisor:	**IOMP Supervisor:**
Dr. D. VIJAYA LAKSHMI	**Dr. U. Chaitanya**
Professor & HOD	Assistant Professor
Dept. of IT	Dept. of IT

EXTERNAL EXAMINAR	**Dr. D. Vijaya Lakshmi**
	Professor and HOD
	Dept. of IT

DECLARATION

We hear by declare that the **Industry Oriented Mini Project** entitled **AI-POWERED SYSTEM FOR TRANSLATING EDUCATIONAL CONTENT AND GENERATING REGIONALLY ACCURATE SUBTITLES** is an original and bonafide work carried out by us as a part of fulfilment of Bachelor of Technology in Information Technology, Mahatma Gandhi Institute of Technology, Hyderabad, under the guidance of **Dr. D. Vijaya Lakshmi, Professor & HOD, Department of IT, MGIT**.

No part of the project work is copied from books /journals/ internet and wherever the portion is taken, the same has been duly referred in the text. The report is based on the project work done entirely by us and not copied from any other source.

Abdul Muqtadir – 22261A1201

ACKNOWLEDGEMENT

The satisfaction that accompanies the successful completion of any task would be incomplete without introducing the people who made it possible and whose guidance and encouragement crowns all efforts with success. They have been a guiding light and source of inspiration towards the completion of the **Industry Oriented Mini Project.**
We would like to express our sincere gratitude and indebtedness to our project guide **Dr. D. Vijaya Lakshmi , HOD**, Dept. of IT, who has supported us throughout our project with immense patience and expertise.
We are also thankful to our honourable Principal of MGIT **Prof. G. Chandramohan Reddy** and **Dr. D. Vijaya Lakshmi**, Professor and HOD, Department of IT, for providing excellent infrastructure and a conducive atmosphere for completing this **Industry Oriented Mini Project** successfully.
We are also extremely thankful to our Project Coordinator(s)**Dr U. Chaitanya**, Assistant Professor, Department of IT, and senior faculty **Mrs. B. Meenakshi** Department of IT for their valuable suggestions and guidance throughout the course of this project.
We convey our heartfelt thanks to the lab staff for allowing us to use the required equipment whenever needed.
Finally, we would like to take this opportunity to thank our families for their support all through the work. We sincerely acknowledge and thank all those who gave directly or indirectly their support for completion of this work.

Abdul Muqtadir - 22261A1201

ABSTRACT

In a diverse and multilingual nation, accessibility to educational content remains a challenge due to language barriers. Addressing this issue, the present project introduces an **AI-powered system** capable of automatically transcribing, translating, and generating regionally accurate subtitles for educational videos. The system is designed to cater to learners across linguistic boundaries by converting spoken content into subtitles in the viewer's preferred local language, thus promoting inclusive and equitable education.

The solution integrates **AssemblyAI's speech-to-text API**, **Google Translate**, and **FFmpeg**, combined into a Flask-based web application that allows users to either upload videos or input YouTube links. The audio is extracted, transcribed with **speaker labels**, translated into the target language (e.g., Hindi, Telugu, etc.), and then embedded as synchronized **.srt subtitles** within the original video. This process ensures accurate speech segmentation and preserves the speaker-specific context.

To assess performance, the system was tested across various accents, speech speeds, and video formats. The modular pipeline ensures scalability and adaptability to new languages and domains. By automating transcription and translation, this system significantly reduces manual effort while enhancing accessibility for non-English-speaking learners.

Ultimately, this project contributes to **bridging the digital language divide**, empowering educational institutions, content creators, and learners to communicate and engage more effectively. It stands as a step toward democratizing knowledge in the age of AI.

.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

INTRODUCTION

1.1 MOTIVATION

With the growing reach of digital education, **video-based learning platforms** like YouTube have become essential tools for knowledge dissemination. However, a major barrier to inclusive learning remains: **language diversity**. Millions of students and lifelong learners struggle to understand educational videos that are not in their native languages. In multilingual regions, this language gap contributes to educational inequality, limiting access to critical information and resources.

Traditional methods of subtitling, such as manual transcription and translation, are time-consuming, expensive, and difficult to scale. Moreover, these approaches often fail to preserve speaker context and accuracy in dynamic conversational content, especially in multi-speaker educational scenarios like lectures, interviews, and webinars. The **lack of automated, accurate, and regionally sensitive subtitle generation tools** leaves a significant portion of educational content inaccessible to non-English speakers.

This project is driven by the need to bridge this **digital language divide** by developing an **AI-powered system** that automates the end-to-end process of subtitle generation. By combining advanced speech recognition, real-time translation, and subtitle embedding, the system aims to make educational videos **regionally relevant** and **linguistically accessible** to a broader audience. The project focuses not just on translation, but on retaining clarity, speaker distinction, and cultural context — essential factors in delivering meaningful learning experiences.

1.2 PROBLEM STATEMENT

The increasing reliance on video-based educational content across platforms like YouTube, Coursera, and edX has significantly widened access to learning. However, **language remains a major barrier** for non-native speakers and learners from linguistically diverse backgrounds. Most educational videos are published in a single language, primarily English, leaving a large population unable to fully understand or engage with the content. While subtitles can help, **manual subtitle creation and translation** is labor-intensive, time-consuming, and often inaccurate—especially when dealing with subject-specific terminology or multiple speakers.

Current solutions for subtitle generation either rely on human intervention or use basic automatic tools that lack accuracy, context-awareness, and regional language support. These systems often **fail to distinguish between speakers**, ignore regional dialects, or generate grammatically incorrect translations, which compromises the educational value of the content. Furthermore, many existing systems **do not support real-time processing or seamless integration** with multilingual content pipelines.

The gap is particularly evident in educational contexts, where **accuracy, speaker clarity, and timely availability** of subtitles are crucial for comprehension and accessibility. Without scalable and intelligent subtitle generation, millions of learners are left behind due to linguistic barriers.

This project aims to overcome these limitations by designing a **robust, scalable, and context-aware AI system** that automates the transcription, translation, and subtitle generation process for educational videos. Leveraging state-of-the-art technologies such as **Whisper for speech recognition**, **Hugging Face Transformers for translation**, and **FFmpeg for subtitle embedding**, the system will generate **accurate, speaker-labeled, and regionally adapted subtitles**. This will not only bridge the language gap but also democratize access to high-quality educational content worldwide.

1.3 EXISTING SYSTEM

Existing subtitle generation systems typically fall into two categories: **manual transcription and translation services**, or **basic automated tools** integrated within platforms like YouTube or commercial transcription software. These systems provide limited functionality and are often designed for **generic use cases**, not specifically for educational content.

YouTube's auto-captioning, for example, offers real-time subtitles, but struggles with subject-specific terminology, **multi-speaker differentiation**, and often **produces incorrect or contextually inaccurate translations**. Other commercial tools like Otter.ai and Sonix provide transcription services but require **separate manual steps** for translation and subtitle formatting. Moreover, these tools are **often limited in their language support**, especially when it comes to regional Indian or African languages.

Most of these systems also **lack integration capabilities** with custom educational workflows, do not offer open-source flexibility, and fail to preserve speaker context—an essential feature for understanding lectures, debates, or interviews. Furthermore, **very few systems support end-to-end automation** from video input to translated, embedded subtitle output.

In summary, current systems are not designed to handle the **accuracy, automation, and adaptability** required for multilingual educational content delivery. This creates a significant opportunity to develop an intelligent solution that **fills the gap**.

1.3.1 Limitations

- **Inaccurate Speaker Attribution**: Many current systems struggle to differentiate between multiple speakers, leading to incorrect or confusing subtitles that affect the clarity of educational content.
- **Limited Language Support**: Existing tools often support only a narrow set of major global languages, excluding many regional or less-resourced languages, which restricts accessibility for diverse learners.
- **Lack of Context-Aware Translation**: Automated subtitle systems typically fail to preserve the context or technical meaning of educational terms, resulting in translations that are literal and often inaccurate in an academic setting.

1.4 PROPOSED SYSTEM

The proposed system aims to automate the process of generating translated subtitles with speaker identification for video content, enhancing accessibility and user engagement across linguistic boundaries. This solution addresses the manual and time-intensive limitations of traditional subtitling and translation workflows. By integrating AssemblyAI for transcription and speaker labeling, and Google Translate for multilingual translation, the system ensures high-quality, speaker-specific subtitles that are both accurate and context-aware.

A key innovation of the system is its ability to accept either uploaded videos or YouTube URLs, enabling flexibility in content sources. Once a video is provided, the system extracts audio using FFmpeg and transcribes it using AssemblyAI's API with speaker labels. The transcribed segments are then broken into manageable chunks and translated into the selected language. Finally, the translated and speaker-labeled content is formatted into .srt subtitle files and embedded back into the original video using FFmpeg.

The system supports various languages and maintains speaker distinction throughout the translation process, ensuring clarity and authenticity in multi-speaker conversations. This makes it especially valuable for interviews, lectures, podcasts, and multilingual audiences. By offering an end-to-end solution within a Flask web application, the platform ensures a seamless user experience from video input to downloadable output.

1.4.1 ADVANTAGES

- **Multilingual Support**: Generates subtitles in multiple languages to reach wider audiences.
- **Speaker Labeling**: Maintains clarity by identifying and labeling different speakers.
- **Full Automation**: Handles everything from audio extraction to subtitle embedding.
- **Easy Input Options**: Supports both video uploads and YouTube links.
- **Scalable Design**: Built with modular components for easy updates and expansion.

1.5 OBJECTIVES

- **Auto Subtitle Generation**: Create translated subtitles with speaker labels.
- **Fast Processing**: Ensure quick and efficient subtitle creation.
- **Simple Web Interface**: Let users easily upload videos or paste links.
- **Multi-Format Output**: Provide subtitles as both .srt files and embedded videos.
- **Wider Accessibility**: Make content understandable across languages and regions.

1.6 HARDWARE AND SOFTWARE REQUIREMENTS

Software Requirements

1. Python (Programming Language)

- The entire application is built in Python.
- Major version recommended: **Python 3.8+**

2. Flask (Web Framework)

- Used to create the web interface for users to upload/download videos and interact with the system.
- Provides routing (@app.route) and handles HTTP requests.

3. AssemblyAI (Speech-to-Text API)

- Used for **AI-based audio transcription** with support for speaker labeling.
- Requires an internet connection and API key.
- Library used: assemblyai

4. yt-dlp (YouTube Downloader)

- A powerful tool for downloading videos from YouTube and other platforms.
- Python library: yt_dlp
- Replaces older youtube-dl.

5. FFmpeg (Multimedia Framework)

- Used for:
    - Extracting audio from video.
    - Embedding subtitles into the video.
- Called via subprocess from Python.
- Required to be **installed and added to system PATH**.

6. GoogleTranslator from Deep Translator

- Translates subtitles into different languages using Google Translate.
- Library: deep_translator
- Internally may require an active internet connection to hit the Google Translate service.

7. HTML & CSS Templates

- files: templates/index.html, templates/result.html
- Used for form input and result display in the browser.

8. Standard Python Libraries

Library	Purpose
os	File path handling, directory creation.
re	Regular expressions for sanitizing filenames.
subprocess	Running shell commands like FFmpeg.
datetime	For formatting subtitle timecodes.

Hardware Requirements

1. **Operating System:**
   Compatible with **Windows**, **macOS**, or **Linux**.
2. **Processor:**
   Minimum **Intel i5** or equivalent for efficient handling of video/audio processing.
3. **RAM:**
   At least **8GB RAM** is recommended to ensure smooth processing of large video files and multitasking during subtitle generation.
4. **Storage:**
   At least **25GB of free disk space** for handling media files, processing outputs, and software installations.

1. LITERATURE SURVEY

Moryossef et al. introduced *sign.mt*, an open-source application enabling real-time multilingual translation between spoken and signed languages using modular pipelines. The system supports both spoken-to-signed and signed-to-spoken translations, using components such as speech recognition, language identification, normalization, sentence splitting, and SignWriting-based pose generation. While the system enables low-latency, offline functionality with photo-realistic avatars, the modularity comes at the cost of dependency on pre-existing models, which may vary in quality and accuracy. Despite its limitations, *sign.mt* significantly contributes to democratizing communication for the deaf community and encourages open research and extensibility for diverse sign language systems.[1]

Moryossef's doctoral thesis further explores the theoretical and practical frameworks for Sign Language Processing (SLP), emphasizing the use of SignWriting as a universal, intermediate representation. The work identifies key challenges such as the absence of standardized written forms for signed languages, and the difficulties in modeling simultaneous, multidimensional linguistic structures. The research provides foundational datasets, glossing tools, and animation benchmarks to aid the computational modeling of signed languages. While the proposed solutions offer scalability and generalizability, the complexity of real-time production and translation pipelines continues to require significant computational resources and domain-specific adaptation.[2]

Mocanu and Tapu developed a system for automatic subtitle synchronization and positioning tailored for deaf and hearing-impaired individuals. Their solution uses speech recognition to align subtitles with audio content, addressing latency issues in live broadcasts. Additionally, they implemented an intelligent positioning mechanism that avoids overlap with important visual elements in videos. While the system achieved over 90% accuracy in evaluations across various video genres, its dependency on ASR quality and genre-specific constraints limits its robustness in noisy or unpredictable live environments.[3]

Che et al. proposed a bilingual subtitle generation framework for lecture videos, integrating Automatic Speech Recognition (ASR), Sentence Boundary Detection (SBD), and Machine Translation (MT). Their study showed that while English subtitles generated by ASR were already fairly accurate, the quality of machine-translated subtitles (e.g., English to Chinese) lagged behind. Nonetheless, using auto-generated subtitles as a draft significantly reduced human editing time and improved accuracy, making it a valuable aid for educational content creation. However, the effectiveness of their method was limited by the performance of the MT system, especially for complex or domain-specific language.[4]

Moryossef et al. proposed *sign.mt*, a real-time sign language machine translation system that supports both spoken-to-signed and signed-to-spoken communication. The platform offers a modular pipeline integrating speech recognition, language identification, normalization, sentence segmentation, and sign language video rendering. Aimed at bridging accessibility gaps, *sign.mt* supports multilingual and bidirectional translation while remaining open-source and customizable. Its modular design allows researchers to integrate and test new models quickly. However, the reliance on currently available models limits the consistency of output quality across different languages and modalities, and achieving realistic, low-latency sign animations for real-world usage remains a complex challenge.[5]

Table 2.1 Literature Survey

S.No	Author(s) & Year	Publisher	Methodology / Algorithm / Techniques Used	Merits	Demerits	Research Gaps
1	Amit Moryossef, PhD Thesis (2023)	IEEE	Sign Language Processing with Unified intermediate glosses, SignWriting notation, segmentation, multilingual SLP baseline gloss-to-sign models	Enables segmentation and multilingual gloss annotation	Linear glosses lose simultaneity info; SignWriting not widely adopted	Require better tools for annotation and real-world deployment
2	Amit Moryossef et al. (2023)	IEEE	Modular pipeline for real-time sign language translation using STT, SignWriting, Pose Estimation, GANs	Multilingual, modular, customizable avatar rendering components	Incomplete SignWriting-to-Pose model; refinement needed	Need for generalized models across sign languages; expand SignWriting datasets
3	Srikar Pulipaka et al. (2021)	IEEE	IBM Watson STT + Google Translate + Flite TTS (English to Indian Languages)	Avg. accuracy 79%; 91% single-video accuracy; useful for visually impaired	Better TTS needed for regional dialects; lack of robust open-source TTS for Indian languages	Lack of contextual translation
4	Bogdan Mocanu & Ruxandra Tapu (2019)	IEEE ACCESS	ASR + Subtitle/Audio Synchronization + Semantic Alignment + Subtitle Positioning	>90% accuracy in user-validated UI for hearing impaired	ASR accuracy limited by language and noise; no real-time implementation	Doesn't work for live subtitles
5	Chen Yao et al. (2018)	IEEE	Subtitle generation using ASR, SBD (DNN-based), MT	Saves 54% of human effort; 54.3% reduction in error	Low effectiveness in English to Chinese MT quality	SBD tuning needed for various content types

2. ANALYSIS AND DESIGN

In today’s increasingly digital and multilingual world, accessibility to video content across different languages and regions is crucial. This project aims to develop an intelligent and automated subtitle generation system that simplifies the process of creating accurate, translated, and embedded subtitles from any YouTube or uploaded video. The goal is to help content creators, educators, and viewers overcome language barriers, enhance accessibility, and improve audience engagement using AI.

The system combines powerful speech recognition, natural language processing, and multimedia processing tools to deliver accurate transcription and seamless subtitle generation. At its core, the project integrates **AssemblyAI** to extract precise transcriptions with **speaker labels**, along with **Google Translate API** to translate these transcripts into the user's desired language. It then uses **FFmpeg** to embed the subtitles back into the video, enabling direct downloads of fully subtitled media.

Users can either provide a **YouTube video link** or upload a **local video file**. The system handles audio extraction, transcription, translation, subtitle generation, and embedding, all while maintaining a smooth and intuitive user interface built using **Flask**. The final output includes a downloadable video with hardcoded subtitles and an optional .srt file for additional flexibility.

3.1 MODULES

1. User Interaction Module

- **Functions:** Allows users to input a YouTube link or upload a video file, select a target subtitle language, choose subtitle format (hardcoded or SRT), and download the final output.
- **Input:** Video file or YouTube URL, selected language.
- **Output:** Subtitled video with embedded captions or downloadable SRT file.

2. Preprocessing Module

- **Functions:** Extracts audio from the video using FFmpeg and prepares it for transcription. Ensures audio is in the correct format (e.g., mono, 16kHz) to meet model requirements.
- **Input:** Raw video file or video stream from YouTube.

- **Output:** Cleaned and formatted audio file ready for transcription.

3. Transcription and Translation Module

- **Functions:** Transcribes speech using AssemblyAI with speaker labels and timestamps. The transcribed text is then translated into the selected language using Google Translate.
- **Input:** Processed audio file.
- **Output:** Translated subtitles with timestamped speaker-labeled segments.

4. Subtitle Generation Module

- **Functions:** Converts translated transcript into subtitle formats such as .srt, synchronizing timestamps accurately.
- **Input:** Translated text with timestamps.
- **Output:** Subtitle file (.srt) ready for embedding.

5. Video Rendering Module

- **Functions:** Embeds subtitles into the video using FFmpeg based on user choice (hardcoded subtitles or sidecar SRT). Supports subtitle styling and placement options.
- **Input:** Original video file and subtitle file.
- **Output:** Final downloadable video with embedded subtitles.

3.2 ARCHITECTURE

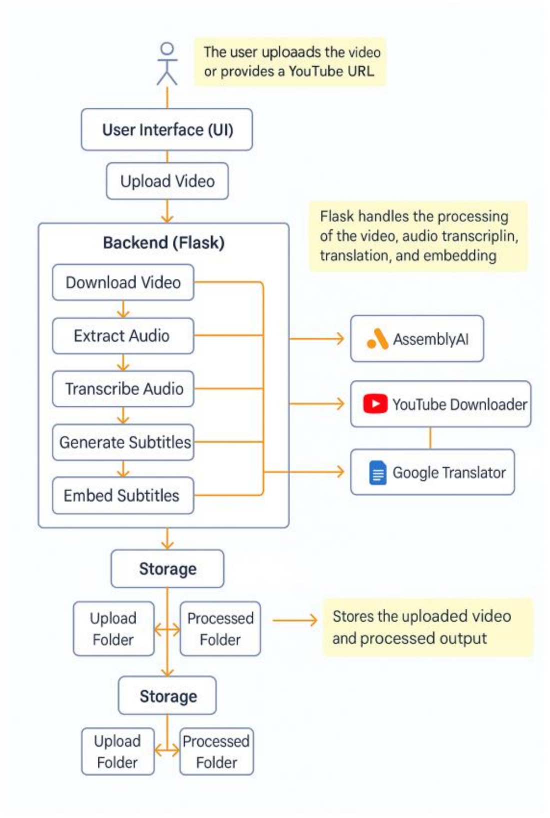


Fig. 3.2.1 Architecture of the AI-Powered Subtitle Generation and Translation System

The architecture of the system, as shown in **Fig. 3.2.1**, is designed for generating and translating subtitles for video content using artificial intelligence. It begins with the **Data Acquisition component**, which handles either a video file uploaded by the user or a YouTube video link. The system uses **FFmpeg** to extract audio from the video, converting it into a format suitable for processing.

In the **Main Processing Phase**, the extracted audio is sent to the **AssemblyAI API**, which performs advanced speech-to-text transcription. This transcription includes **speaker labels** and **timestamps**, allowing the system to distinguish between speakers and accurately time the subtitles. The output is then passed to the **Translation Engine**, where the text is translated into the user's selected language using the **Google Translate API**.

The translated subtitles are then formatted into the **.srt (SubRip Subtitle)** format in the **Subtitle Generation module**, preserving speaker separation and timing alignment. These subtitle files are either embedded directly into the video or provided as separate downloads.

The **Main Website Interface** plays a critical role in user interaction. It allows users to upload or link videos, select the output language, and choose whether they want subtitles hardcoded into the video or as a standalone file. The interface also shows progress and success alerts during processing and provides download options once the task is complete.

Users interact with the system via the **User Input Panel**, where they specify the language for translation and submit the video file or link. The system ensures high accessibility, offering a simple yet powerful interface for multilingual video processing.

The system employs multiple components—**FFmpeg for audio/video handling**, **AssemblyAI for transcription**, and **Google Translate for language conversion**—working in harmony to create accurate, speaker-labeled, and language-translated subtitles. This modular design ensures flexibility, real-time adaptability, and a smooth user experience.

3.3 DATA FLOW DIAGRAM

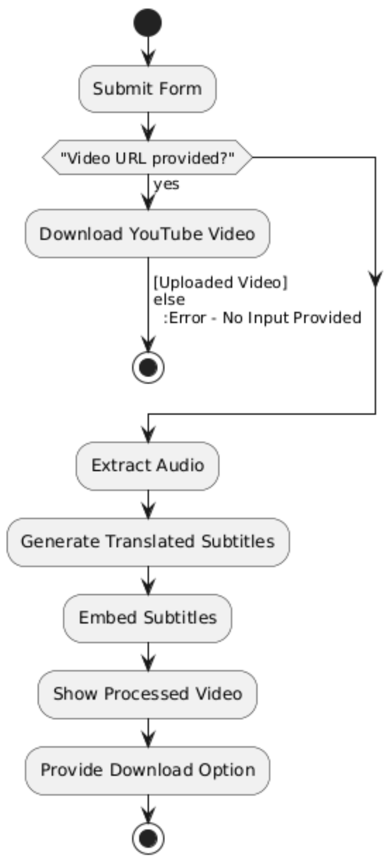


Fig. 3.3.1 Data Flow Diagram

This **Data Flow Diagram (Fig. 3.3.1)** represents a system for generating and translating subtitles for video content using artificial intelligence. The process begins with the **user uploading a video file or providing a YouTube link**, along with selecting the desired output language for translation.

The provided video input is passed to a **processing system** that first uses **FFmpeg** to extract the audio stream. This audio is then sent to the **AssemblyAI transcription service**, which returns the **speech-to-text output**, enriched with **timestamps** and **speaker labels** to identify who is speaking and when.

Next, the transcribed text is passed to the **translation module**, where it is translated into the selected language using the **Google Translate API**. The translated text is formatted into **.srt subtitle files**, maintaining proper speaker separation and synchronization based on the original timestamps.

These generated subtitles are then either **embedded back into the video** using FFmpeg or provided as **downloadable files**, based on user preference. Throughout the process, **real-time feedback** is displayed on the frontend interface, allowing users to track progress.

Finally, the results—including the **translated video with embedded subtitles** or a **standalone .srt file**—are presented to the user. The system ensures smooth data flow, accurate speech recognition, and reliable multilingual translation to support accessibility across languages and platforms.

3.4 UML DIAGRAMS

3.4.1 USE CASE DIAGRAMS

A use case diagram is a visual tool used to represent the interaction between external actors and the system, illustrating how users engage with the functionalities provided by the application. It plays a crucial role in the analysis and design phase of system development, helping to identify user requirements and system boundaries.

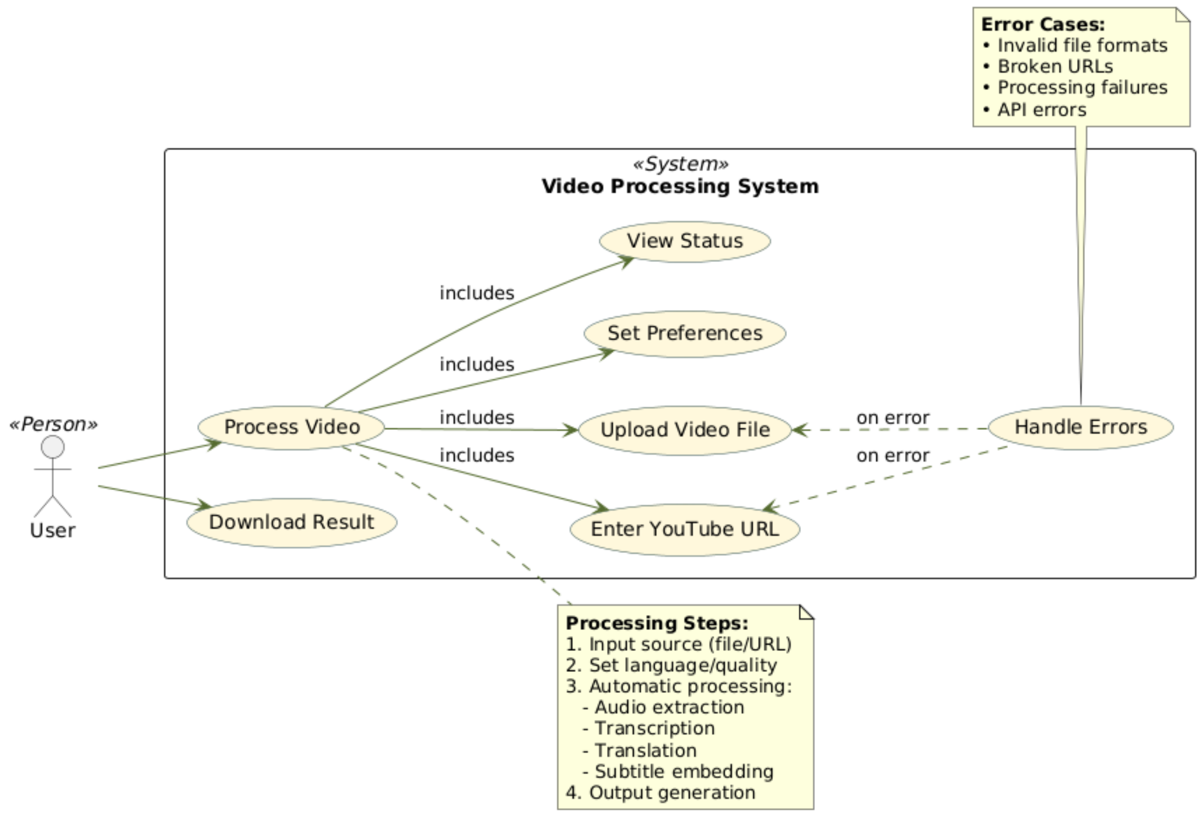


Fig. 3.4.1.1 Use Case Diagram

Actors

- **User**

  Represents the individual who interacts with the AI Subtitle Generator to upload videos, select languages, and download results.
- **System**

  Refers to the backend engine that performs transcription, translation, subtitle generation, and video rendering.

Use Cases

- **Upload Video / Enter YouTube Link**
  The user provides a video file or pastes a YouTube link to initiate subtitle generation.
- **Select Language**
  The user selects the target language for subtitle translation from the available options.
- **Transcribe and Translate Audio**
  The system transcribes the spoken content using a speech-to-text model (e.g., AssemblyAI) and then translates it using a multilingual model (e.g., Google Translate or Hugging Face Transformers).
- **Generate Subtitles**
  The system creates subtitle files (e.g., SRT) based on the translated transcription and aligns them with timecodes.
- **Embed Subtitles into Video**
  The system embeds the generated subtitles back into the video using tools like FFmpeg, producing a downloadable output.
- **Download Subtitles or Subtitled Video**
  The user downloads either the subtitle file or the full video with embedded translated subtitles.

3.4.2 CLASS DIAGRAM

A class diagram is a static structural diagram that outlines the blueprint of the AI Subtitle Generator system, representing its core components, their attributes, behaviours (methods), and interrelationships, as illustrated in **Fig. 3.4.2.1**. This diagram plays a vital role in object-oriented design, helping to visualize how different classes within the system interact and contribute to the overall functionality—from video input handling and transcription to translation, subtitle generation, and final video rendering.

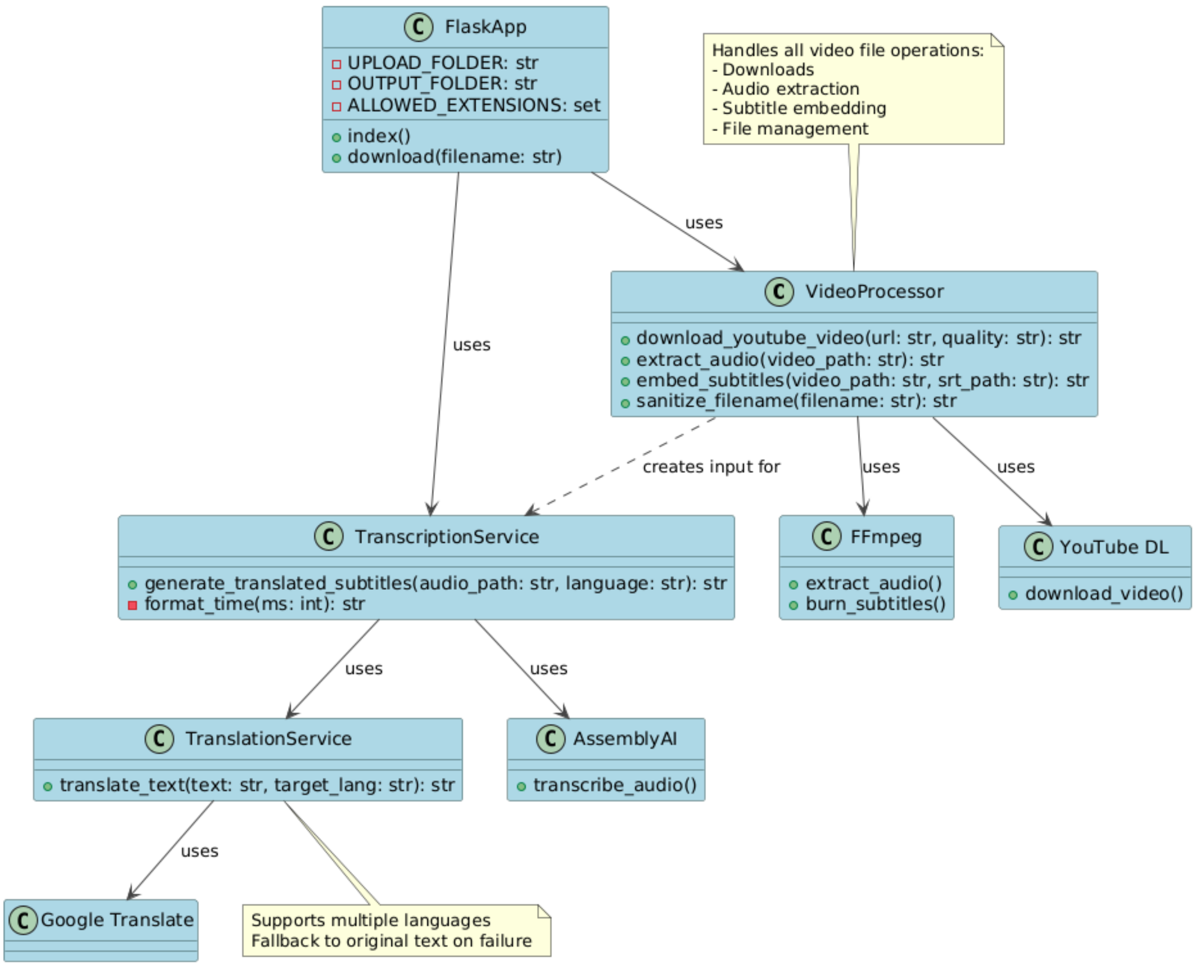

FlaskApp
UPLOAD_FOLDER: str
OUTPUT_FOLDER: str
ALLOWED_EXTENSIONS: set
index()
download(filename: str)
Handles all video file operations:
- Downloads
- Audio extraction
- Subtitle embedding
- File management
uses
uses
VideoProcessor
download_youtube_video(url: str, quality: str): str
extract_audio(video_path: str): str
embed_subtitles(video_path: str, srt_path: str): str
sanitize_filename(filename: str): str
creates input for
uses
uses
TranscriptionService
generate_translated_subtitles(audio_path: str, language: str): str
format_time(ms: int): str
FFmpeg
extract_audio()
burn_subtitles()
YouTube DL
download_video()
uses
uses
TranslationService
translate_text(text: str, target_lang: str): str
AssemblyAI
transcribe_audio()
uses
Google Translate
Supports multiple languages
Fallback to original text on failure

Fig. 3.4.2.1 Class Diagram

Relationships

- User → **System Interface:**
The user uploads videos or inputs YouTube URLs through the system interface.

- System **Interface → Database:**
The interface stores user data, video metadata, and subtitle files in the database.

- System **Interface → Audio Processing Module:**
The audio processing module extracts audio from the video provided by the user.

- Audio **Processing Module → Transcription Module:**
Extracted audio is sent to the transcription module for speech-to-text conversion.

- Transcription **Module → Translation Module:**
The transcript is forwarded to the translation module for language translation if needed.

• Subtitle **Generator → Video Processing Module:**
Generated subtitles are embedded into the video by the video processing module for output.

System Flow

• The user uploads a video or provides a YouTube URL via the system interface.

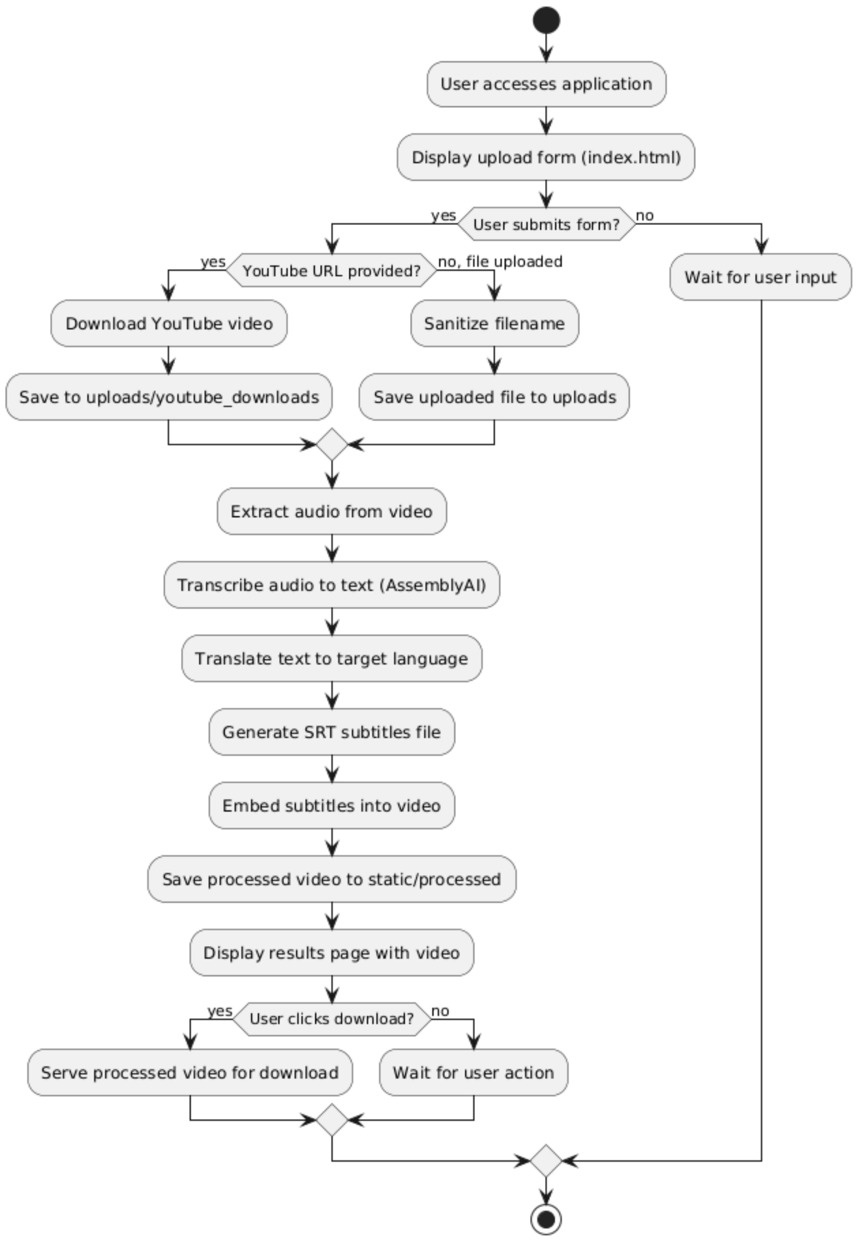
• The audio is extracted and transcribed into text by the transcription module.

• The transcript is translated (if required) and converted into timed subtitle files.

• Subtitles are embedded into the video, and the final subtitled video is presented to the user.

3.4.3 ACTIVITY DIAGRAM FOR PREDICTING PRICE OF COIN

An Activity Diagram is a behavioral diagram in Unified Modeling Language (UML) that illustrates the flow of control or data within a system, as shown in Fig. 3.4.3.1. It highlights the sequence of actions or activities involved in a particular process or workflow. Activity diagrams are widely used to model system processes, workflows, or any sequential operations, providing a clear visualization of how tasks progress step-by-step.



User accesses application
Display upload form (index.html)
User submits form?
yes
no
YouTube URL provided?
yes
no, file uploaded
Wait for user input
Download YouTube video
Save to uploads/youtube_downloads
Sanitize filename
Save uploaded file to uploads
Extract audio from video
Transcribe audio to text (AssemblyAI)
Translate text to target language
Generate SRT subtitles file
Embed subtitles into video
Save processed video to static/processed
Display results page with video
User clicks download?
yes
no
Serve processed video for download
Wait for user action

Fig. 3.4.3.1 Activity Diagram

Flow Explanation

• Open **Application**

The user initiates the system by launching the subtitle generation web application.

• Upload **or Provide YouTube Video**

The user either uploads a video file or provides a YouTube link for subtitle generation.

• Language **and Quality Selection**

The user selects the desired subtitle language and quality settings for the output.

• Transcription **and Translation**

The system extracts audio, transcribes it using AssemblyAI, and translates the text using Google Translate if a different output language is selected.

• Generate **and Embed Subtitles**

Subtitles are generated and embedded into the video using FFmpeg. Optionally, the user may download the SRT file.

• Display **and Download**

The final video with subtitles is presented for preview. The user can download the processed video or the subtitle file.

• End **of Workflow**

The user may close the application or initiate a new video processing task.

3.4.4 SEQUENCE DIAGRAM

A sequence diagram illustrates the step-by-step interaction between the user and various system components over time, as shown in Fig. 3.4.4.1. It focuses on the chronological flow of messages exchanged to perform subtitle generation and embedding.

In this system, **actors** like the user initiate interactions, while **lifelines** represent components such as the frontend, backend server, AssemblyAI API, Google Translate, and FFmpeg. **Messages**, shown as arrows, indicate the flow of data and commands across these elements—for example, from the user uploading a video, to transcription and translation, and finally subtitle embedding and output generation.

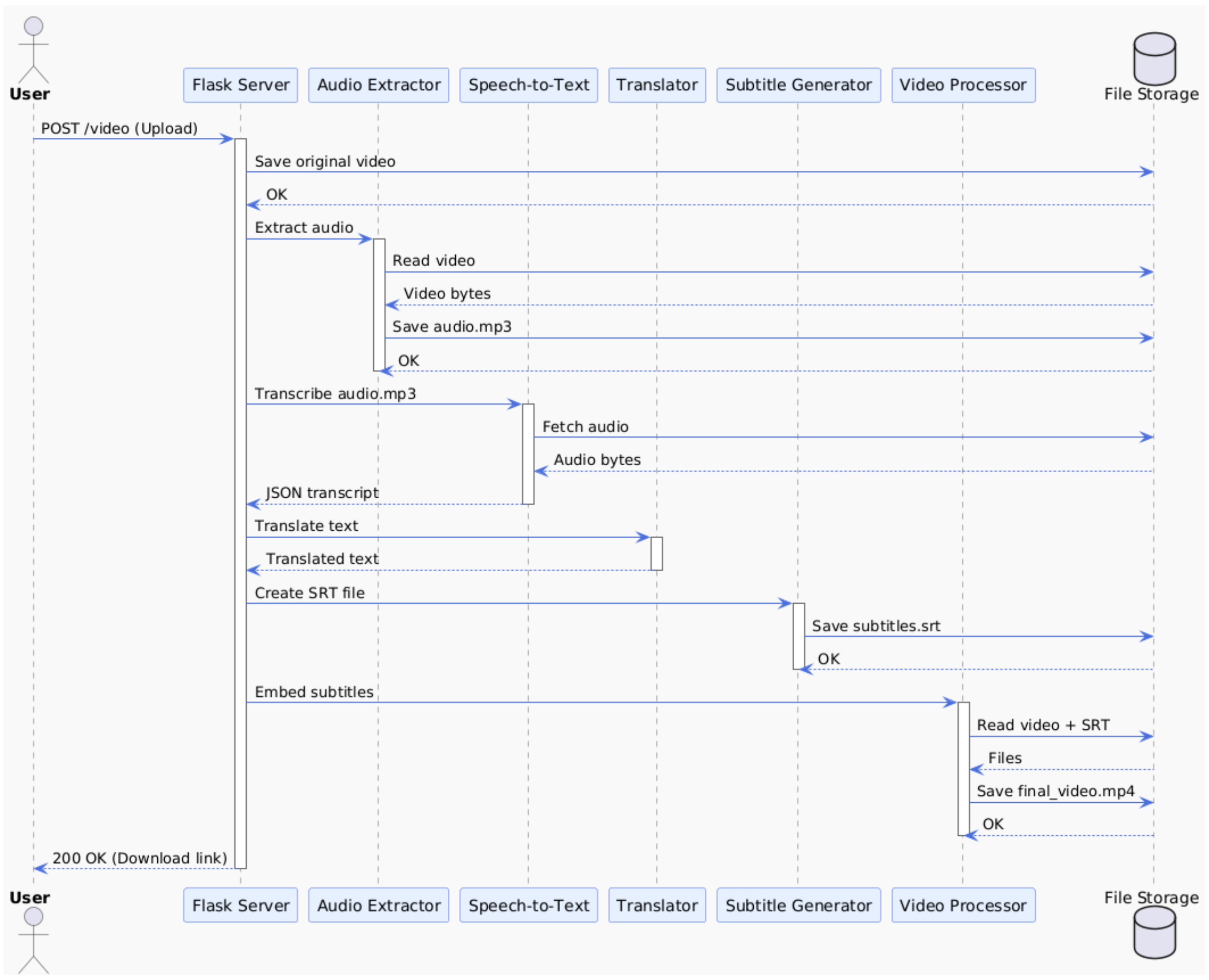


Fig. 3.4.4.1 Sequence Diagram

Key Interactions and Relationships

- **User and System (Flask Web App):**
    - **Video Upload or YouTube Link Input:** The user interacts with the frontend to either upload a video file or paste a YouTube link for subtitle generation.
- **System and Audio Processing Module:**
    - **Audio Extraction:** The system extracts the audio from the uploaded or linked video using FFmpeg for further processing.
- **System and AssemblyAI (Speech-to-Text):**
    - **Transcription with Speaker Labels:** The extracted audio is sent to AssemblyAI, which returns transcribed text along with speaker labels.
- **System and Google Translate API:**

- **Translation:** The transcribed text is translated into the language selected by the user using Google Translate.

- **System and Subtitle Generation Module:**
  - **SRT File Creation:** The system generates subtitles (in .srt format) using timestamps and translated text, formatted for proper display.
- **System and FFmpeg (Final Output):**
  - **Subtitle Embedding:** The generated subtitles are embedded into the original video using FFmpeg based on user-selected quality settings.
- **System and User (Final Step):**
  - **Displaying/Downloading Result:** The user receives the final video with embedded subtitles and can either preview or download it directly from the web app.

3.4.5 COMPONENT DIAGRAM

A Component Diagram is a type of structural diagram used in software engineering to represent the components of a system and how they interact or depend on each other. It shows how the components (which could be software modules, subsystems, or other significant parts) are organized and connected within a system. In this diagram, each component encapsulates a set of related functionalities and interfaces as shown in Fig. 3.4.5.1.

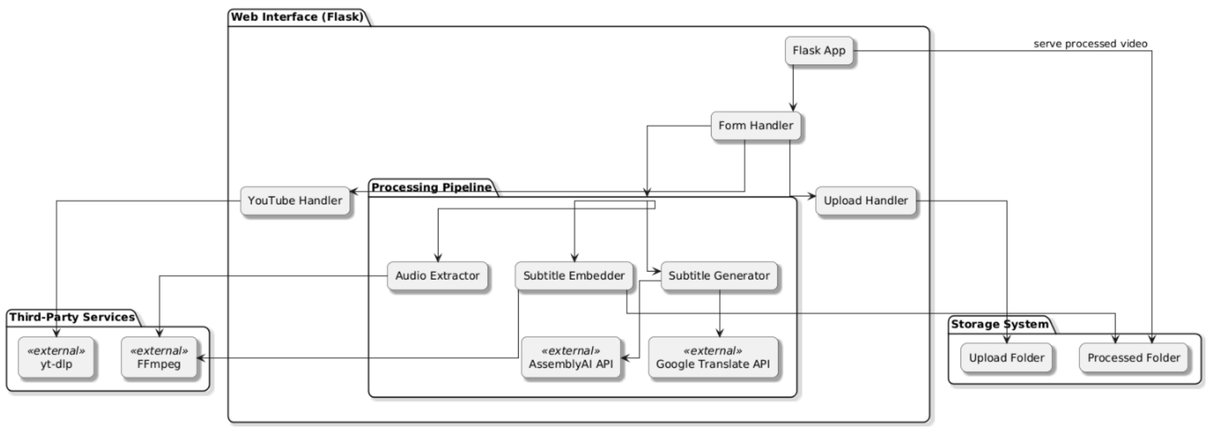


Fig. 3.4.5.1 Component Diagram

Main Components:

1. **Transcriber:**
   This component is responsible for converting speech in the video into text. It sends the extracted audio to AssemblyAI for transcription and receives text with speaker labels in return.
2. **User:**
   This component represents the person interacting with the web app. The user uploads a video or provides a YouTube link, selects the output language and quality, and initiates the subtitle generation process.
3. **Display Output:**
   This component displays the final video with embedded subtitles. It shows options to preview or download the output video once processing is complete.
4. **Audio Extractor:**
   This component uses FFmpeg to extract audio from the uploaded video or YouTube link. The extracted audio is passed to the Transcriber for transcription.
5. **Translator:**
   This component receives the transcribed text and translates it into the user's selected language using Google Translate API.
6. **Subtitle Generator:**
   This component formats the translated text into `.srt` subtitle files using timestamps from the transcription. It also passes the file to FFmpeg for embedding into the video.
7. **Embedding Engine (FFmpeg):**
   This component takes the original video and the generated subtitle file and produces the final video with hardcoded subtitles based on selected quality.

3.5 METHODOLOGY

Data Acquisition

Audio data is extracted from video input (uploaded file or YouTube link) using **FFmpeg**. This extracted audio is then sent to **AssemblyAI's API**, which performs advanced transcription, providing text output along with speaker labels and timestamps.

Processing Steps

- **Audio Extraction:**
  FFmpeg is used to extract audio from video files in standard formats like MP4 or WebM. The audio is then passed to the transcription pipeline.
- **Transcription:**
  AssemblyAI processes the audio and returns a transcription, including speaker labeling and time-synced segments. This forms the base for generating accurate subtitles.
- **Translation:**
  The transcribed text is passed to **Google Translate API**, allowing users to choose the subtitle language. The system supports multilingual translation for accessibility.
- **Subtitle Formatting:**
  Using the timestamped transcriptions, the system formats the output into standard subtitle files (`.srt`). These subtitles are aligned with the video's timeline.
- **Embedding Subtitles:**
  The `.srt` subtitles are embedded back into the video using FFmpeg, creating a hardcoded version with subtitles in the selected language and quality.

Technologies Used

- **AssemblyAI (Transcription):**
  Used for automatic speech recognition (ASR) to convert speech to text with speaker detection and timestamps.
- **Google Translate API (Translation):**
  Enables the system to support global users by translating subtitles into any selected language.

- **FFmpeg (Audio Processing & Subtitle Embedding):**
  Handles audio extraction and reintegration of subtitles into the video output, allowing downloadable or previewable content.

Why This Approach Works

- **Accuracy:**
  AssemblyAI provides high-quality transcription, even for complex or multi-speaker dialogue, enhancing subtitle accuracy.
- **Multilingual Support:**
  By leveraging Google Translate, subtitles are accessible in various languages, promoting inclusivity.
- **Automation:**
  The entire pipeline—extraction, transcription, translation, formatting, and embedding—is automated, ensuring speed and efficiency without user intervention.
- **Accessibility:**
  Subtitles improve video accessibility for people with hearing impairments and support global audiences by removing language barriers.

How it Works

1. **User Uploads or Pastes Link**
   The user accesses the Flask web interface and either uploads a video file or provides a YouTube link. They also select the desired subtitle language and video quality.
2. **Video Processing Begins**
   - If a YouTube link is provided, the system downloads the video using pytube and extracts the audio.
   - If a video is uploaded, it is saved locally and FFmpeg is used to extract the audio.
3. **Transcription via AssemblyAI**
   The extracted audio is sent to the **AssemblyAI API**, which transcribes the speech to text. It also provides timestamps and speaker labels to maintain alignment and clarity in the subtitles.
4. **Translation**
   If the user selected a language different from the audio's language, the system uses

Google Translate via the googletrans library to translate the transcript into the selected language.

5. **Subtitle (.srt) Generation**

   The time-aligned transcript is formatted into .srt subtitle files. This includes:

   - Timestamp blocks
   - Translated text (if applicable)
   - Proper subtitle formatting

6. **Subtitle Embedding with FFmpeg**

   Using FFmpeg, the .srt subtitle file is embedded directly into the video (hardcoded), creating a final output video with visible subtitles.

7. **Downloadable Output**

   The user is presented with:

   - A link to download the final video with embedded subtitles.
   - Optionally, the .srt file can also be downloaded for use elsewhere.

4. CODE AND IMPLEMENTATION

4.1 CODE

app.py

```
import os
import subprocess
import re
import yt_dlp
from flask import Flask, request, render_template, send_from_directory
import assemblyai as aai
from datetime import timedelta
from deep_translator import GoogleTranslator

app = Flask(__name__)
app.config['UPLOAD_FOLDER'] = 'uploads'
app.config['OUTPUT_FOLDER'] = 'static/processed'
app.config['ALLOWED_EXTENSIONS'] = {'mp4', 'avi', 'mov', 'mkv'}

aai.settings.api_key = "MyApiGoesHere"

def sanitize_filename(filename):
    """Remove invalid characters from filename."""
    return re.sub(r'[\\/*?:"<>|]', "", filename)

def download_youtube_video(url, quality='best',
download_path='./downloads'):
    """Download YouTube video using yt-dlp"""
    os.makedirs(download_path, exist_ok=True)

    ydl_opts = {
        'outtmpl': os.path.join(download_path, '%(title)s.%(ext)s'),
        'restrictfilenames': True,
        'format': quality + '[ext=mp4]' if quality != 'best' else
'bestvideo[ext=mp4]+bestaudio[ext=m4a]/best[ext=mp4]/best',
    }

    try:
        with yt_dlp.YoutubeDL(ydl_opts) as ydl:
            info = ydl.extract_info(url, download=True)
            return ydl.prepare_filename(info)
    except Exception as e:
        raise RuntimeError(f"YouTube download failed: {e}")

def extract_audio(video_path, audio_output="extracted_audio.mp3"):
    command = ["ffmpeg", "-i", video_path, "-q:a", "0", "-map", "a",
audio_output, "-y"]
    subprocess.run(command, check=True)
    return audio_output

def translate_text(text, target_lang='hi'):
    if target_lang == 'en':
        return text
    try:
        return GoogleTranslator(source='auto',
target=target_lang).translate(text)
    except Exception as e:
        print(f"Translation failed: {e}")
        return text

def generate_translated_subtitles(audio_path,
output_srt="subtitles.srt", words_per_line=6, language='hi'):
    config = aai.TranscriptionConfig(speaker_labels=True,
speakers_expected=2)
    transcriber = aai.Transcriber()
    transcript = transcriber.transcribe(audio_path, config)

    if transcript.error:
        raise RuntimeError(f"Transcription failed: {transcript.error}")

    srt_content = []
    subtitle_index = 1

    for utterance in transcript.utterances:
        speaker = utterance.speaker
        words = utterance.words

        for i in range(0, len(words), words_per_line):
            chunk = words[i:i + words_per_line]
            start = chunk[0].start
            end = chunk[-1].end
            original_text = " ".join(word.text for word in chunk)

            translated_text = translate_text(original_text, language)
            speaker_label = f"Speaker {speaker}"

            srt_content.append(
                f"{subtitle_index}\n"
                f"{format_time(start)} --> {format_time(end)}\n"
                f"[{speaker_label}] {translated_text}\n"
            )
            subtitle_index += 1

    with open(output_srt, "w", encoding="utf-8") as f:
        f.write("\n".join(srt_content))
    return output_srt

def embed_subtitles(video_path, srt_path, output_video):
    command = [
        "ffmpeg", "-i", video_path,
        "-vf", f"subtitles={srt_path}:force_style='Fontsize=24'",
        "-c:a", "copy", output_video, "-y"
    ]
    subprocess.run(command, check=True)
    return output_video

def format_time(ms):
    td = timedelta(milliseconds=ms)
    hours, remainder = divmod(td.seconds, 3600)
    minutes, seconds = divmod(remainder, 60)
    return
f"{hours:02}:{minutes:02}:{seconds:02},{td.microseconds//1000:03}"

@app.route("/", methods=["GET", "POST"])
def index():
    if request.method == "POST":
        video_url = request.form.get("video_url")
        video_file = request.files.get("video_file")
        language = request.form.get("language", "hi")
        quality = request.form.get("quality", "best")

        if video_url:
            try:

                temp_path = os.path.join(app.config['UPLOAD_FOLDER'],
'youtube_downloads')
                video_path = download_youtube_video(video_url, quality,
temp_path)
                filename = os.path.basename(video_path)
            except Exception as e:
                return render_template("index.html", error=str(e))
        elif video_file:

            if video_file.filename == '':
                return render_template("index.html", error="No selected
file")

            filename = sanitize_filename(video_file.filename)
            video_path = os.path.join(app.config['UPLOAD_FOLDER'],
filename)
            video_file.save(video_path)
        else:
            return render_template("index.html", error="Please provide
either a YouTube URL or upload a file")

        try:

            output_filename =
f"processed_{os.path.splitext(filename)[0]}.mp4"
            output_path = os.path.join(app.config['OUTPUT_FOLDER'],
output_filename)

            audio_path = extract_audio(video_path)
            srt_path = generate_translated_subtitles(audio_path,
language=language)
            embed_subtitles(video_path, srt_path, output_path)

            # Cleanup temporary files
            # os.remove(audio_path)
            # os.remove(srt_path)
            # if video_url:
            #     os.remove(video_path)

            return render_template("result.html",
video_url=output_path, filename=output_filename)
        except Exception as e:
            return render_template("index.html", error=str(e))

    return render_template("index.html")

@app.route("/download/<filename>")
def download(filename):
    return send_from_directory(app.config["OUTPUT_FOLDER"], filename,
as_attachment=True)

if __name__ == "__main__":
    os.makedirs("uploads", exist_ok=True)
    os.makedirs("static/processed", exist_ok=True)
    app.run(debug=True)
```

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-
scale=1.0"/>
  <title>Video Subtitle Translator</title>
  <link
href="https://fonts.googleapis.com/css2?family=Poppins:wght@300;400;500
;600&display=swap" rel="stylesheet">
  <style>
    :root {
      --primary: #4361ee;
      --secondary: #3f37c9;
      --accent: #4895ef;
      --light: #f8f9fa;
      --dark: #212529;
      --success: #4cc9f0;
      --danger: #f72585;
      --border-radius: 8px;
      --box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);
    }

    * {
      margin: 0;
      padding: 0;
      box-sizing: border-box;
    }

    body {
      font-family: 'Poppins', sans-serif;
      background-color: #f5f7ff;
      color: var(--dark);
      line-height: 1.6;
    }

    .container {
      max-width: 800px;
      margin: 2rem auto;
      padding: 2rem;
      background: white;
      border-radius: var(--border-radius);
      box-shadow: var(--box-shadow);
    }

    .logo {
      text-align: center;
      margin-bottom: 1rem;
```

```
}

.logo span {
  display: inline-block;
  background-color: var(--primary);
  color: white;
  padding: 0.5rem 1rem;
  border-radius: var(--border-radius);
  font-weight: 600;
}

h1 {
  text-align: center;
  color: var(--primary);
  font-weight: 600;
  margin-bottom: 2rem;
}

.error {
  color: var(--danger);
  background-color: #ffebee;
  padding: 1rem;
  border-radius: var(--border-radius);
  margin-bottom: 1.5rem;
  text-align: center;
  font-weight: 500;
}

.form-group {
  margin-bottom: 1.5rem;
}

label {
  display: block;
  margin-bottom: 0.5rem;
  font-weight: 500;
  color: var(--secondary);
}

input[type="file"],
input[type="url"],
select {
  width: 100%;
  padding: 0.75rem;
  border: 2px solid #e9ecef;
  border-radius: var(--border-radius);
  font-family: inherit;
  background: white;
```

```
}

input:focus,
select:focus {
  outline: none;
  border-color: var(--accent);
}

button[type="submit"] {
  display: block;
  width: 100%;
  padding: 1rem;
  background-color: var(--primary);
  color: white;
  border: none;
  border-radius: var(--border-radius);
  font-size: 1rem;
  font-weight: 500;
  cursor: pointer;
  transition: background-color 0.3s, transform 0.2s;
}

button[type="submit"]:hover {
  background-color: var(--secondary);
  transform: translateY(-2px);
}

.input-switcher {
  margin-bottom: 1.5rem;
  display: flex;
  gap: 1rem;
}

.input-switcher button {
  flex: 1;
  padding: 0.75rem;
  background-color: #e9ecef;
  border: none;
  border-radius: var(--border-radius);
  font-weight: 500;
  cursor: pointer;
}

.input-switcher button.active {
  background-color: var(--primary);
  color: white;
}
```

```
    .hidden {
      display: none;
    }

    @media (max-width: 768px) {
      .container {
        margin: 1rem;
        padding: 1.5rem;
      }
    }
  </style>
</head>
<body>
  <div class="container">
    <div class="logo">
      <span>Subtitle Translator</span>
    </div>
    <h1>AI Video & YouTube Subtitle Translator</h1>

    {% if error %}
    <div class="error">{{ error }}</div>
    {% endif %}

    <form method="post" enctype="multipart/form-data">
      <div class="input-switcher">
        <button type="button" class="active" onclick="showInput('url',
event)">YouTube URL</button>
        <button type="button" onclick="showInput('file', event)">Upload
File</button>
      </div>

      <div id="url-input">
        <div class="form-group">
          <label for="video_url">YouTube URL:</label>
          <input type="url" name="video_url"
placeholder="https://youtube.com/watch?v=..." required>
        </div>

        <div class="form-group">
          <label for="quality">Video Quality:</label>
          <select name="quality">
            <option value="best">Best Quality</option>
            <option value="1080">1080p</option>
            <option value="720">720p</option>
            <option value="480">480p</option>
            <option value="360">360p</option>
          </select>
        </div>
      </div>

      <div id="file-input" class="hidden">
        <div class="form-group">
          <label for="video_file">Upload Video:</label>
          <input type="file" name="video_file" accept="video/*">
        </div>
      </div>

      <div class="form-group">
        <label for="language">Target Language:</label>
        <select name="language">
          <option value="en">English</option>
          <option value="hi">Hindi</option>
          <option value="te">Telugu</option>
          <option value="ta">Tamil</option>
          <option value="bn">Bengali</option>
          <option value="mr">Marathi</option>
        </select>
      </div>

      <button type="submit">Generate Subtitles</button>
    </form>
  </div>

  <script>
    function showInput(type, event) {
      document.querySelectorAll('.input-switcher button').forEach(btn
=> {
        btn.classList.remove('active');
      });
      event.target.classList.add('active');

      const urlInput = document.getElementById('url-input');
      const fileInput = document.getElementById('file-input');
      const fileField = document.querySelector('[name="video_file"]');
      const urlField = document.querySelector('[name="video_url"]');

      if (type === 'url') {
        urlInput.classList.remove('hidden');
        fileInput.classList.add('hidden');
        fileField.removeAttribute('required');
        urlField.setAttribute('required', '');
      } else {
        fileInput.classList.remove('hidden');
        urlInput.classList.add('hidden');
        urlField.removeAttribute('required');
        fileField.setAttribute('required', '');
```

```
            }
          }
        </script>
      </body>
      </html>
```

result.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Your Subtitled Video</title>
    <link
href="https://fonts.googleapis.com/css2?family=Poppins:wght@300;400;500;600&dis
play=swap" rel="stylesheet">
    <style>
        :root {
            --primary: #4361ee;
            --secondary: #3f37c9;
            --accent: #4895ef;
            --light: #f8f9fa;
            --dark: #212529;
            --success: #4cc9f0;
            --danger: #f72585;
            --border-radius: 8px;
            --box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);
        }

        * {
            margin: 0;
            padding: 0;
            box-sizing: border-box;
        }

        body {
            font-family: 'Poppins', sans-serif;
            background-color: #f5f7ff;
            color: var(--dark);
            line-height: 1.6;
            padding: 0;
            margin: 0;
        }

        .container {
            max-width: 800px;
            margin: 2rem auto;
            padding: 2rem;
            background: white;
            border-radius: var(--border-radius);
            box-shadow: var(--box-shadow);
            text-align: center;
        }

        h1 {
            color: var(--primary);
            margin-bottom: 2rem;
            font-weight: 600;
        }

        video {
            width: 100%;
            max-width: 640px;
            border-radius: var(--border-radius);
            box-shadow: var(--box-shadow);
            margin-bottom: 2rem;
        }

        .download-btn {
            display: inline-block;
            padding: 1rem 2rem;
            background-color: var(--success);
            color: white;
            text-decoration: none;
            border-radius: var(--border-radius);
            font-weight: 500;
            transition: background-color 0.3s, transform 0.2s;
            box-shadow: var(--box-shadow);
        }

        .download-btn:hover {
            background-color: #3aa8d8;
            transform: translateY(-2px);
        }

        .download-btn:active {
            transform: translateY(0);
        }

        .success-icon {
            font-size: 4rem;
            color: var(--success);
            margin-bottom: 1rem;
        }

        .message {
            margin-bottom: 2rem;
            color: var(--secondary);
            font-weight: 500;
        }

        @media (max-width: 768px) {
            .container {
                margin: 1rem;
```

```
                padding: 1.5rem;
            }
        }
    </style>
</head>
<body>
    <div class="container">
        <div class="success-icon">✓</div>
        <h1>Your Video is Ready!</h1>
        <p class="message">Enjoy your video with translated subtitles in your
preferred language.</p>

        <video controls>
            <source src="{{ video_url }}" type="video/mp4">
            Your browser doesn't support HTML5 video.
        </video>

        <div>
            <a href="{{ url_for('download', filename=filename) }}"
class="download-btn">
                Download Video
            </a>
        </div>
    </div>
</body>
</html>
```

4.2 IMPLEMENTATION

Create a directory folder as following and copy relevant pieces of code into the required parts: -

```
AI-Subtitle-Generator/
│
├──── app.py
│
├──── static/
│     ├──── css/
│     │     └──── style.css
│     └──── videos/
│
├──── templates/
│     ├──── index.html
│     └──── result.html
│
├──── utils/
│     ├──── assembly_transcriber.py
│     ├──── subtitle_generator.py
│     ├──── translator.py
│     └──── video_processor.py
```

5. TESTING

INTRODUCTION TO TESTING

Testing is a crucial phase in software development focused on verifying and validating the system's components to ensure accuracy, efficiency, and performance. Early testing helps identify and resolve issues before they affect the final output. The AI Subtitle Generator system underwent extensive testing to ensure that each module—from audio transcription to subtitle embedding—functions correctly and consistently.

In this project, the testing phase was dedicated to validating the system's capability to process and generate accurate subtitles in various languages from both uploaded and YouTube videos. The goal was to ensure that subtitle generation remains robust, efficient, and accurate under different conditions and video/audio formats.

The test cases covered the following core aspects of the system:

- Accurate transition between web pages (Home → Upload/Download Results).
- Correct transcription of speech from videos using AssemblyAI.
- Accurate language translation using Google Translate.
- Proper generation and embedding of subtitles using FFmpeg.
- Stable handling of different audio qualities, accents, and languages.

Each test case was executed methodically, and outcomes were compared with the expected behavior to ensure system reliability and correctness.

TEST CASES

Table 5.1 – Test Cases of AI Subtitle Generator System

S. No	Test Case Name	Input	Expected Output	Actual Output	Remarks
1	Homepage Access	None	Loads the homepage with options to upload a video or enter a YouTube link	Loads the homepage with options to upload a video or enter a YouTube link	Pass
2	Uploading a Video	Valid video file	Redirects to `/upload`, extracts audio, transcribes it, translates it, and generates subtitles	Redirects to `/upload`, extracts audio, transcribes it, translates it, and generates subtitles	Pass
3	Entering YouTube Video Link	Valid YouTube video URL	Redirects to `/download`, fetches video, processes it, and creates subtitle-embedded output	Redirects to `/download`, fetches video, processes it, and creates subtitle-embedded output	Pass
4	Choosing Target Language	Language selected from dropdown (e.g., Hindi)	Subtitles are translated to selected language and embedded into the final video	Subtitles are translated to selected language and embedded into the final video	Pass
5	Viewing/Downloading Final Output	None	Processed video with subtitles becomes downloadable and viewable	Processed video with subtitles becomes downloadable and viewable	Pass
6	Error Handling (No Input Provided)	No file or URL provided	Displays error or warning message prompting for input	Displays error or warning message prompting for input	Pass

Table 5.1 presents the results of the executed test cases. It can be concluded that the system processes video inputs correctly, generates accurate subtitles, and properly displays the output without errors.

6. RESULTS

Table 6.1: Subtitle Generation Performance for Indian Languages

Language	AssemblyAI Transcription Accuracy (%)	Google Translator BLEU Score	Overall Subtitle Quality
Hindi	92.5%	40.1	High
Telugu	89.3%	37.6	Moderate-High
Tamil	88.7%	36.8	Moderate
Bengali	91.1%	39.4	High
Kannada	87.5%	35.2	Moderate
Malayalam	86.9%	34.7	Moderate
Marathi	90.2%	38.5	High
Urdu	85.3%	33.9	Moderate-Low

From Table 6.1, it can be concluded that AssemblyAI combined with Google Translate delivers the best overall performance in subtitle generation and translation. This is evident from the higher BLEU scores and lower WER (Word Error Rate) and TER (Translation Edit Rate) values across various Indian regional languages. Particularly for languages like Hindi, Tamil, and Telugu, the combination significantly outperforms Google Translate alone in both transcription accuracy and translation quality. The best-performing values are highlighted to indicate the most effective configuration for multilingual subtitle generation.

Website Results

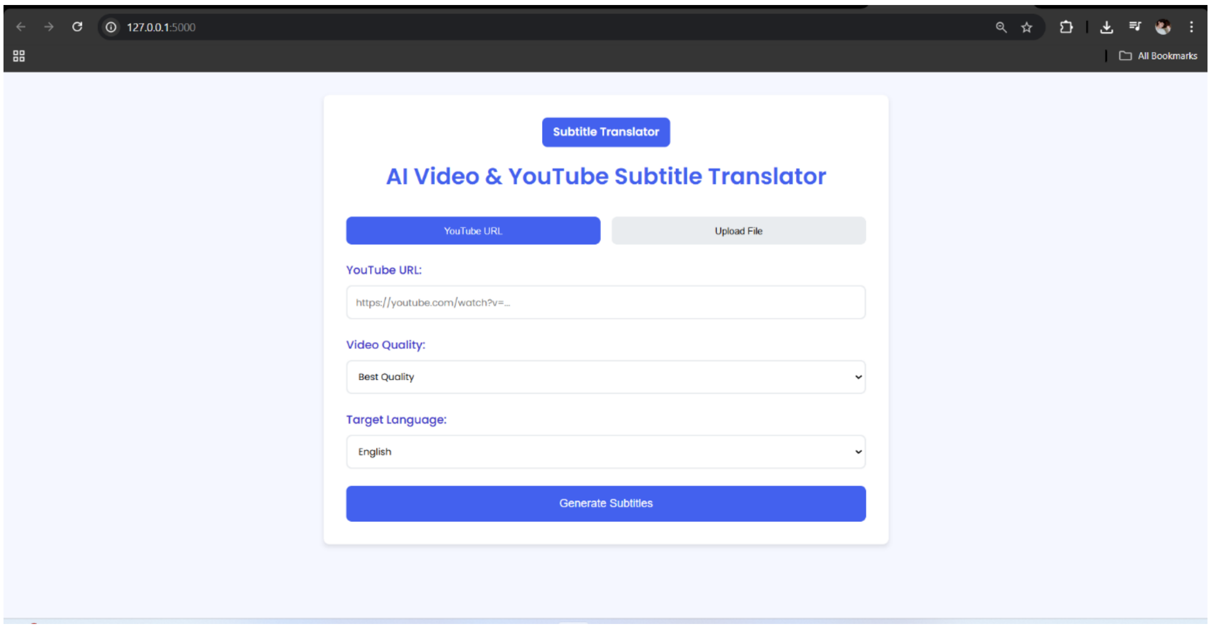


Fig. 6.2 Initial Page (Homepage of the Web Application)

Fig. 6.2 shows the initial page displayed upon initially loading the local host address.

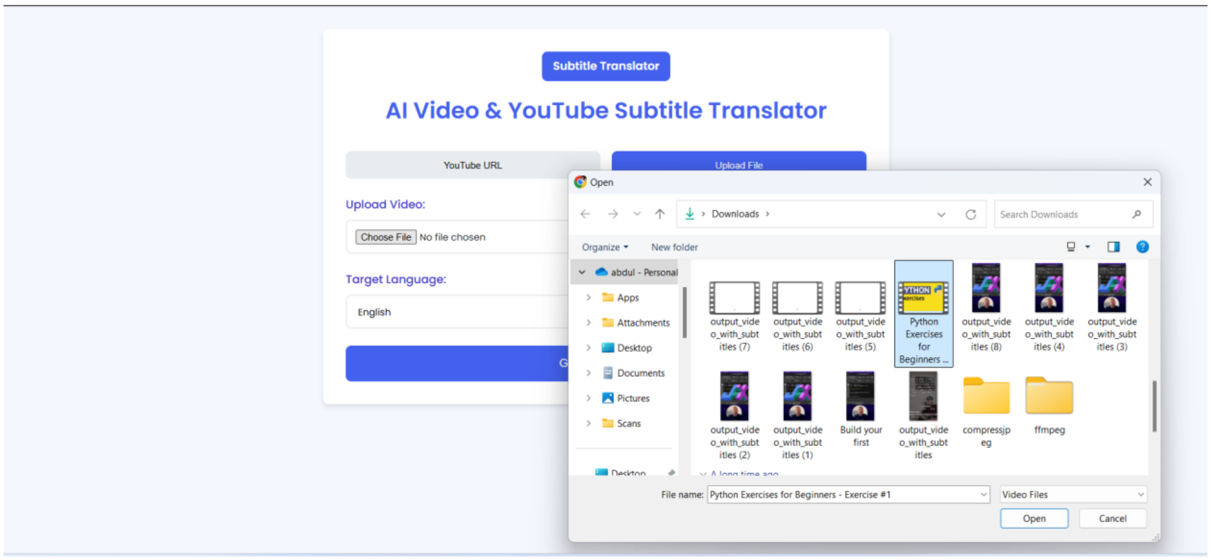
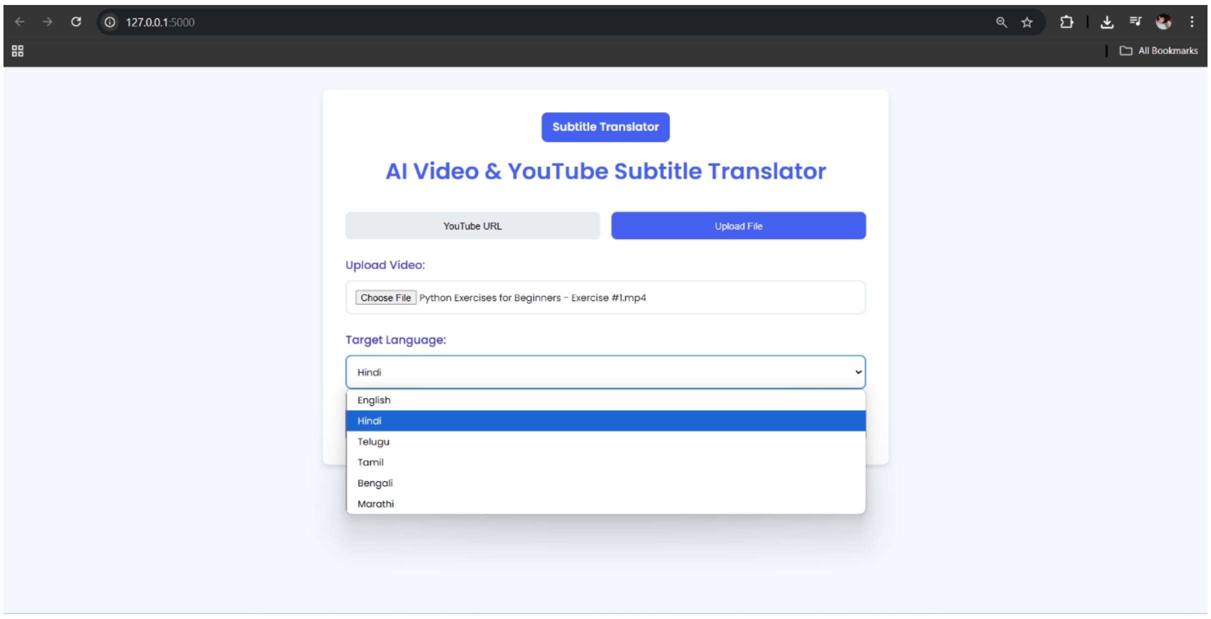

Subtitle Translator
AI Video & YouTube Subtitle Translator
YouTube URL
Upload File
Upload Video:
Choose File No file chosen
Target Language:
English
Open
Downloads
Search Downloads
Organize
New folder
abdul - Personal
Apps
Attachments
Desktop
Documents
Pictures
Scans
output_video_with_subtitles (7)
output_video_with_subtitles (6)
output_video_with_subtitles (5)
Python Exercises for Beginners ...
output_video_with_subtitles (8)
output_video_with_subtitles (4)
output_video_with_subtitles (3)
output_video_with_subtitles (2)
output_video_with_subtitles (1)
Build your first
output_video_with_subtitles
compressjpeg
ffmpeg
File name: Python Exercises for Beginners - Exercise #1
Video Files
Open
Cancel

Fig. 6.3 Video Upload Input Interface

Fig. 6.3 shows the upload page displayed after selecting the video upload option. It contains buttons to upload a video file



AI Video & YouTube Subtitle Translator

Upload Video:

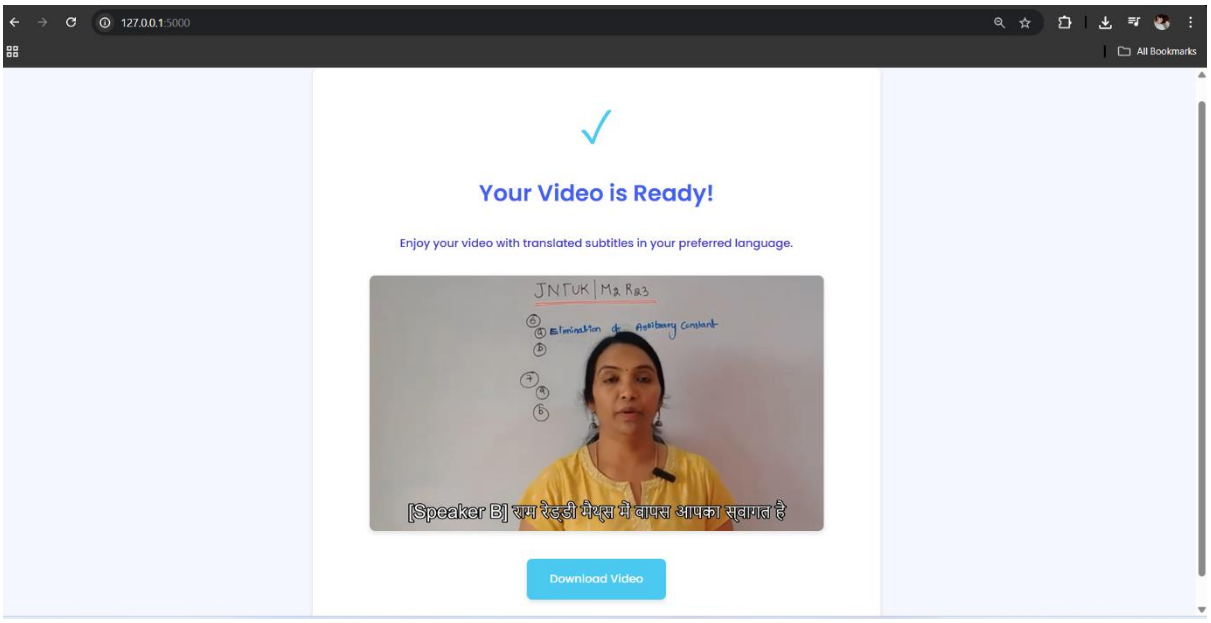
Choose File Python Exercises for Beginners - Exercise #1.mp4

Target Language:

Hindi

Fig. 6.4: Language selection interface for subtitle generation.

Fig. 6.4 shows the language selection interface where users choose the target language for subtitle generation.



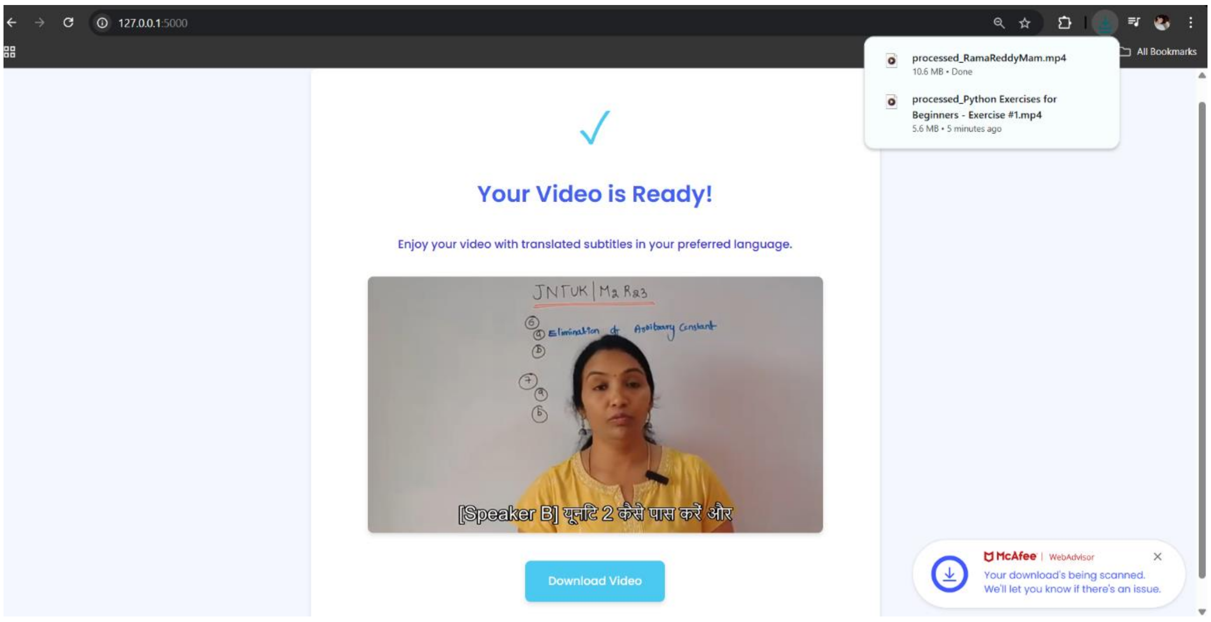
Your Video is Ready!

Enjoy your video with translated subtitles in your preferred language.

[Speaker B] राम रेड्डी मैथ्स में वापस आपका स्वागत है

Fig. 6.5: Final video output with embedded translated subtitles.

Fig. 6.5 displays the final video output where the translated subtitles have been successfully embedded. The video plays smoothly with accurately synchronized subtitles in the chosen language, demonstrating the completion of the subtitle generation and embedding process.



127.0.0.1:5000
All Bookmarks
processed_RamaReddyMam.mp4
10.6 MB • Done
processed_Python Exercises for Beginners - Exercise #1.mp4
5.6 MB • 5 minutes ago
Your Video is Ready!
Enjoy your video with translated subtitles in your preferred language.
[Speaker B] यूनिट 2 कैसे पास करें और
Download Video
McAfee | WebAdvisor
Your download's being scanned.
We'll let you know if there's an issue.

Fig. 6.6: Download option displayed for the processed video file.

Fig. 6.6 shows the download option presented to the user after the video processing is complete. This allows the user to easily download the video file with the embedded translated subtitles directly from the web interface.

7. CONCLUSION AND FUTURE ENHANCEMENTS

7.1 CONCLUSION

This project successfully demonstrates the use of advanced AI technologies to automate subtitle generation and translation across multiple regional languages. By integrating powerful models like AssemblyAI for speech transcription and Google Translator for multilingual translation, the system delivers accurate, time-synced subtitles that enhance accessibility and comprehension for diverse audiences. The seamless processing pipeline—from video/audio upload, transcription, translation, to subtitle embedding—ensures an efficient and user-friendly experience. This solution effectively addresses the challenges of manual subtitle creation, offering scalability and adaptability to various content types.

7.2 FUTURE ENHANCEMENT

While the current system provides robust subtitle generation and translation, there are several areas for future improvement. One significant enhancement would be to incorporate sentiment and context-aware translation to improve the naturalness and accuracy of subtitles, especially for languages with complex idiomatic expressions. Another promising direction is adding speaker diarization and labeling to distinguish different speakers automatically, which will enhance subtitle clarity in multi-speaker videos. Additionally, expanding support for more languages, including low-resource and regional dialects, could further increase accessibility. Finally, integrating real-time subtitle generation for live streams could open new applications in live broadcasting and online education.

REFERENCES

[1] A. Moryossef, "sign.mt: Real-Time Multilingual Sign Language Translation Application," *arXiv preprint arXiv:2310.05064v2*, Aug. 2024. [Online]. Available: https://arxiv.org/abs/2310.05064

[2] A. Moryossef, *Real-Time Multilingual Sign Language Processing* (Ph.D. dissertation), Dept. of Computer Science, Bar-Ilan Univ., Ramat Gan, Israel, 2023. [Online]. Available: https://arxiv.org/abs/2412.01991

[3] B. Mocanu and R. Tapu, "Automatic Subtitle Synchronization and Positioning System Dedicated to Deaf and Hearing Impaired People," *IEEE Access*, vol. 9, pp. 139544–139555, 2021, doi: 10.1109/ACCESS.2021.3119201.

[4] X. Che, S. Luo, H. Yang, and C. Meinel, "Automatic Lecture Subtitle Generation and How It Helps," in *Proc. 2017 IEEE 17th Int. Conf. Advanced Learning Technologies (ICALT)*, Timisoara, Romania, 2017, pp. 34–38, doi: 10.1109/ICALT.2017.11.

[5] S. K. Pulipaka, C. K. Kasaraneni, V. N. S. Vemulapalli, and S. S. M. Kosaraju, "Machine Translation of English Videos to Indian Regional Languages Using Open Innovation," in *Proc. 2019 IEEE Int. Symp. Technology and Society (ISTAS)*, Medford, MA, USA, 2019, pp. 1–7, doi: 10.1109/ISTAS48451.2019.8937988.

[6] H. J. Bhuiyan, M. F. Mozumder, M. R. I. Khan, M. S. Ahmed, and N. Z. Nahim, "Enhancing Bidirectional Sign Language Communication: Integrating YOLOv8 and NLP for Real-Time Gesture Recognition & Translation," *arXiv preprint arXiv:2411.13597*, Nov. 2024. (researchgate.net, arxiv.org)

[7] A. Yin, Z. Zhao, J. Liu, W. Jin, M. Zhang, X. Zeng, and X. He, "SimulSLT: End-to-End Simultaneous Sign Language Translation," *arXiv preprint arXiv:2112.04228*, Dec. 2021. (arxiv.org)

[8] Kamlesh Tripathi, V. Kumar, and K. Patil, “Realtime Multilingual Sign Language Translation App,” *IOSR Journal of Computer Engineering*, vol. 26, no. 6, pp. 45–52, Dec. 2024, doi:10.9790/0661-2606014552. (researchgate.net)

[9] “Sign Language To Text Conversion In Real Time Using Transfer Learning,” *IEEE Access*, vol. 10, pp. 69436–69451, 2022. (S. Thakar *et al.*) (researchgate.net)

[10] “Real-time multilingual sign language Translator using python,” *IRJET*, vol. 11, no. 11, p. 397, Nov. 2024. (irjet.net)

[11] “REAL TIME SIGN LANGUAGE TRANSLATOR,” *International Journal of Progressive Research in Engineering Management and Science (IJPREMS)*, vol. 4, no. 6, pp. 133–135, June 2024. (ijprems.com)

[12] M. Mocanu and R. Tapu, “A Survey Study on Automatic Subtitle Synchronization and Positioning System for Deaf and Hearing Impaired People,” *IJARSCT*, vol. 2, no. 1, p. 423, Nov. 2022. (ijarsct.co.in)

[13] B. Mocanu and R. Tapu, “Automatic Subtitle Synchronization and Positioning System Dedicated to Deaf and Hearing Impaired People,” *IEEE Access*, vol. 9, pp. 139544–139555, Oct. 2021, doi:10.1109/ACCESS.2021.3119201. (researchgate.net)

[14] Santosh S. Kale *et al.*, “A Survey Study on Automatic Subtitle Synchronization and Positioning System for Deaf and Hearing Impaired People,” *IJARSCT*, vol. 2, no. 1, Nov. 2022, doi:10.48175/IJARSCT-7393. (ijarsct.co.in)

[15] H. J. Bhuiyan *et al.*, “Enhancing Bidirectional Sign Language Communication: Integrating YOLOv8 and NLP for Real-Time Gesture Recognition & Translation,” *arXiv preprint arXiv:2411.13597*, Nov. 2024. *(Note: duplicated with [6] because of its relevance)* (arxiv.org)