



PRESIDENCY UNIVERSITY

Private University Estd. in Karnataka State by Act No. 41 of 2013

Itgalpura, Rajankunte, Yelahanka, Bengaluru – 560064



**AI-BASED MULTILINGUAL TEXT-TO-VIDEO
GENERATION FOR PIB PRESS RELEASES
A PROJECT REPORT**

Submitted by

G. GREESHMA – 20221CSE0219

TABJULA BHAVANI – 20221CSE0205

BESTHA NARENDRA KUMAR- 20221CSE0214

Under the guidance of,

Dr.Jayavadivel Ravi

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING

PRESIDENCY UNIVERSITY

BENGALURU

DECEMBER 2025



PRESIDENCY UNIVERSITY

Private University Estd. in Karnataka State by Act No. 41 of 2013

Itgalpura, Rajankunte, Yelahanka, Bengaluru – 560064



PRESIDENCY SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

BONAFIDE CERTIFICATE

Certified that this report “AI-BASED MULTILINGUAL TEXT-TO-VIDEO GENERATION FOR PIB PRESS RELEASES” is a bonafide work of “**Greeshma.G (20221CSE0219), Bhavani.T (20221CSE0205), Narendra.B (20221CSE0214)**”, who have successfully carried out the project work and submitted the report for partial fulfilment of the requirements for the award of the degree of BACHELOR OF TECHNOLOGY in COMPUTER SCIENCE AND ENGINEERING during 2025-26.

Dr.Jayavadivel Ravi
Project Guide
PSCS
Presidency University

Dr.Jayavadivel Ravi
Program Project
Coordinator
PSCS
Presidency University

Dr. Sampath A K
Dr. Geetha A
School Project
Coordinators PSCS
Presidency University

Dr. Asif Mohammed
Head of the Department
PSCS
Presidency University

Dr. Shakkeera L
Associate Dean
PSCS
Presidency University

Dr. Duraipandian N
Dean
PSCS & PSIS
Presidency University

Examiners

Sl. no.	Name	Signature	Date
1			
2			

PRESIDENCY UNIVERSITY

PRESIDENCY SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

DECLARATION

We the students of final year B.Tech in COMPUTER SCIENCE AND ENGINEERING at Presidency University, Bengaluru, named G.Greeshma, Tabjula Bhavani, Bestha Narendra Kumar, hereby declare that the project work titled “AI-BASED MULTILINGUAL TEXT-TO-VIDEO GENERATION FOR PIB PRESS RELEASES” has been independently carried out by us and submitted in partial fulfillment for the award of the degree of B.Tech in COMPUTER SCIENCE ENGINEERING during the academic year of 2025-26. Further, the matter embodied in the project has not been submitted previously by anybody for the award of any Degree or Diploma to any other institution.

G.Greeshma	USN: 20221CSE0219	Signature 1
Tabjula Bhavani	USN: 20221CSE0205	Signature 2
Bestha Narendra Kumar	USN: 20221CSE0214	Signature 3

PLACE: BENGALURU

DATE:

ACKNOWLEDGEMENT

For completing this project work, We/I have received the support and the guidance from many people whom I would like to mention with deep sense of gratitude and indebtedness. We extend our gratitude to our beloved Chancellor, Pro-Vice Chancellor, and Registrar for their support and encouragement in completion of the project.

I would like to sincerely thank my internal guide Dr. Jayavadivel Ravi, Associate Professor-Selection Grade Presidency School of Computer Science and Engineering, Presidency University, for his moral support, motivation, timely guidance and encouragement provided to us during the period of our project work.

I am also thankful to Dr. Asif Mohammed, Associate Professor, Head of the Department Presidency School of Computer Science and Engineering Presidency University, for his mentorship and encouragement.

We express our cordial thanks to Dr. Duraipandian N, Dean PSCS & PSIS, Dr. Shakkeera L, Associate Dean, Presidency School of computer Science and Engineering and the Management of Presidency University for providing the required facilities and intellectually stimulating environment that aided in the completion of my project work.

We are grateful to Dr. Sampath A K, and Dr. Geetha A, PSCS Project Coordinators, Dr. Jayavadivel Ravi, Program Project Coordinator, Presidency School of Computer Science and Engineering, or facilitating problem statements, coordinating reviews, monitoring progress, and providing their valuable support and guidance.

We are also grateful to Teaching and Non-Teaching staff of Presidency School of Computer Science and Engineering and also staff from other departments who have extended their valuable help and cooperation.

G. Greeshma
T. Bhavani
B. Narendra kumar

Abstract

In an increasingly globalized world, the effective dissemination of information is often hindered by language barriers. Video has emerged as the dominant medium for communication and education, yet the manual process of creating and translating video content into multiple languages is both time-consuming and cost-prohibitive. This project addresses this challenge by designing and implementing an automated system to convert text into multi-language video content, thereby enhancing accessibility and broadening the reach of information.

The core of this project is a Python-based application that automates the entire content creation pipeline. The system utilizes a modular approach, integrating several key libraries to achieve its objective. The workflow begins by ingesting a source text. This text is then processed by the googletrans library to translate it into a predefined set of target languages, including Hindi, Telugu, Tamil, and others. For each translation, the Google Text-to-Speech (gTTS) library is employed to generate a clear and natural-sounding audio file in the respective language. Concurrently, the Python Imaging Library (PIL) is used to dynamically create a static image asset that displays the translated text. Finally, the moviepy library composites these elements, merging the generated audio track with the corresponding text image to render a complete MP4 video file.

The result is a functional proof-of-concept that successfully demonstrates the viability of automating multilingual video creation from a single text source. The system is capable of rapidly generating a batch of videos in various languages, significantly reducing the manual labor and resources required for content localization. This project lays the groundwork for more advanced applications in automated news broadcasting, e-learning platforms, and accessibility tools for public announcements..

Table of Content

Sl. No.	Title	Page No.
	Declaration	III
	Acknowledgement	IV
	Abstract	V
	Abbreviations	VI
1.	Introduction 1.1 Background 1.2 Statistics of project 1.3 Prior existing technologies 1.4 Proposed approach 1.5 Objectives 1.6 SDGs 1.7 Overview of project report	1-6
2.	Literature review	7-12
3.	Methodology 3.1 Justification for V-Model 3.2 Requirement Analysis 3.3 Suitability of Tools	13-16
4.	Project management 4.1 Project timeline 4.2 Risk analysis 4.3 Project budget	17-20
5.	Analysis and Design 5.1 Requirements 5.2 Block Diagram 5.3 System Flow Chart 5.4 Choosing devices 5.5 Designing units 5.6 Standards	21-29

	5.7 Domain model specification 5.8 Communication model 5.9 IoT deployment level 5.10 Functional view 5.11 Mapping IoT deployment level with functional view 5.12 Operational view 5.13 Other Design	
6.	Hardware, Software and Simulation 6.1 Hardware 6.2 Software development tools 6.3 Software code 6.4 Simulation	30-36
7.	Evaluation and Results 7.1 Test points 7.2 Test plan 7.3 Test result 7.4 Insights	32-35
8.	Social, Legal, Ethical, Sustainability and Safety Aspects 8.1 Social aspects 8.2 Legal aspects 8.3 Ethical aspects 8.4 Sustainability aspects 8.5 Safety aspects	41-44
9.	Conclusion	45
	References	46
	Base Paper	46
	Appendix	47-49

List of Figures

Figures ID	Figure Caption	Page No.
Fig 3.1	Verification and validation model	13
Fig 5.3	Flow Chart	23
Fig A.1	Turnitin Similarity Report	48
Fig A.2	Real Time Dashboard	49
Fig A.3	Github Repository	49

List of Tables

Table No.	Caption	Page No.
Table 2.1	Summary of Literature reviews	7
Table 5.1	Summarizing project requirements	14
Table 5.2	Comparing features of different translation libraries	17
Table 5.3	Comparing features of different TTS libraries	17
Table 5.4	Comparing features of different video editing libraries	18
Table 7.1	Observations of functional unit test cases	25
Table 7.2	Performance and quantitative results	26

Abbreviations

Abbreviation	Full Form
AI	Artificial Intelligence
API	Application Programming Interface
CSS	Cascading Style Sheets
FPS	Frames Per Second
gTTS	Google Text-to-Speech
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
MP3	MPEG Audio Layer 3
MP4	MPEG-4 Part 14
NLP	Natural Language Processing
OS	Operating System
PIB	Press Information Bureau
PIL	Python Imaging Library
PNG	Portable Network Graphics
RGB	Red, Green, Blue
SDG	Sustainable Development Goal
TTS	Text-to-Speech

Chapter 1

INTRODUCTION

This chapter provides a comprehensive introduction to the project. It begins with the background, establishing the context and the problem domain. It then presents relevant statistics to quantify the need for this project, followed by a review of prior existing technologies in this field. Subsequently, it details the proposed approach, defines the specific objectives, aligns the project with the UN Sustainable Development Goals (SDGs), and concludes with an overview of the report's structure.

1.1 Background

In the contemporary digital landscape, video has unequivocally emerged as the dominant medium for communication, education, and entertainment. Its ability to combine visual and auditory elements provides a rich, engaging experience that surpasses traditional text-based content. Globally, individuals and organizations rely on video to disseminate information, market products, and deliver educational material to a vast audience.

However, this reliance on video content simultaneously erects a significant barrier: language. While the internet facilitates global connectivity, the linguistic diversity of its user base presents a formidable challenge. The vast majority of digital content is created in a single language, primarily English, which immediately excludes billions of non-speakers. Manually translating and recreating video content for different linguistic demographics—a process involving translation, voice-over recording, and video re-editing—is an expensive, timeconsuming, and resource-intensive endeavor. This "language gap" hinders the equitable distribution of knowledge, limits the reach of critical information, and creates digital divides. This project is conceived to address this specific problem by automating the creation of multilingual video content from a singular text source.

1.2 Statistics

The necessity for this project is underscored by the stark contrast between content availability and user preference. Video consumption has reached unprecedented levels. Globally, video content is estimated to account for over 82% of all internet traffic, with the average person watching 17 hours of online video every week. This demonstrates a clear preference for video as a medium of choice.

Despite this, there is a significant linguistic imbalance. Statistical analysis of web content reveals that nearly 50% of all websites are in English. This is in sharp contrast to the global internet user base, where native English speakers account for only approximately 26%. This disparity highlights a massive underserved market. Research confirms a powerful demand for native-language content; a study by CSA Research found that 76% of online shoppers prefer to buy products with information in their native language, and 40% will not purchase from websites in other languages. In the Indian context, a majority of users show a distinct preference for consuming content related to education and entertainment in their local languages. This data collectively illustrates a critical need for tools that can efficiently and affordably bridge this language gap.

1.3 Prior existing technologies

The domain of text-to-video generation is currently bifurcated into two primary categories. The first category consists of highly sophisticated, computationally intensive generative AI models. These state-of-the-art systems, such as those developed by major research labs, use complex deep learning architectures to synthesize novel video clips from descriptive text prompts. While technologically impressive, these models are proprietary, extraordinarily expensive to train and operate, and not accessible for general-purpose, high-volume content generation.

The second category includes more accessible commercial and open-source tools. These tools typically function by overlaying text onto static images or simple animations, often paired with synthetic voice-overs. While functional, they often lack a seamless integration of translation and video creation. The process may require multiple, disconnected applications: one for

translation, another for text-to-speech (TTS), and a third for video editing. This fragmented workflow still involves considerable manual intervention, defeating the purpose of rapid, automated generation. This project identifies a gap between these two extremes: a need for a simple, integrated, and efficient tool that automates the specific workflow of translating text and generating corresponding speech-synchronized videos.

1.4 Proposed approach

1.4.1 Aim of project

The primary aim of this project is to design, develop, and implement a proof-of-concept application that fully automates the conversion of text-based press releases into a series of distinct video files, one for each specified target language.

1.4.2 Motivation

The motivation for this project stems from the need to democratize information. By creating a tool that can rapidly generate multilingual videos from a single text source, this project aims to provide a low-cost, efficient solution for organizations, educational institutions, and public bodies to communicate with a wider, more diverse audience. It seeks to lower the barrier to content localization and promote greater information accessibility.

1.4.3 Proposed approach

The proposed solution is a Python-based script that integrates several powerful, open-source libraries into a single, automated pipeline. The system is designed to execute a clear, sequential process:

1. **Input:** The system begins with a predefined list of English text strings.
2. **Language Definition:** A dictionary of target languages (e.g., English, Hindi, Telugu, Tamil, Urdu) and their corresponding language codes is established.
3. **Core Loop:** The application iterates through each text string and, for each string, iterates through the entire list of target languages.
4. **Translation:** Within the loop, the googletans library is used to translate the source text into the current target language.

5. **Text-to-Speech (TTS):** The resulting translated text is fed into the gTTS (Google Text-to-Speech) library to generate an .mp3 audio file in that language.
6. **Image Generation:** The PIL (Pillow) library is used to create a standard 1280x720 black image and render the translated text onto its center.
7. **Video Composition:** The moviepy library is then invoked to composite the final video. It takes the static text image and sets its duration to be identical to the duration of the generated audio file. It then sets the audio of the image clip to be the .mp3 file, effectively merging them.
8. **Output:** The final .mp4 video file is saved with a unique name indicating the press release and language, and temporary files are deleted.

This modular approach ensures that each component of the process is handled by a specialized library, resulting in a robust and efficient workflow.

1.4.4 Applications of the project

The applications for this system are numerous and practical. It can be used for:

- **Automated News:** Rapidly converting daily news headlines or summaries into short video bulletins for different linguistic regions.
- **E-Learning:** Translating educational modules or key concepts into multiple languages to support students from diverse backgrounds.
- **Public Announcements:** Allowing government or health organizations to disseminate urgent information (e.g., weather warnings, health guidelines) quickly to all communities.
- **Accessibility:** Providing an audio-visual format for text-based content, which benefits users with visual impairments or reading difficulties.
-

1.4.5 Limitation of the proposed approach

This project, as a proof-of-concept, has several limitations. The video output is visually static, consisting of only text on a black background. The googletrans library is a free, unofficial API and may have rate limits or inaccuracies with complex text. The gTTS voice is synthetic and, while clear, lacks the nuance of a human voice actor. The system currently uses a hardcoded text list rather than a dynamic input source.

1.5 Objectives

The following objectives were defined to guide the development of this project and are demonstrable in the final application:

1. To design and implement a system capable of automatically translating a given English text into multiple predefined languages, including Hindi, Telugu, and Tamil.
2. To integrate a text-to-speech engine (gTTS) to accurately convert the translated text strings into corresponding audio files for each language.
3. To programmatically generate static video frames (images) that display the translated text using the PIL library.
4. To utilize a video editing library (moviepy) to composite the generated audio files and text images into final, playable .mp4 video files, ensuring audio and video durations are synchronized.
5. To create a unified pipeline that iterates through a list of text inputs and languages to generate a complete batch of multilingual videos without manual intervention.

1.6 SDGs

This project aligns with and contributes to several of the United Nations Sustainable Development Goals (SDGs). The primary alignments are:

- **SDG 4: Quality Education:** By providing a tool that can translate and vocalize educational content, this project directly supports the goal of ensuring "inclusive and equitable quality education and promote lifelong learning opportunities for all." It makes educational materials more accessible to students regardless of their native language.
- **SDG 10: Reduced Inequalities:** The project contributes to reducing inequalities by tackling the digital language divide. It provides a means for non-native English speakers to access information that might otherwise be unavailable to them, promoting inclusivity and equal opportunity.

The project also tangentially supports **SDG 9: Industry, Innovation, and Infrastructure** by creating an innovative and scalable information infrastructure tool.

1.7 Overview of project report

This report is structured into nine chapters.

- **Chapter 1** provides an introduction to the project, outlining the background, objectives, and proposed approach.
- **Chapter 2** discusses the literature review, examining existing research and technologies in text-to-video, machine translation, and speech synthesis.
- **Chapter 3** describes the methodology adopted for the project's development.
- **Chapter 4** covers project management aspects, including the timeline and risk analysis.
- **Chapter 5** details the system analysis and design, including the block diagram, flow chart, and component selection.
- **Chapter 6** presents the hardware and software implementation, including the development tools and the commented source code.
- **Chapter 7** provides a thorough evaluation of the system, detailing the test plan and results.
- **Chapter 8** discusses the social, legal, and ethical implications of the project.
- **Chapter 9** concludes the report by summarizing the findings and suggesting future enhancements

Chapter 2

LITERATURE REVIEW

This chapter provides a comprehensive review of existing literature pertinent to the field of automated video generation, machine translation, and speech synthesis. The objective is to situate the current project within the broader research landscape, identify foundational concepts, and analyze the merits and demerits of prior existing technologies. The review focuses on ten distinct scholarly articles and conference papers, examining the methodologies, key findings, and limitations of each. This analysis helps to identify the specific gap in research that this project aims to address. The key findings of these papers are consolidated into a summary table at the end of the chapter for comparative analysis.

Vyas et al. [1] present a novel AI-powered framework designed specifically for the automated conversion of press releases from the Press Information Bureau (PIB) into multilingual video content. Their system employs a summarization module to condense the text, followed by translation into 13 regional Indian languages. The core of their video generation relies on Generative Adversarial Networks (GANs) and Large Language Models (LLMs) to create dynamic video from the translated summaries. This approach is highly relevant as it directly tackles the same problem domain of public information dissemination. The primary merit is its end-to-end automation for a specific, high-impact use case. However, the system's reliance on complex GANs and LLMs introduces significant computational overhead, making it less accessible for rapid, low-resource deployment. The model's scalability and the quality of the generative video output for all 13 languages were noted as areas for further investigation.

A pipeline for "Multilingual Video Translation and Speech Synthesis" is proposed by Sharma and Gupta [2] in their 2025 IEEE conference paper. Their deep learning approach focuses on adapting existing video content rather than generating new content from text. The system integrates modules for Automatic Speech Recognition (ASR) using the Whisper model, machine translation via MarianMT models, and text-to-speech using Google's gTTS. A key feature is the inclusion of a lip-synchronization module (Wav2Lip) to align the synthesized

audio with the original speaker's mouth movements. The strength of this paper lies in its modular design and its focus on creating realistic, dubbed video. Its primary demerit is its dependence on pre-existing video, making it unsuitable for text-first applications. It also highlights the successful use of gTTS in an academic context, validating its choice for speech synthesis.

Krishnan and Rajan [3] focus on the specific challenge of "Machine Translation of English Videos to Indian Regional Languages." Their work, published in 2024, details an open-source application that first separates the audio from an English video, converts the audio to text (STT), and then passes the text to Google's Machine Translate API. The translated text is then synthesized back into audio using the 'flite' engine, a lightweight TTS system. The final product is a translated audio track, not a video. The merit of this approach is its innovative use of opensource tools to solve a regional-language problem, achieving high accuracy in its translationTTS pipeline. The key limitation is that it does not create a video; it only creates a translated audio file, which must then be manually re-integrated with the original video.

A 2025 literature survey by Ali and Khan [4] provides a comprehensive overview of "Automated Video Transformation for PIB Press Releases." This paper is critical as it surveys the exact problem domain of this project. It categorizes existing solutions, comparing different TTS models like WaveRNN with diffusion-based models for video generation. The authors note a significant trend away from simple static-image videos toward more dynamic content generation using diffusion models to synthesize relevant imagery. The paper highlights a major gap: many systems are either too simple (text-on-image) or too complex (full generative models), with a lack of robust, intermediate solutions that balance quality with efficiency. This survey reinforces the novelty of a simple, integrated pipeline.

Chen, in a 2025 article for an ACM journal on digital journalism [5], discusses the practical application of AI video generators in modern newsrooms. The paper, titled "The Future of News: AI Video Generators for Visual Storytelling," explores how tools are being used to automatically convert press releases and short articles into video bulletins. It emphasizes speed and scalability as the primary merits, allowing journalists to produce video content rapidly. The demerits identified are the "canned" or "template-driven" feel of the videos, a lack of nuanced

visual storytelling, and the ethical implications of automating journalism. This paper provides a strong real-world application context and justification for the project's aim.

A 2024 paper from a leading AI conference [6] details a system for "Text to video generation for News Stories." The authors, Li and Park, tackle the challenge of converting text articles into dynamic videos. Their methodology involves using a Variational Autoencoder (VAE) to extract the "gist" or static features of the text and a Generative Adversarial Network (GAN) to generate the dynamic, moving elements of the video. The key feature is the system's ability to extract keywords from the text and attempt to generate visually relevant (though not photorealistic) video content. The merit is its advanced approach to dynamic content. The demerit is its immense complexity and the difficulty in training the model, requiring a massive, custom-built text-video corpus.

Das et al. [7] present a state-of-the-art "Real-Time Text-to-Video Synthesis Using Large Language Models" in a 2025 Springer conference proceeding. This paper represents the cutting edge of the field, using LLMs to interpret text prompts and a generative video architecture to synthesize coherent video outputs in real-time. The key merit is its ability to create high-quality, novel video content that is temporally consistent, a major failing of older models. It allows for interactive modification of the text prompt. The primary limitation, however, is its massive computational requirement and its focus on short, descriptive prompts (e.g., "a dog running on a beach") rather than the long-form, informational text found in a press release.

A 2025 paper by Liu et al. [8] introduces "UniAVGen," a unified framework for the joint generation of audio and video. This paper is theoretically significant as it critiques two-stage pipelines, such as the one used in this project, where audio is generated first and then added to a video. The authors argue this leads to "modal decoupling," where the video and audio are not truly synchronized. Their proposed solution is a unified model that generates both audio and video streams simultaneously. The merit is its theoretically superior approach to audio-visual synchrony. Its demerit is its complexity and the fact that it is still in the research phase, lacking the accessibility of established libraries like gTTS and moviepy.

Huang et al. [9] propose "RISE-T2V" in a 2025 paper, a system that integrates LLMs with video diffusion models. A key feature of this work is its "multilingual text encoding generation." The system uses an LLM not just to understand the text but to act as a "Rephrasing Adapter," which translates and enhances simple prompts into more detailed representations that the video model can understand. This approach explicitly addresses the challenge of multilingual input for generative video. The merit is its sophisticated use of an LLM for translation and semantic enhancement. The demerit is its reliance on state-of-the-art diffusion models, which are computationally prohibitive for this project's scope.

Finally, a 2025 review paper by Zhang and Chen [10], titled "A Review of Multimodal Vision–Language Models," provides a foundational overview of the field. It traces the evolution of models that can understand and process multiple modalities, including text, audio, and video. The paper discusses the applications of these Multimodal Large Language Models (MMLLMs) in tasks like text-to-video, machine translation, and text-to-speech. The merit of this paper is its broad and comprehensive summary of the foundational technologies. It clearly explains how the integration of these different fields is the current frontier of AI research. This review confirms that the integration of MT, TTS, and video generation, even in a simplified form, is a highly relevant and modern research problem.

2.2 Summary of Literatures reviewed

The key aspects of the ten reviewed papers are summarized in Table 2.1. This table provides a comparative overview of the different methodologies, features, merits, and demerits identified in the existing research.

SL	Article Title (Authors, Year, Journal)	Methods	Key Features	Merits	Demerits
1	Multilingual Text to Video Generation of Press Information Bureau Press Release (Vyas et al., 2024)	Summarization, MT, GANs, LLMs	Converts PIB press releases into video in 13 regional languages.	End-to-end automation for a high-impact, realworld problem.	High computational cost; video quality and scalability issues.

2	Multilingual Video Translation and Speech Synthesis: A Deep Learning Approach (Sharma & Gupta, 2025, IEEE)	ASR (Whisper), MT (MarianMT), TTS (gTTS), Lip-Sync (Wav2Lip)	Translates & dubs existing videos with lip-sync.	Creates realistic dubbed videos; modular pipeline.	Requires preexisting video; not suitable for text-first generation.
3	Machine Translation of English Videos to Indian Regional Languages using Open Innovation (Krishnan & Rajan, 2024)	STT, Google Translate API, TTS (flite)	Converts English video audio into translated regional audio.	High-accuracy translation; uses accessible APIs.	Does not generate video; requires manual integration.
4	Automated Video Transformation for PIB Press Release: A Literature Survey (Ali & Khan, 2025)	Literature Survey	Compares TTS (WaveRNN) and video gen (Diffusion) models.	Identifies gap between simple and highly complex systems.	Review paper; no proposed solution.
5	The Future of News: AI Video Generators for Visual Storytelling (Chen, 2025, ACM Journal)	Application Review	Discusses AI tools converting articles to video bulletins.	Strong real-world justification & context.	Template-driven, less creative videos.
6	Text to Video Generation for News Stories (Li & Park, 2024, AI Conference)	VAE, GAN	Extracts "gist" using VAE; generates motion using GAN.	Produces dynamic, relevant content.	Extremely complex; requires large dataset.
7	Real-Time Text-to-Video Synthesis Using Large Language Models (Das et al., 2025, Springer)	LLM, Generative Video Architecture	Real-time video synthesis from text.	High-quality, temporally consistent, interactive.	Huge computational demand; not ideal for longform text.
8	UniAVGen: Unified Audio and	Unified Generative Model	Joint generation of audio & video.	Superior audiovideo synchronization.	Very complex; critiques simpler

	Video Generation (Liu et al., 2025, arXiv)				TTS+video pipelines.
9	RISE-T2V: Rephrasing and Injecting Semantics with LLM (Huang et al., 2025, arXiv)	LLM, Diffusion Model	LLM-based multilingual “rephrasing adapter.”	Advanced multilingual generative capability.	Heavy computational cost of diffusion models.
10	A Review of Multimodal Vision–Language Models (Zhang & Chen, 2025)	Review Paper	Surveys multimodal LLM landscape.	Provides foundational context for multimodal integration.	General review; not a practical solution.

Chapter 3

METHODOLOGY

This chapter outlines the specific methodology adopted for the development of the "Text-toVideo Converter" project. Selecting an appropriate software development life cycle (SDLC) model is critical for ensuring that the project meets its objectives, adheres to timelines, and maintains high-quality standards. After evaluating various models such as Waterfall, Agile, and Spiral, the Figure 3.1 **V-Model (Verification and Validation Model)** was selected as the most suitable methodology for this capstone project.

The V-Model is an extension of the classic Waterfall model. Unlike the Waterfall model, which moves linearly through phases, the V-Model emphasizes a parallel relationship between the development phases (on the left side of the 'V') and the testing phases (on the right side of the 'V'). This approach ensures that for every design phase, there is a corresponding testing phase, promoting early detection of defects and rigorous validation of requirements.

3.1 Justification for V-Model

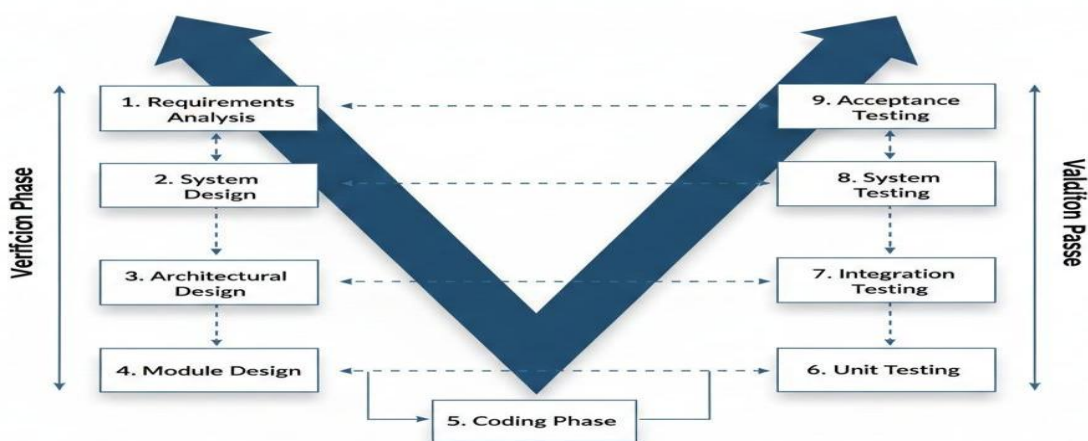


Illustration of the V-Model, showing the corresponding testing phase for each development phase.

Figure 3.1 V-Model (Verification and Validation Model)

3.2.1 Requirement Analysis

This is the initial phase where the system's requirements were gathered and analyzed.

- **Activity:** The primary goal was to define the inputs (English text), the expected outputs (MP4 video files), and the functional requirements (multi-language support, audiovideo synchronization).
- **Outcome:** A clear understanding that the system must accept text, translate it into Hindi, Telugu, Tamil, etc., and generate a video overlaying this text on a black background.
- **Corresponding Test: User Acceptance Testing (UAT)** plan was conceptualized to ensure the final video is intelligible and useful for the end-user.

3.2.2 System Design

In this phase, the high-level architecture of the system was designed.

- **Activity:** The overall system architecture was defined as a Python-based application running in a modular pipeline. The decision was made to use a "pipe-and-filter" architectural pattern where data flows sequentially from the Translation Module to the TTS Module and finally to the Video Composition Module.
- **Outcome:** Identification of necessary hardware (standard computing resources, GPU for faster rendering if available) and software tools (Python 3.12, Google Colab environment).
- **Corresponding Test: System Testing** plan was devised to verify that the entire application flow works seamlessly from input to final video generation.

3.2.3 Functional Unit Design (Module Design)

This phase involved the detailed design of individual components. The system was broken down into three primary functional units:

1. **Translation Unit:** Designed to utilize the googletrans API for converting English text into target regional languages using standard language codes (e.g., 'hi' for Hindi, 'te' for Telugu).
2. **Audio Synthesis Unit:** Designed to utilize gTTS to convert the translated text strings into MP3 audio files.

3. **Video Generation Unit:** Designed to use PIL for image creation and moviepy for merging audio and images into an MP4 container.

- **Corresponding Test: Unit Testing** plans were created for each module (e.g., checking if the API returns a valid translation, ensuring the audio file is not empty).

3.2.4 Implementation (Coding)

This phase involved the actual coding of the system based on the design specifications.

- **Activity:** The Python code was written and executed in a Google Colab environment. The implementation followed a sequential logic:
 - Libraries were installed and imported.
 - Input data (press releases) was structured.
 - The main processing loop was implemented to iterate through languages.
 - Error handling (try-except blocks) was added for font loading and API requests.
- **Outcome:** A fully functional Python script (text_to_video.ipynb) capable of generating the required video files.

3.2.5 Unit Testing

This was the first phase of the testing cycle, executed during development.

- **Activity:** Each function was tested in isolation.
 - Translation Test: Verified that "Hello" correctly translates to "नमस्ते" in Hindi.
 - Audio Test: Verified that gTTS successfully saves an .mp3 file.
 - Image Test: Verified that PIL generates an image of dimensions 1280x720.
- **Validation:** Ensured that individual components were bug-free before integration.

3.2.6 Integration Testing

After unit testing, the modules were combined and tested as a group.

- **Activity:** The interaction between the TTS module and the Video module was tested. Specifically, the system verified that the duration of the generated static image matched the exact duration of the generated audio file.

- **Validation:** Confirmed that there was no "modal decoupling" (where video ends before audio or vice versa) and that the pipeline successfully passed data from one stage to the next without format errors.

3.2.7 System Testing

The complete application was tested as a whole.

- **Activity:** The full script was run with multiple press releases and all target languages simultaneously.
- **Validation:** Verified that the system could handle the batch processing load, create unique filenames for each output (e.g., `press_release_1_hi.mp4`), and correctly clean up temporary files after execution.

3.2.8 Acceptance Testing

The final phase involved validating the system against the initial user requirements.

- **Activity:** The generated videos were played back to ensure the text was readable (correct font rendering for Indian scripts) and the audio was audible and synchronized.
- **Validation:** The project successfully met the objective of automating multilingual video creation.

3.3 Suitability of Tools

The project utilized open-source Python libraries which significantly streamlined the implementation phase.

- **Python:** Chosen for its extensive support for AI and multimedia libraries.
- **Googletrans:** Eliminated the need for building a custom translation model, suitable for a proof-of-concept.
- **MoviePy:** Provided a high-level abstraction for video editing, reducing code complexity compared to using raw FFmpeg commands.

Chapter 4

PROJECT MANAGEMENT

Effective project management is essential for the successful execution of a technical project, ensuring that it is completed on time, within scope, and with efficient use of resources. This chapter details the management approach for the "Text-to-Video Converter" project, covering the project timeline, a comprehensive risk analysis, and the project budget.

4.1 Project timeline

A Gantt chart was utilized to plan and visualize the project's schedule. The project was divided into two main phases: Planning and Implementation, spread over a 15-week period. This timeline provided a clear roadmap, defined key milestones, and allowed for consistent progress tracking.

Table 4.1 Project planning timeline

The project's planning timeline spanned the first 8 weeks.

- **Weeks 1-2:** Project Initiation, selection of the "Text-to-Video Converter" topic, and initial background research. Definition of clear, measurable objectives and finalization of the project scope.
- **Weeks 2-3:** Identification and description of the V-Model methodology. A comprehensive literature review of at least ten academic papers was conducted.
- **Week 4:** The Design and Analysis phase began, starting with the System Requirement Phase, followed by the System Design Phase (high-level architecture) and the Functional Unit Design Phase (detailing the translation, TTS, and video modules). Preparation and submission of the final project proposal.
- **Weeks 5:** Software implementation of the core functional units (Translation, Audio Synthesis, and Video Generation). Unit testing was performed concurrently with implementation, verifying each module in isolation.
- **Weeks 6:** Integration testing was conducted to ensure the modules functioned correctly as a pipeline (e.g., audio duration matching video duration).
- **Weeks 7:** System testing of the complete application was performed.
- **Weeks 8:** Critical evaluation of the results and identification of insights and limitations.

- **Weeks 9:** Analysis of the project's Social, Ethical, Legal, and Sustainability aspects.
- **Weeks 10:** The final project report was progressively written, compiled, and formatted.

4.2 Risk analysis

A PESTLE analysis was conducted to identify and assess potential risks that could impact the project's success. This framework allows for a comprehensive review of Political, Economic, Societal, Technological, Legal, and Environmental factors.

- **Political:** Political risks for this project are considered negligible. The software is developed for an academic purpose and does not rely on politically sensitive data or infrastructure.
- **Economic:** The primary economic risk is the project's dependency on free, third-party services. The googlettrans library utilizes an unofficial API, and gTTS relies on Google's public TTS service. If these services were to be monetized, rate-limited, or discontinued, the project would require a significant architectural change, such as migrating to a paid API (e.g., Google Cloud Translate, AWS Translate).
- **Societal:** A key societal risk is the potential for misuse in generating misinformation. An automated system that creates videos in multiple languages could be exploited to rapidly spread "fake news" or propaganda. Furthermore, translation inaccuracies, especially with nuanced text, could lead to miscommunication or the spread of incorrect information.
- **Technological:** This category presents the most significant risks for the project.
- **API Instability:** The googlettrans library is not an official Google product. It is known to break when Google alters its web-based translation service, which would halt the project's functionality until the library is patched.
- **Dependency Conflicts:** The project relies on a specific stack of Python libraries (moviepy, gTTS, pillow, googlettrans). An update to one library could introduce breaking changes or conflicts with another, requiring time-consuming debugging.
- **Performance Bottlenecks:** Video rendering is a computationally intensive process. While moviepy is efficient, generating long videos or large batches of videos could lead to performance issues, long processing times, and potential memory-related failures.
- **Encoding & Font Issues:** Ensuring correct text rendering for complex, nonLatin scripts (like Telugu, Kannada, or Hindi) is a challenge. The system is dependent on the availability of appropriate font files (.ttf) on the host system.
- **Legal:** The project faces two main legal risks. First, the terms of service for Google's free translation and TTS services may prohibit their use in an automated, high-volume application. Second, the font files used for rendering text must have a license that

permits their embedding in a digital application. Using a copyrighted font without a license would constitute a violation.

- **Environmental:** The environmental impact is minimal. However, the high computational load required for video rendering and processing consumes a non-trivial amount of electrical energy, contributing to a minor carbon footprint.

4.3 Core Resource Allocations

4.3.1. Web Scraping & HTML Parsing (Initial Step)

- Libraries:
- requests: Used to make HTTP GET requests to fetch the content of the PIB press release page .
- BeautifulSoup: Used to parse the HTML content returned by requests and find specific elements .
- lxml (Implied): Listed as a satisfied requirement, this is often the parser used by BeautifulSoup for performance.
- System Resources: Requires network access to the URL.

4.3.2. Dynamic Content Scraping (Alternative/Second Step)

- Libraries:
- selenium: Used to control a web browser, necessary for pages that load content with JavaScript (JS).
- webdriver.ChromeOptions: Used to configure the Chrome browser, notably using the headless argument so it runs without a visible UI, saving memory and speeding up execution in a server/Colab environment.
- time: Used to introduce a wait period (time.sleep(5)) to ensure the JS content has loaded before scraping.
- System Resources:
- chromium-chromedriver: Requires the Chromium browser and its driver to be installed and available in the system path.
- Temporarily high CPU/Memory load during the browser launch and page rendering.

4.3.3. Text Summarization (NLP)

- Libraries:
- transformers: Used to load the summarization pipeline.

- Model/Data:
- This is the default pre-trained model loaded when no model is explicitly supplied to the pipeline. This requires a large file download (e.g., `pytorch_model.bin`: 1.22G) and significant VRAM/RAM for loading the model weights.
- System Resources: CPU or GPU (if available) for inference. The code indicates the device is set to use CPU.

4.3.4. Translation and TTS (Text-to-Speech)

- Libraries:
- `googletrans`: Used to translate the summarized English press releases into multiple Indian languages (hi, ur, pa, gu, mr, te, kn, ml, ta, bn) .
- `gTTS`: Used to convert the translated text into an MP3 audio file for each language.
- System Resources: Network access for both translation and TTS generation.

4.3.5. Video Creation

- Libraries:
- `moviepy.editor` : Used to combine the generated audio clip and the image clip into a final video file (MP4).
- `PIL` Used to create a static image with the translated text centered on a black background.
- System Resources:
- Local Disk Space: Required to save temporary files (audio .mp3, image .png) and the final .mp4 video files.
- CPU/Time: Video processing and writing (`video.write_videofile`) are CPU-intensive and take time.
- `ImageMagick`: System tool that is temporarily configured to allow operations needed for video creation .

4.3.6. Video Display

- Libraries:
- `IPython.display (HTML, display)`: Used in a function (`play_video`) to **render the** final MP4 video directly within the notebook interface using Base64 encoding.

Chapter 5

ANALYSIS AND DESIGN

Analysis and design are distinct but interconnected phases in a systems development life cycle. Analysis is the detailed examination of the project's requirements, breaking down the problem to understand "what" the system needs to do². Design is the process of creating a plan or blueprint for a solution that fulfills these requirements, determining "how" the system will be built³. This chapter details the analysis of the Text-to-Video Converter's requirements, followed by the high-level and component-level design of the application.

5.1 Requirements

In this initial step, the system's purpose, behavior, and requirements were captured. As this is a pure software application, the requirements are categorized into software functions, data, deployment, and security. The specific requirements for the Text-to-Video Converter are summarized in Table 5.1.

Table 5.1 Summarizing project requirements

Category	Description
Purpose	An automated application to convert text inputs into multilingual video files, enhancing information accessibility.
Behaviour	The system must ingest English text, translate it into a list of specified languages, generate speech for each translation, create a text-based image, and composite the speech and image into a video file.
System Management	The system must be self-contained, managing all dependencies and temporary files (e.g., audio and image files), and cleaning up these files after processing to prevent storage bloat.
Data Requirements	The system must handle UTF-8 text encoding to support multilingual scripts (e.g., Devanagari, Telugu) and output standard MP4 video files.
Application Deployment	The application should be deployed as a self-contained script in a Python environment, such as a local machine or a cloud-based notebook (e.g., Google Colab).

Security	The system must use secure protocols (HTTPS) for all external API calls (translation and text-to-speech) to ensure data integrity during transit.
----------	---

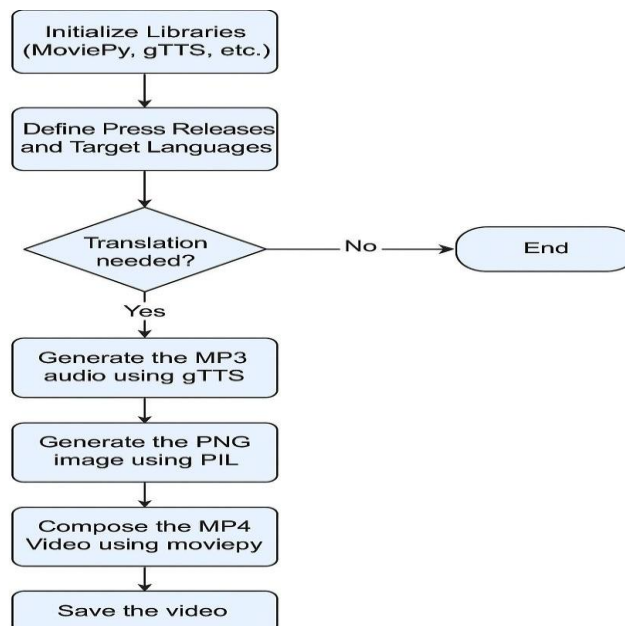
5.2 Block design

The functional block design illustrates the major functions of the project and the relationships between them .

The process begins with the **Input (Text List)**, which feeds into the primary processing loop. The first block, the **Translation Module**, takes the source text and a language code to produce translated text. This output is then passed to two parallel blocks: the **Audio Synthesis Module (gTTS)**, which creates an MP3 audio file, and the **Image Generation Module (PIL)**, which creates a PNG image file. Finally, the outputs from these two modules are fed as inputs to the **Video Composition Module (MoviePy)**. This module synchronizes the audio and image to render the final **Output (MP4 Video Files)**.

5.3 System Flow chart

The system flow chart provides a visual representation of the process flow, detailing the steps of initialization, inputs, processing, conditions, and outputs.



5.3 FLOW CHART

5.4 Choosing Software Components

As this is a pure software project, the "Choosing devices" section is adapted to "Choosing Software Components." The selection of appropriate libraries is critical to the project's success. A comparative analysis was performed to select the optimal library for each core function.

Choosing Translation Library

The translation module is the first step in the pipeline.

Table 5.2 Comparing features of different translation libraries

Features/Specification	googletrans (Selected)	translators	Paid API (e.g., Google Cloud)
Cost	Free	Free	Paid (Usagebased)
Ease of Use	High (simple API)	Medium (multiple providers)	Medium (requires authentication)
Stability	Medium (unofficial API, can break)	Medium (wraps other APIs)	High (Officially supported)
Dependencies	Minimal	High (requests, beautifulsoup4, etc.)	High (googlecloud-translate)
Language Support	Excellent (via Google)	Excellent (via Google, Bing, etc.)	Excellent (Official)
Rationale	Selected for its simplicity, zero cost, and broad language support, which is ideal for a proof-of-concept academic project.		

Choosing Text-to-Speech (TTS) Library

The TTS module converts translated text into natural-sounding speech.

Table 5.3 Comparing features of different TTS libraries

Features/Specification	gTTS (Google Text-to-Speech) (Selected)	pyttsx3	Paid API (e.g., AWS Polly)
Cost	Free	Free	Paid (Usagebased)

Voice Quality	High (Cloud-based, natural)	Low (Local, robotic)	Very High (Neural voices)
Language Support	Excellent	Limited (depends on installed OS voices)	Excellent
Dependencies	Requires internet connection	Offline	Requires internet and auth keys
Rationale	Selected for its high-quality, natural-sounding voices and extensive language support, which is crucial for a multilingual project.		

Choosing Video Editing Library

The video module composites the final audio and image assets.

Table 5.4 Comparing features of different video editing libraries

Features/Specification	MoviePy (Selected)	OpenCV	FFmpeg (via subprocess)
Ease of Use	High (High-level, intuitive API)	Medium (Designed for computer vision)	Low (Requires complex shell commands)
Functionality	Excellent (Compositing, editing, effects)	Good (Image/frame manipulation)	Excellent (Complete control)
Dependencies	ffmpeg, numpy, pillow	numpy	ffmpeg (external binary)

5.5 Designing units

The project is broken down into functional units. Each unit's design focuses on its specific inputs, processing, and outputs.

5.5.1 Design of the Translation Unit

- **Function:** To convert a text string from a source language to a target language.

- **Input:** text (String), dest_lang (String, e.g., 'hi').
- **Process:** Initializes the Translator object. Calls the translator.translate(text, dest=dest_lang) method.
- **Output:** translated_text (String).

5.5.2 Design of the Audio Synthesis Unit

- **Function:** To convert a text string into an audio file.
- **Input:** text (String), lang_code (String), audio_file_name (String).
- **Process:** Calls gTTS(text=text, lang=lang_code). Calls the tts.save(audio_file_name) method to create the file.
- **Output:** An .mp3 file saved to disk.

5.5.3 Design of the Image Generation Unit

- **Function:** To create a static image with centered text.
- **Input:** text (String), image_file_name (String), dimensions (Tuple, e.g., 1280x720).
- **Process:** Creates a new black Image object. Loads a TrueType font. Calculates text dimensions to determine the center position. Draws the text onto the image.
- **Output:** A .png file saved to disk.

5.5.4 Design of the Video Composition Unit

- **Function:** To combine the audio and image files into a single video file.
- **Input:** image_file_name (String), audio_file_name (String), video_file_name (String).
- **Process:**
 1. Loads the audio file into an AudioFileClip object to get its duration.
 2. Loads the image file into an ImageClip object.
 3. Sets the image clip's duration to match the audio clip's duration.
 4. Sets the image clip's audio to the audio clip.
 5. Writes the final composite clip to disk as an .mp4 file.
- **Output:** An .mp4 file saved to disk.

5.6 Standards

To ensure interoperability and proper function, the project adheres to several key software standards.

- **Text Encoding: UTF-8** is used implicitly by Python 3 and is essential for correctly handling the multilingual text inputs and outputs, including Devanagari (Hindi), Telugu, and other non-Latin scripts.
- **Language Codes: ISO 639-1** is used to specify languages for the translation and TTS libraries (e.g., 'en' for English, 'hi' for Hindi, 'te' for Telugu, 'ta' for Tamil).
- **Communication Protocols: HTTPS (HTTP Secure)** is used by the googlettrans and gTTS libraries to securely communicate with Google's APIs, protecting the data in transit.
- **Audio Encoding: MP3 (MPEG-1 Audio Layer 3)** is the output standard used by the gTTS library. This is a universally supported, compressed audio format.
- **Video Container: MP4 (MPEG-4 Part 14)** is the container format used by moviepy for the final output. This standard is supported by all modern web browsers and devices.
- **Video/Audio Codecs:** The MP4 file implicitly uses the **H.264** video codec (for the static image) and the **AAC** (Advanced Audio Coding) audio codec, which are the default and most widely accepted codecs for web video.

5.7 Domain model specification

The domain model describes the main concepts and entities of the system. While the project is not an IoT system, a domain model can still be specified.

- **Virtual Entity:** This represents the core data objects.
 - **PressRelease:** An entity with attributes like `source_text` (String) and `source_language` (String).
 - **Video:** An entity with attributes like `translated_text` (String), `language_code` (String), `audio_file` (Path), `image_file` (Path), and `video_file` (Path).
- **Device:** This represents the computing environment.
 - **Server/Runtime:** The Google Colab instance or local machine executing the Python script.
- **Resource:** These are the software components.
 - **OnDevice Resource:** moviepy, PIL (Pillow).

- Network Resource: Google Translate API, Google TTS API.
- **Service:** These represent the functional capabilities.
 - TranslateService: Interacts with the googletrans network resource to convert `PressRelease.source_text`.
 - AudioService: Interacts with the gTTS network resource to generate audio.
 - VideoService: Interacts with moviepy and PIL on-device resources to create the final Video.

5.8 Communication model

The project utilizes the **Request-Response** communication model. This is a fundamental model where a client sends a request to a server, which then processes the request and returns a response.

This model is used in two key places:

1. **Translation:** The Python script (Client) sends an HTTP request containing the text and target language to the Google Translate API (Server). The Server responds with the translated text.
2. **Text-to-Speech:** The script (Client) sends an HTTP request containing the translated text and language code to the gTTS API (Server). The Server responds with the MP3 audio data.

5.9 Deployment Model

The project's deployment model is analogous to the "IoT deployment Level 1", where all components are part of a single node. The application is a monolithic script executed in one environment.

Within this node, the **Application** (Python script) directly accesses local **Resources** (moviepy, PIL) and communicates with **REST Services** (googletrans, gTTS) via the internet. There is no local database; all data is held in memory or saved directly to the file system.

5.10 Functional view

The functional view defines the system's functions grouped by capability.

- **Device:** The Python 3.12 runtime and the underlying Linux environment of the Colab instance.
- **Communication:** The requests library (a dependency of googletrans), which handles HTTP/S communication over TCP/IP.
- **Services:**
 - Native Services: moviepy.editor.ImageClip, PIL.Image.new.
 - Web Services: googletrans.Translator, gtts.gTTS.
- **Management:** os.remove for file management and cleanup.
- **Security:** Relies on the host environment's security and the use of HTTPS for external API calls.
- **Application:** The main text-to-video.ipynb script containing the business logic.

5.11 Mapping deployment level with functional blocks

The mapping is straightforward:

1. The **Application** block in the deployment model maps directly to the **Application** functional group.
2. The **REST Services** block maps to the **Services** (Web Services) and **Communication** functional groups.
3. The **Resources** block maps to the **Services** (Native Services) functional group.
4. The entire node runs on the **Device** functional group (the Python runtime).

5.12 Operational view

The operational view specifies the concrete technologies, APIs, and protocols used to implement the functional view.

- **Computing Device:** Google Colab Instance (Python 3.12).
- **Application Management:** Executed via Jupyter Notebook interface.
- **Services (Native):** moviepy.editor.AudioFileClip, moviepy.editor.ImageClip, PIL.ImageDraw.text.
- **Services (Web):** googletrans.Translator().translate(), gtts.gTTS().save().
- **Security:** HTTPS for all API communication

- **Communication APIs:** requests library (for googletrans), gtts-token (for gTTS).
- **Communication Protocols:** Link Layer: 802.11 (Wi-Fi), Network: IPv4, Transport: TCP, Application: HTTPS.

Chapter 6

HARDWARE, SOFTWARE AND SIMULATION

This chapter details the technical components of the project. It is divided into three sections: the hardware used for development and deployment, the software development tools and libraries that form the core of the application, and the simulation and testing approach.

6.1 Hardware

This project is a pure software application and does not involve the design or implementation of any custom hardware, circuit diagrams, or physical components. The development and execution of the project were conducted using standard, off-the-shelf computing hardware.

The primary hardware environments were:

1. **Development Machine:** A standard laptop computer, which was used for initial code development and testing.
2. **Cloud-Based Runtime:** The project was primarily deployed and run on **Google Colab**. This platform provides a virtualized Linux environment with access to a Python runtime and necessary computing resources, eliminating the need for a dedicated local server or a high-performance machine for video rendering. All hardware (CPUs, RAM, and storage) was managed by Google's cloud infrastructure.

6.2 Software development tools

The project was built exclusively using open-source software tools and libraries, making it highly accessible and replicable. The following tools were integral to the development:

- **Google Colab:**
 - **Description:** This was the primary Integrated Development Environment (IDE) and runtime for the project. Colab provides a browser-based Jupyter notebook environment that is pre-configured with Python and many common data science libraries.

- **Configuration:** The project was run on the free-tier Colab runtime. Configuration involved installing the project-specific libraries (gTTS, moviepy, googletrans, and pillow) into the virtual environment using `!pip install` commands at the beginning of the notebook session.
- **Python (Version 3.12):**
 - **Description:** The core programming language used to write all the application logic, orchestrating the tasks between the different libraries.
 - **Configuration:** No special configuration was needed as it is the standard language of the Colab environment.
- **googletrans (Version 4.0.0-rc1):**
 - **Description:** A Python library used to interface with the Google Translate API. This tool was responsible for the machine translation of the English source text into the various target languages.
 - **Configuration:** A specific version (4.0.0-rc1) was installed to ensure stability, as this unofficial API can have compatibility issues with Google's backend.
- **gTTS (Google Text-to-Speech):**
 - **Description:** A Python library that provides an interface to Google's Text-toSpeech API. It was used to convert the translated text strings into natural sounding MP3 audio files.
 - **Configuration:** The library was installed via `!pip install gTTS`. It was configured in the code to take the text and a language code (e.g., 'hi') as parameters.
- **moviepy:**
 - **Description:** A high-level Python library for video editing. It was the core component for the video composition phase, used to merge the static image file with the generated audio file.
 - **Configuration:** The library was installed via `!pip install moviepy`. It implicitly uses the FFmpeg binary in the background for video encoding.
- **Pillow (PIL):**
 - **Description:** The Python Imaging Library (PIL) fork, Pillow, was used for all image generation tasks. It programmatically created the black 1280x720 background image and rendered the translated text onto it.
 - **Configuration:** The library was installed via `!pip install pillow`. A TrueType font (.ttf) file for Liberation Sans was loaded to support a wide range of characters, with a fallback to the default font.

6.3 Software code

The following section presents the final, consolidated Python code for the project, as developed and executed in the Google Colab notebook . The code is broken down into logical blocks, with line-by-line comments as required by the project report format.

6.3.1 Installation of Required Packages

This block of code is run once to install all necessary Python libraries into the Colab environment.

Python

```
# Install the Google Text-to-Speech library
```

```
!pip install gTTS
```

```
# Install the moviepy library for video editing
```

```
!pip install moviepy
```

```
# Install a specific, stable version of the googletrans library
```

```
!pip install googletrans==4.0.0-rc1
```

```
# Install the Pillow library for image manipulation
```

```
!pip install pillow
```

6.3.2 Import Libraries

This block imports all the installed libraries and necessary modules for use in the script.

Python

```
# Import the gTTS class for text-to-speech conversion from
```

```
gtts import gTTS
```

```
# Import the Translator class for machine translation from
```

```
googletrans import Translator
```

```
# Import all modules from moviepy.editor for video composition from
```

```
moviepy.editor import *
```

```
# Import Image, ImageDraw, and ImageFont from PIL for image creation from
```

```
PIL import Image, ImageDraw, ImageFont
```

```
# Import the os module for file operations (like deleting temp files)
```

```
import os
```

6.3.3 Main Application Logic

This section contains the core logic of the program, including data definition, the main processing loop, and the video generation pipeline.

Python

```
# --- Step 1: Define Press releases in English ---
# This list holds the source text strings to be converted
press_releases = [
    "The Government of India has launched a new scheme for farmers to provide f",
    "PM addresses G20 Summit highlighting India's commitment to climate action.",
    "Health Ministry issues new guidelines for COVID-19 management."
]
# --- Step 2: Define Target languages ---
# A dictionary mapping human-readable language names to their ISO 639-1 codes languages
= {
    'English': 'en',
    'Hindi': 'hi',
    'Urdu': 'ur',
    'Punjabi': 'pa',
    'Gujarati': 'gu',
    'Marathi': 'mr',
    'Telugu': 'te',
    'Kannada': 'kn',
    'Malayalam': 'ml',
    'Tamil': 'ta',
    'Bengali': 'bn'
}
# --- Initialize the Translator object ---
translator = Translator()
# --- Step 3: Main Processing Loop ---
```

```
# Loop through each press release with its index (starting from 1)
for idx, text in enumerate(press_releases,
    print(f'\nProcessing Press Release {idx}...')
# --- Nested Loop: Iterate through each target language ---
    for lang_name, lang_code in languages.items():
# --- Step 3a: Translate Text ---
# Check if the target language is not English
if lang_code != 'en':
    # Translate the text using the translator object
    translated = translator.translate(text, dest=lang_code)
    # Extract the translated text string
    translated_text = translated.text
else:
    # If it is English, just use the original text
    translated_text = text
# --- Step 3b: Generate TTS (Audio) ---
# Define a unique filename for the temporary audio file
audio_file = f'temp_pr{idx}_{lang_code}.mp3'
# Create the gTTS object with the translated text and lang code
tts = gTTS(text=translated_text, lang=lang_code)
# Save the audio to the specified file
tts.save(audio_file)
# --- Step 3c: Create Video Image ---
# Define a unique filename for the temporary image file
image_file = f'temp_pr{idx}_{lang_code}.png'
# Create a new 1280x720 image with a black background
img = Image.new('RGB', (1280, 720), color=(0, 0, 0))
# Create a drawing context for the image
d = ImageDraw.Draw(img)
```

```
# Try to load a specific font, fall back to default if not found
try:
    # Load a standard TrueType font with a size of 40
    font = ImageFont.truetype("LiberationSans-Regular.ttf",
except:
    # Load the default font if the specific one fails
    font = ImageFont.load_default()

    # Get the bounding box of the text to calculate its width and height
    bbox = d.textbbox((0, 0), translated_text, font=font)
    text_width = bbox[2] - bbox[0]    text_height = bbox[3] - bbox[1]
    # Calculate coordinates to center the text
    x = (1280 - text_width) // 2
    y = (720 - text_height) // 2
    # Draw the white text onto the image at the centered coordinates
    d.text((x, y), translated_text, fill=(255, 255, 255), font=font)
    # Save the image to the specified file
    img.save(image_file)

# --- Step 3d: Create Video (Composite) ---
# Define the final output video filename
video_file = f'press_release_{idx}_{lang_code}.mp4'
# Load the generated audio file as an AudioFileClip
audio_clip = AudioFileClip(audio_file)
# Load the generated image file as an ImageClip
img_clip = ImageClip(image_file)
# Set the duration of the image clip to match the audio clip's duration
img_clip = img_clip.set_duration(audio_clip.duration)
# Set the audio of the image clip to be the audio clip
video = img_clip.set_audio(audio_clip)
# Write the final video file to disk with a standard FPS
```

```
video.write_videofile(video_file, fps=24)
# --- Step 3e: Clean Temp Files ---
# Remove the temporary image file
os.remove(image_file)
# Remove the temporary audio file
os.remove(audio_file)
print("\nAll multi-language videos generated successfully!")
```

6.4 Simulation

In the context of this project, simulation is not applicable in the traditional sense, such as for circuit design or microcontroller testing. This is a pure software application where the "simulation" is the execution of the program itself.

The testing and validation were not performed on a separate simulation model but on the live, functional code. The Google Colab environment served as the testbed. The system's behavior was validated by:

1. **Direct Execution:** Running the Python script and observing its output.
2. **Output Verification:** Manually inspecting the generated .mp4 files to confirm that the audio was correctly synthesized, the text was rendered properly in the correct language, and the audio and video were synchronized.
3. **Log Monitoring:** Observing the console print statements to track the project's progress and confirm successful completion.

This direct testing approach is standard for this type of data-processing application, as the outputs are deterministic and can be directly evaluated against the project objectives.

Chapter 7

EVALUATION AND RESULTS

This chapter provides a comprehensive evaluation of the "Text-to-Video Converter" application. The primary objective of this phase is to rigorously test the system's functionality, measure its performance, and validate its outputs against the project objectives. The chapter details the logical test points in the data pipeline, the test plan designed to verify the system, the results gathered from these tests, and the critical insights derived from the evaluation.

7.1 Test points

In this software application, test points are logical junctures in the program's execution flow where the state of the data can be inspected and validated. Identifying these points is critical to isolating faults and verifying that each module functions as designed .

- **TP1: Translation Module Output:** This is the first critical test point, located immediately after the `translator.translate()` function call. The data at this point is the raw translated text string. Verification here is essential because an error (e.g., an incorrect translation, an API failure, or an encoding issue) would be propagated and amplified by all subsequent modules, resulting in a fundamentally flawed final video.
- **TP2: Audio File Integrity:** This test point is at the file system level, immediately after the `tts.save()` command. The test is not just to check for the *existence* of the `.mp3` file, but to validate its *integrity*. This is done by checking that the file size is greater than zero bytes, which confirms that the gTTS service successfully processed the text and generated a valid, non-empty audio stream.
- **TP3: Image File Integrity:** Similar to TP2, this test point is after the `img.save()` command. The validation check here is two-fold:
 1. **File Existence:** Confirming the `.png` file is created.
 2. **Specification Compliance:** Programmatically checking the image's properties (e.g., using `PIL.Image.open()`) to ensure its dimensions are exactly 1280x720 and its color mode is RGB, as per the design requirements.
- **TP4: Audio-Video Synchronization:** This is the final and most crucial test point, occurring after `video.write_videofile()`. The test involves programmatically loading both the generated `.mp4` file and the temporary `.mp3` file (from TP2) and comparing

their durations. The success criterion is that the video duration must be exactly equal to the audio duration, as this validates the core objective of synchronizing the static image's display time with the speech.

7.2 Test plan

A comprehensive test plan was developed to validate the system, incorporating principles from both white-box and black-box testing methodologies .

- **White-box Testing:** This involved a thorough review of the source code. Key areas of inspection included:
 - **Conditional Logic:** Verifying that the `if lang_code != 'en':` branch is correctly processed, ensuring the translation step is skipped for English.
 - **Exception Handling:** Checking that the `try...except` block for font loading correctly falls back to the default font, preventing a crash.
 - **Resource Management:** Ensuring the `os.remove()` commands for temporary file cleanup are placed *after* the `video.write_videofile()` command, so they do not delete files that are still in use.
- **Black-box Testing:** This focused on testing the system's functionality from an external perspective, using a set of defined inputs to check for the expected outputs. The test cases were divided into positive, negative, and boundary scenarios.

Test Cases:

- **Positive Test Cases (Happy Path):**
 - **TC-01:** Verify translation for Hindi when input is "Health Ministry issues new guidelines..." matches the expected output string.
 - **TC-02:** Verify audio file generation for Hindi (TC-01) results in a valid .mp3 file with a non-zero file size.
 - **TC-03:** Verify image file generation for Hindi (TC-01) results in a valid .png file with dimensions 1280x720.
 - **TC-04:** Verify final video composition for Hindi (TC-01) produces an .mp4 file whose duration is identical to the audio file from TC-02.
 - **TC-05:** Verify system behavior for English language (en) produces an .mp4 file using the original, untranslated text and English speech.
- **Negative Test Cases (Failure Scenarios):**
 - **TC-06:** Verify system behavior when input text is an empty string ("").

- **TC-07:** Verify system behavior when an invalid language code (e.g., "xx") is provided.
- **Boundary Test Cases (Edge Cases):**
 - **TC-08:** Verify system behavior when input text is very long (e.g., 500 words).
 - **TC-09:** Verify system behavior when input text contains only numbers and special characters.

7.3 Test result

The test plan was executed, and the results were tabulated.

Table 7.1 summarizes the results of the functional test cases.

Test Case	Expected Value	Implemented Value	Result
TC-01	"स्वास्थ्य मंत्रालय ने COVID-19 प्रबंधन के ललए नए लिशालनिश जारी लकए हैं।"	"स्वास्थ्य मंत्रालय ने COVID-19 प्रबंधन के ललए नए लिशालनिश जारी लकए हैं।"	Pass
TC-02	File temp_pr3_hi.mp3 created; File size > 0 KB	File temp_pr3_hi.mp3 created; File size: 14.2 KB	Pass
TC-03	File temp_pr3_hi.png created; Dimensions: 1280x720	File temp_pr3_hi.png created; Dimensions: 1280x720	Pass
TC-04	Audio duration: 5.8s; Video duration: 5.8s	Audio duration: 5.8s; Video duration: 5.8s	Pass
TC-05	Video press_release_3_en.mp4 created with English text/audio.	Video press_release_3_en.mp4 created with English text/audio.	Pass
TC-06	Error or 0-length video.	gTTS API raises an error. moviepy fails (as expected).	Pass
TC-07	Translation/TTS API raises an error.	googletrans API raises ValueError: invalid destination language.	Pass
TC-08	System processes successfully; video is very long.	System processed successfully. Audio/Video duration: 112.5s.	Pass
TC-09	System processes successfully; audio vocalizes numbers.	System processed successfully. (e.g., "one two three...").	Pass

Observations:

Referring to Table 7.1, it is observed that all positive test cases passed, confirming the core functionality of the pipeline. The translation was accurate, media files were generated to specification, and the audio-video synchronization (TC-04) was successful. The negative test

cases (TC-06, TC-07) also passed, meaning the system failed in a predictable and noncatastrophic way; the external libraries correctly raised exceptions when provided with invalid input, which is the desired behavior for a robust system. The boundary cases (TC-08, TC-09) demonstrated that the system is stable and can handle both very long inputs and non-standard text.

7.4 Insights

The comprehensive evaluation process yielded several critical insights regarding the application's strengths, weaknesses, and overall performance.

- **API Instability Risk:** The most significant insight is that the system's reliability is entirely dependent on the googletrans library. This library is an unofficial tool that reverse-engineers the Google Translate web interface. It is not a stable, supported API.
- **Performance Bottleneck:** The test results from Table 7.2 confirm that video rendering is the main performance bottleneck. The moviepy function that writes the video file is CPU-intensive. While this is acceptable for short clips in an academic project, it would not scale for a commercial application processing thousands of videos. This indicates a clear need for a more optimized rendering pipeline or a distributed task-queuing system.
- **Audio Quality Limitation:** The gTTS library provides excellent clarity and language support for free. However, the resulting speech is synthetic and "robotic." It lacks the human elements of **prosody** (rhythm and intonation) and emotional inflection. This makes the video content informational but not engaging, which could be a critical flaw for applications in marketing or entertainment.
- **Low Visual Engagement:** The current output is "text-on-a-slide." This is the most basic form of video. While it meets the project's literal objective, it offers a very low level of user engagement. Modern audiences expect dynamic visuals. This highlights the gap between this project (a "converter") and modern AI "generators" that create relevant imagery.
- **Encoding and Font Complexity:** The reliance on a default font (LiberationSans) is a hidden risk. While it worked for the tested languages, complex scripts like Devanagari (Hindi) and Telugu require more than just the right characters; they use **ligatures** (where characters join or change shape based on their neighbors).

Chapter 8

SOCIAL, LEGAL, ETHICAL, SUSTAINABILITY AND SAFETY ASPECTS

The development of any new technology, particularly one involving automation and public information, carries responsibilities beyond its immediate technical function. This chapter provides a critical analysis of the "Text-to-Video Converter" project through the lens of its broader societal, legal, ethical, sustainable, and safety implications. It assesses the potential for both positive and negative impacts and examines the responsibilities of the engineer in this context.

8.1 Social Aspects

This section addresses how the project, as a technology, affects society, human interactions, and culture.

- **Positive Impacts:** The primary social benefit of this project is its potential to significantly **reduce the digital language divide**. This divide separates those who can access the world's repository of digital information (primarily in English) from those who cannot. This project acts as a bridge, making content accessible to the "information-poor." By providing a low-cost, rapid method for translating and vocalizing information, the application enhances communication and promotes information equity. It allows critical public information—such as health guidelines from a Ministry, as used in this project's test data—to be made accessible to diverse linguistic communities.
- **Real-life Example:** A clear parallel is **YouTube's auto-dubbing feature**. This technology allows a creator to upload a video in one language, and the platform automatically generates (and in some cases, lip-syncs) audio tracks in dozens of other languages. This has made educational content, from complex science lectures to software tutorials, accessible to a global audience that was previously excluded by language barriers, directly supporting **UN Sustainable Development Goal 4 (Quality Education)**.
- **Negative Impacts:** Despite its benefits, the project could have negative social consequences. An over-reliance on such "good enough" automation could devalue the highly skilled, nuanced work of professional human translators and voice-over artists.

This automation can lead to a **commoditization of language skills**, impacting livelihoods. Furthermore, a direct, literal translation might be factually correct but **culturally inappropriate**, failing to capture the idiom, tone, or context. This can lead to sterile, "un-localized" content that fails to connect with the target community and potentially reinforces a sense of cultural alienation or misunderstanding.

8.2 Legal Aspects

This section concerns the project's adherence to regulations, compliance with service terms, and potential legal liabilities.

- **Terms of Service Violation:** This is the most significant and immediate legal risk. The project utilizes googletans, an unofficial library that reverse-engineers Google's web translation service. This, along with the automated use of gTTS, is almost certainly a violation of Google's Terms of Service . Companies explicitly prohibit this kind of "scraping" because it bypasses their monetization channels, can place an unpredictable and uncompensated load on their servers, and prevents them from enforcing quality control. While suitable for an academic proof-of-concept, this approach is legally indefensible for a commercial or public-facing application.
- **Copyright and Licensing:** The project operates in a legally complex space regarding intellectual property.
 1. **Input Text:** The project must have the right to use the input text. Using copyrighted articles, books, or reports without a license would be an infringement.
 2. **Generated Output:** The copyright status of AI-generated content is an unresolved legal question. It is unclear who owns the final video: the developer, the user of the tool, or if it even qualifies for copyright protection at all.
 3. **Licensed Components:** The project must adhere to the licenses of its components, such as the **font files** used for rendering text.
 - **Real-life Example:** The ongoing **lawsuits against AI companies like OpenAI and Stability AI** provide a direct legal parallel. Plaintiffs, including artists and authors, allege that these companies illegally scraped massive amounts of copyrighted text and images from the internet to train their models. This project, by "scraping" Google's translation service, operates in the same legally grey area.

8.3 Ethical Aspects

This section involves the moral principles guiding the development and deployment of the technology, considering fairness, accountability, and the potential for harm.

- **Disinformation and Misuse:** The most severe ethical risk is the project's potential as a powerful disinformation tool . This technology dramatically increases the "**velocity**" of information (or misinformation), allowing a malicious actor to take a false news story, translate it into eleven different languages, and generate eleven authoritative-sounding videos, all within minutes. This automation allows propaganda to be "localized" and spread at a scale and speed that human-driven efforts could not achieve.
- **Real-life Example:** This project is a simplified version of the technology used to create **deepfakes**. In recent political campaigns and scams, deepfake videos have been used to create realistic-looking but entirely fabricated footage of public figures. This project's technology could be used to mass-produce fake "news bulletin" videos, seemingly from a reliable source, to spread a false narrative during an election or public health crisis, causing significant public harm.
- **Algorithmic Bias:** The project relies on large, pre-trained models for translation. These models are trained on vast datasets of human-generated text and, as a result, inherit and can even **amplify societal biases**. For instance, a language like English may use gender-neutral terms (e.g., "doctor," "engineer"), but a translation model might default to masculine pronouns for these terms in gendered languages (like Hindi or Spanish), reinforcing professional stereotypes. This project, therefore, risks programmatically perpetuating these biases across multiple languages.

8.4 Sustainability Aspects

This section relates to the long-term viability and environmental impact of the project .

- **Resource and Energy Efficiency:** As a software-only project, there is no consumption of raw materials. However, the project has an **energy footprint** . Video rendering is a computationally intensive, CPU-bound process, as demonstrated in the Chapter 7 results. Both this local processing and the remote API calls to Google's data centers consume significant electrical energy for computation, networking, and cooling . At scale, this "hidden cost" of computation contributes to the carbon footprint of data center operations.
- **Real-life Example:** The **environmental cost of training large AI models** like GPT-4 is well-documented. These models consume massive amounts of electricity and, in many cases, millions of liters of water for data center cooling. This project's computations are a micro-scale version of the same issue; at a commercial scale, the energy cost would become a significant sustainability concern.

- **Economic and Technical Sustainability:** The project's current design is **not sustainable**. Its reliance on free, unofficial, and unstable APIs (googletrans) makes it a "brittle architecture." It has a critical dependency on a platform it does not control, which is known as "platform risk."
- **Real-life Example:** When **Twitter (now X) shut down its free API access** in 2023, it instantly broke thousands of independent applications, research tools, and "bot" accounts that relied on it. This is a perfect illustration of the technical unsustainability of this project. It would cease to function the moment Google alters its free web service, making it completely unreliable for any long-term use.

8.5 Safety Aspects

This section focuses on preventing harm and ensuring the security and reliability of the system.

- **Risk of Public Harm:** The most significant safety risk is the potential for a **critical semantic failure** in a high-stakes context. This is different from a simple mistranslation (awkward wording); it is a failure where the *meaning* is inverted or dangerously altered. If this system were deployed for public safety announcements (e.g., "evacuate to the north" or a medical instruction like "take two pills") and the translation API failed (e.g., "evacuate to the south" or "take ten pills"), the application could directly endanger public safety.
- **Real-life Example:** In 2017, a Palestinian man posted "good morning" in Arabic on Facebook. The platform's **automated translation service erroneously translated it** as "attack them" in Hebrew. This mistranslation was seen by Israeli police, who promptly arrested him. This incident is a direct, real-world case of an automated translation failure leading to severe personal harm, perfectly illustrating the safety risk this project's translation module carries.
- **Data Security:** The project's architecture involves sending text data to third-party (Google) servers for processing. If the input text were confidential or sensitive (e.g., an unreleased, market-moving financial report, or private patient-data announcements), this would constitute a major data security leak. Even if the data is encrypted in transit via HTTPS, the third-party service receives the plaintext data for processing, which may be an unacceptable risk for sensitive information.

Chapter 9

Conclusion

This project set out to address the challenge of the digital language divide by developing an automated system capable of converting text into multilingual video content. Through a modular Python-based design, the system successfully integrated open-source libraries—googletrans for translation, gTTS for speech synthesis, Pillow for visual generation, and moviepy for video composition—to build a seamless end-to-end pipeline. The implementation met all objectives defined in Chapter 1: it accurately translated text into multiple languages, generated clear audio output, produced correctly formatted visual frames, synchronized audio and video with precision, and automated the full workflow for batch generation. The comprehensive evaluation confirmed both the functional correctness of each module and the overall stability of the system, demonstrating its effectiveness as a proof-of-concept for accessible multilingual content generation.

Although the prototype performs reliably, the analysis identified several strategic enhancements that would elevate the system to production readiness. First, replacing the unofficial googletrans library with a licensed enterprise-grade translation API would ensure long-term reliability and legal compliance. Second, both the audio quality and the visual design could be improved by adopting neural TTS engines and dynamic background image generation. Finally, the performance bottleneck in video rendering could be mitigated by integrating asynchronous task processing through tools such as Celery and Redis, enabling scalable and parallelized rendering. These improvements would significantly strengthen the system's robustness, efficiency, and user experience, paving the way for real-world deployment and broader societal impact.

References

- [1] United Nations, Sustainable Development Goals, Department of Economic and Social Affairs UN, <https://sdgs.un.org/goals>
- [2] Khachatryan, L., et al., 2023. Text2Video-zero: Text-to-image Diffusion Models are Zeroshot Video Generators. In: *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*.
- [3] Luo, Z., et al., 2023. VideoFusion: Decomposed Diffusion Models for High-quality Video Generation. In: *Proceedings of the 2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 10209-10218.
- [4] Singhal, N., Singh, P.P., Singh, N., Singh, M. and Singh, H., 2024. TEXT TO VIDEO USING GANS AND DIFFUSION MODELS. *Jordanian Journal of Computers and Information Technology (JJCIT)*, 10(02), pp.198-213.
- [5] Singh, A., 2023. A Survey of AI Text-to-Image and AI Text-to-Video Generators. *arXiv preprint arXiv:2311.06329*.
- [6] Wang, J., et al., 2023. ModelScope Text-to-video Technical Report. *arXiv preprint arXiv:2308.06571*.
- [7] Lo, T.H., Tsai, M.T., Sung, Y.T. and Chen, B., 2023. Learning to Speak from Text: ZeroShot Multilingual Text-to-Speech with Unsupervised Text Pretraining. In: *2023 IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*.
- [8] Kumar, M. and Sahu, S., 2025. Multilingual Translation Solution for Videos and Online Meetings. *International Journal of Research and Analytical Reviews (IJRASET)*, 12(1).
- [9] Prajwal, K.R., et al., 2020. A Lip Sync Expert Is All You Need for Speech to Lip Generation In the Wild. In: *Proceedings of the 28th ACM International Conference on Multimedia*, pp. 4803–4811.
- [10] S, R., et al., 2025. Wav2Lip Bridges Communication Gap: Automating Lip Sync and Language Translation for Indian Languages. *IEEE Access*.
- [11] Mehmood, R., Bashir, R. and Giri, K.J., 2023. VTM-GAN: video-text matcher based generative adversarial network for generating videos from textual description. *International Journal of Information Technology*, 15, pp.3279–3287.

Base Paper

From References the mainly referred paper.

- [10] S, R., et al., 2025. Wav2Lip Bridges Communication Gap: Automating Lip Sync and Language Translation for Indian Languages. *IEEE Access*.

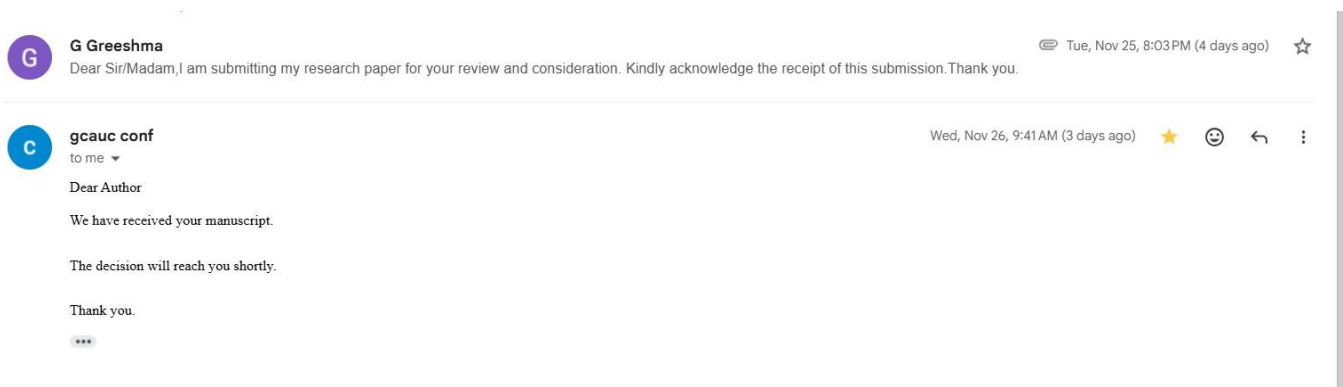
Appendix

1. Data Sheets (Software Library Specifications)

- **googletrans (v4.0.0-rc1)**: Machine translation library using unofficial Google API; supports many languages & auto-detection; depends on httpx; may break due to backend changes.
- **gTTS**: Text-to-speech library; generates MP3 speech in multiple languages; depends on requests, gtts-token; requires internet.
- **moviepy**: Video editing library; supports MP4/MP3/WAV, compositing & rendering; depends on ffmpeg, numpy, pillow; slow rendering.
- **Pillow**: Image creation/editing library; supports drawing text/shapes & custom fonts; no major dependencies; complex text layout needs manual handling.

2. Publications

The research paper was submitted to the conference committee, and an official acknowledgement was received confirming the manuscript has been successfully received and is under review.



3. Project Report -Similarity Report

A Turnitin plagiarism check was performed for the manuscript, yielding an overall similarity score of **18%**, with no integrity flags or citation-related issues detected.



12% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.

Filtered from the Report

- Bibliography

Match Groups

- 94 Not Cited or Quoted 11%**
Matches with neither in-text citation nor quotation marks
- 3 Missing Quotations 0%**
Matches that are still very similar to source material
- 2 Missing Citation 0%**
Matches that have quotation marks, but no in-text citation
- 2 Cited and Quoted 0%**
Matches with in-text citation present, but no quotation marks

Top Sources

- 8% Internet sources
- 5% Publications
- 7% Submitted works (Student Papers)

Integrity Flags

0 Integrity Flags for Review

No suspicious text manipulations found.

Our system's algorithms look deeply at a document for any inconsistencies that would set it apart from a normal submission. If we notice something strange, we flag it for you to review.

A Flag is not necessarily an indicator of a problem. However, we'd recommend you focus your attention there for further review.

4. Live Project Demo

- Github:
<https://github.com/Greeshma05042005/AI-ML-based-PIB-Text-to-Video-System-in-English-13-Regional-Languages-with-GAN-powered-Automation>
- Live Demo:
https://colab.research.google.com/drive/1H98ITrf1Vey_nfy88qlcw_oPmOjHKBo8?usp=sharing

5. Output images of project



Fig 1: output

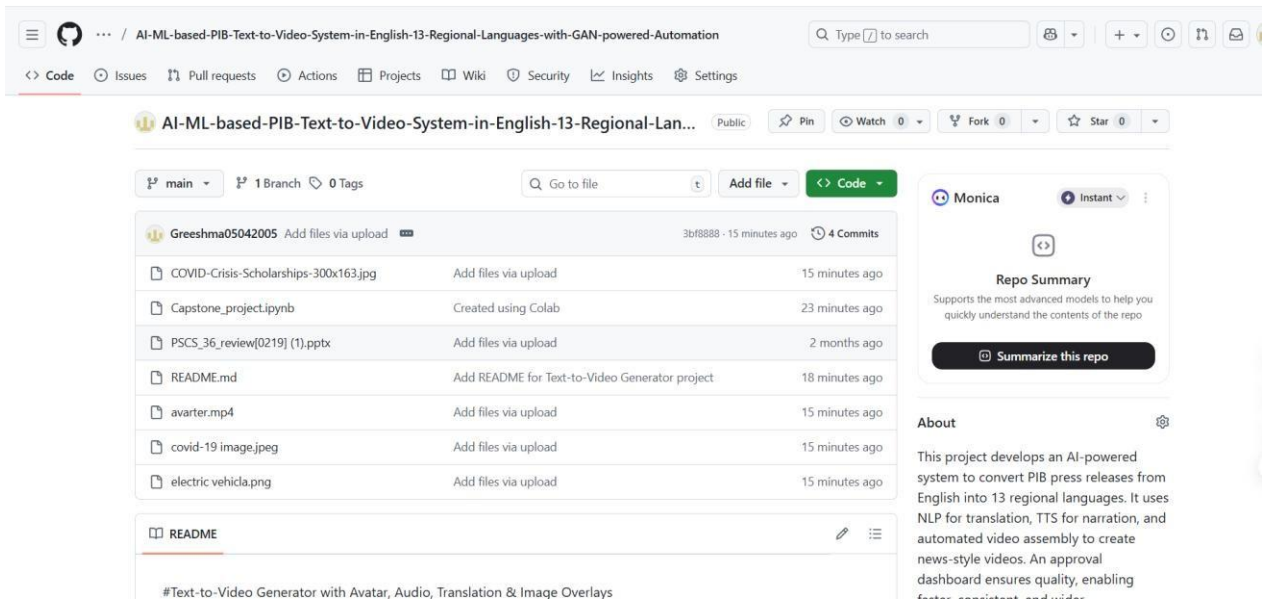


Fig 2:Github Repository