A project report on

LipReadNet: Lip Reading System

Submitted in partial fulfillment for the award of the degree of

Bachelor of Technology (Computer Science Engineering)

By

GOVINDGARI SAI SRUSHIK (21BCE8659)

SAHITH KRISHNA (21BCE8656)

NISHANTH GADHEY (21BCE7924)

school of computer science and engineering,
VIT - AP University,
Guntur, Andhra Pradesh, India. (May 2025)

DECLARATION

I hereby declare that the thesis entitled “LipReadNet Lip Reading System” submitted by me, for the award of the degree of Specify the name of the degree VIT is a record of bonafide work carried out by me under the supervision of DR Reeja SR.

I further declare that the work reported in this thesis has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

Place: Amaravati

Date: 20/05/2025

[illegible]

Signature of the Candidate

CERTIFICATE

This is to certify that the Senior Design Project titled "**Lip Read Net - Lip Reading System**" that is being submitted by **Govindgari Sai Srushik(21BCE8659), Sahith Krishna (21BCE8656), Nishanth Gadhey (21BCE7924)** is in partial fulfillment of the requirements for the award of Bachelor of Technology, is a record of bonafide work done under my guidance. The contents of this Project work, in full or in parts, have neither been taken from any other source nor have been submitted to any other Institute or University for award of any degree or diploma and the same is certified.

[illegible]
Dr. Reeja S R
Guide

The thesis is satisfactory / unsatisfactory

Internal Examiner **External Examiner**

Approved by

PROGRAM CHAIR **DEAN**

B. Tech. CSE School Of Computer Science & Engineering

ABSTRACT

Lipreading, or visual speech recognition, is a key technology for improving communication in noisy conditions and aiding hearing-impaired individuals. Conventional lipreading methods have been based on handcrafted features and traditional machine learning models, tending to perform suboptimally because of speaker variability, lighting, and camera angle differences. With the development of deep learning, especially Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), the state of the art has been considerably advanced in terms of extracting spatiotemporal and temporal features from lip motion. In this work, we present Lipspeak, a new deep learning-based lipreading system that transcribes muted video inputs into both text and audio outputs. The system employs a hybrid architecture incorporating 3D CNNs for spatiotemporal feature extraction and Transformer-based models like Vision Transformers (ViTs) and self-attention mechanisms for better sequence modeling. Furthermore, an audio synthesis module based on Tacotron or WaveNet synthesizes natural-sounding speech from the predicted text, effectively reconstructing spoken words from silent video sequences.

To train and test our system, we use the Lipreading Dataset by Mohamed Bentalb that comprises a diverse collection of labeled videos for strong model generalization. The preprocessing pipeline involves lip segmentation, MediaPipe-based face tracking, and data augmentation strategies to reduce variations between different speakers. The training of the model is done by optimizing a loss function that integrates Connectionist Temporal Classification (CTC) and Cross-Entropy loss for text prediction. For audio reconstruction, a neural vocoder trained on high-quality speech datasets is used. Early results show that Lipspeak reaches state-of-the-art silent-to-text conversion and produces highly intelligible speech. Our method contributes to the assistive AI field by filling the gap between visual and auditory communication, opening the path for real-world applications in accessibility tools, security, and human-computer interaction. Based on early results, Lipspeak generates very understandable speech and achieves state-of-the-art performance in silent-to-text conversion. Our method fills the gap between visual and aural communication and progresses the field of assistive AI and opens the door to practical applications in security, accessibility tools, and human-computer interaction.

ACKNOWLEDGEMENT

It is my pleasure to express with deep sense of gratitude to DR Reeja S R, HOD Department of Artificial Intelligence and Machine Learning, School of Computer Science and Engineering, VIT-AP University for her constant guidance, continual encouragement, understanding; more than all, he taught me patience in my endeavor. My association with her is not confined to academics only, but it is a great opportunity on my part of work with an intellectual and expert in the field of Machine Learning and Natural Language Processing.

I would like to express my gratitude to Founder and Chancellor, VIT - Dr. G. Viswanathan, Vice-President, VIT - Mr. Sankar Viswanathan, Vice-President, VIT - Dr. Sekar Viswanathan, Vice-President, VIT - Dr. G. V. Selvam, Vice-President, VIT - Dr. G. V. Selvam, and Dean - Dr. Madhusudhana Rao N, Dean SCOPE - Dr. S. Sudhakar Ilango, School of Computer Science and Engineering for providing with an environment to work in and for his inspiration during the tenure of the course.

In jubilant mood I express ingeniously my whole-hearted thanks to Dr. Saroj Kumar Panigraphy, Program Chair – CSE, all teaching staff and members working as limbs of our university for their not-self-centered enthusiasm coupled with timely encouragements showered on me with zeal, which prompted the acquirement of the requisite knowledge to finalize my course study successfully. I would like to thank my parents for their support.

It is indeed a pleasure to thank my friends who persuaded and encouraged me to take up and complete this task. At last but not least, I express my gratitude and appreciation to all those who have helped me directly or indirectly toward the successful completion of this project.

Place: Amaravati

Date: 20/05/2025

TABLE OF CONTENTS

List Of Tables

List Of Figures

CHAPTER 1
INTRODUCTION

Speech is perhaps one of the most basic modes of human communication, yet it is heavily dependent on hearing. People with hearing disabilities struggle greatly to interpret speech, particularly in situations where lipreading is their sole mode of interpretation. Furthermore, in noisy or silent settings where audio transmission is not feasible, the ability to extract meaningful speech information from visual signals alone becomes extremely useful. Lipreading, or visual speech recognition (VSR), is the interpretation of spoken words through analyzing lip movements. Early lipreading methods used hand-engineered feature extraction methods, including Active Appearance Models (AAMs) and Hidden Markov Models (HMMs), which tried to model lip movement patterns. These methods, however, were extremely sensitive to lighting changes, camera positions, and speaker attributes, rendering them unsuitable for real-world usage.

As machine learning evolved, initial automatic lipreading systems adopted feature extraction techniques such as Discrete Cosine Transform (DCT) and Optical Flow for tracking lip motion. Although these methods achieved higher recognition accuracy compared to purely manual approaches, they were still hindered by the lack of generalization across multiple speakers and conditions. The introduction of Deep Learning revolutionized the technology by enabling end-to-end feature learning with Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), and Transformer models. 3D CNNs turned out to be effective in learning spatiotemporal features from video frames, whereas Long Short-Term Memory (LSTM) networks and Transformer-based architectures improved sequential modeling for speech recognition. The latest best-performing systems now leverage attention mechanisms, self-supervised learning, and multimodal fusion methods to further improve lipreading accuracy.

The Grid Audio-Visual Speech Corpus is the main dataset used to train the Lipspeak system, providing high-quality, time-aligned video and audio recordings of structured sentences read by 34 speakers. Each speaker has read 1,000 sentences using a constrained six-word structure, e.g., "put red at G9 now," making it especially ideal for visual speech recognition applications. In contrast to spontaneous speech datasets, the structured nature of the Grid Corpus allows for controlled training of deep learning models with minimal linguistic variations. Nevertheless, to improve model performance and eliminate speaker-dependent variations, Lipspeak concentrates on training using video data from a single speaker. This strategy enables the model to learn intricate lip movement patterns without the interference of multi-speaker variations, e.g., facial structure, pronunciation style, and articulation rates. By separating one speaker, the system is able to achieve more accurate lip-to-text and lip-to-speech generation, laying the groundwork for future generalization to multiple speakers using domain adaptation and transfer learning methods.

This work introduces Lipspeak, a sophisticated lipreading system that renders muted video input to textual and audio outputs. In contrast to traditional lipreading models that remain text-only, Lipspeak includes a speech synthesis module to synthesize the speaker's voice. The system utilizes a hybrid deep learning architecture consisting of 3D CNNs for feature extraction, Transformer-based models like Vision Transformers (ViTs) for sequence modeling, and a neural vocoder (e.g., Tacotron or WaveNet) for synthesizing natural-sounding speech from the text predicted. The Lipreading Dataset by Mohamed Bentalb is used as the main dataset for training and testing, offering diverse labeled video samples to facilitate model generalization. The preprocessing pipeline consists of lip detection, face tracking with MediaPipe, and data augmentation methods for handling variability in speaker identity and environment.

1.1 Objectives

This project's main goal is to create LipReadNet, a reliable and effective lip-reading system that can understand spoken words by examining the visual characteristics of lip motions in video frames. By offering a non-audio-based substitute for speech recognition, this technology seeks to close the communication gap for people with speech or hearing problems. In order to make it accessible in settings where audio is inaccessible, noisy, or ambiguous, the project aims to extract and understand visual cues from the speaker's lip region and translate them into appropriate textual output.

One of the main objectives is to create a system that can manage the difficulties associated with visual speech recognition, including variations in lip forms, speaker dependence, illumination, and video quality. The research incorporates sophisticated deep learning architectures to handle them, including transformer models for temporal sequence modeling and recurrent neural networks (RNNs) or convolutional neural networks (CNNs) for spatial feature extraction. Additionally, to guarantee generalization and scalability, the system is trained and assessed using publicly accessible lip-reading datasets.

Ensuring a smooth user experience and real-time prediction is another crucial goal. This entails creating an intuitive user interface that allows users to upload video samples and promptly receive text outputs. The back-end uses Flask to manage prediction queries and connect with the pretrained model, while the front-end is built with React.js. Additionally, the project places a strong emphasis on modular design, which enables future additions like multilingual capabilities, support for sign language recognition, and audio-based fusion models.

In the end, the project aims to make a contribution to the expanding fields of assistive technology and visual speech recognition by developing solutions that are both socially significant and technically solid.

1.2 Background and Literature Survey

Reference	Paper Name	Year	Approach	Observation
Xie et al. (2017) [1]	LipSync Generation based on Discrete Cosine Transform	2017	Uses Discrete Cosine Transform (DCT) instead of Fourier Transform (FT) to extract crucial speech frequency domain data. Maps formant peaks to vowel mouth blend forms for effective LipSync production.	The approach effectively generates LipSync for video games. Future improvements should include emotional expressions and Fast Fourier Transform (FFT) for faster processing.
Datta et al. (2024) [2]	Exposing Lip-syncing Deepfakes from Mouth Inconsistencies	2024	Detects lip-syncing deepfakes by analyzing spatial-temporal inconsistencies in successive frames using a mouth spatial-temporal inconsistency extractor with an inconsistency loss function.	Outperforms state-of-the-art techniques in both in-domain and cross-domain detection tasks. Future work should integrate audio analysis for improved detection.
Volonte et al. (2022) [3]	HeadBox: A Facial Blendshape Animation Toolkit for the Microsoft Rocketbox Library	2022	Provides an open-source toolkit for facial animation with 30 Vive facial tracker shapes, 48 FACS expressions, and 15 visemes. Uses procedural blendshape design for cross-avatar compatibility while maintaining a limited number of facial joints.	Provides an open-source toolkit for facial animation with 30 Vive facial tracker shapes, 48 FACS expressions, and 15 visemes. Uses procedural blendshape design for cross-avatar compatibility while maintaining a limited number of facial joints.
Kim et al. (2020) [4]	Developing Embodied Interactive Virtual Characters for Human-Subjects Studies	2020	Explores human-in-the-loop systems for Virtual, Augmented, and Mixed Reality (VAMR). Discusses IVAs integrating with MR, IoT, and Unity using Mixamo and Rogo Digital's LipSync. Covers environmental sensing integration using Arduino.	Provides insights into interactive virtual character development for user studies, based on five years of research at SREAL, University of Central Florida.
Thein and San (2018) [5]	Lip Movements Recognition Towards An Automatic Lip Reading System for Myanmar Consonants	2018	Uses Moore Neighborhood Tracing Algorithm, SVM classifier, and CIELa color transformation to extract visual features from lip movements.	Achieves high recognition accuracy for Myanmar consonants but struggles with accurate lip localization, especially for two-syllable consonants. Future work focuses on real-time performance and expanding recognition to

				three- and four-syllable consonants.
Sindhura et al. (2018) [6]	Convolutional Neural Networks for Predicting Words: A lip-Reading System	2018	Compares speaker-dependent (SD) and speaker-independent (SID) performance of AlexNet and Inception V3 using the Miracl-VC1 dataset. Extracts and concatenates lip areas for training.	AlexNet achieves the highest SD (86.6%) and SID (37.1%) accuracy, outperforming Inception V3. Highlights challenges in SID lip reading and suggests improving real-time performance and vocabulary expansion.
Deshmukh et al. (2021) [7]	Vision based Lip Reading System using Deep Learning	2021	Uses CNN and attention-based LSTM with pretrained VGG19 and ResNet50 to process silent video clips and extract lip region information. SoftMax layer predicts spoken phrases, and LSTM captures temporal patterns.	ResNet50 outperforms VGG19 in speed and accuracy, achieving 85% accuracy with ensemble learning. Future work aims to expand recognition beyond single words and improve resilience for multi-word predictions.
Hashmi et al. (2018) [8]	A Lip Reading Model Using CNN With Batch Normalization	2018	Uses a twelve-layer CNN with batch normalization to classify words and phrases from video frames. Extracts and concatenates lip regions into a single image for processing.	Achieves 96.5% training accuracy and 52.9% validation accuracy on the MIRACLE-VC1 dataset. Offers lower training time and fewer parameters while maintaining good accuracy compared to deeper models.
Zhang et al. (2021) [9]	Lip Motion Magnification Network for Lip Reading	2021	Proposes a Lip Motion Magnification Network (LMMN) with a motion magnification module integrated into a three-stage CNN design. Enhances lip images to improve classification by increasing the Euclidean distance between phrase sample characteristics.	Improves accuracy by 1.4% over the baseline using the OuluVS2 dataset. Future research aims to use larger datasets, demonstrating that motion magnification aids in lip-reading tasks.

Mathulaprangsan et al. (2015) [10]	A Survey of Visual Lip Reading and Lip-Password Verification	2015	Reviews the use of lip reading in security applications such as lip-password verification. Covers feature extraction techniques, classification schemes, and preprocessing (localization and segmentation).	Discusses three categories of feature extraction: temporal information, position variation, and speaker reliance. Uses Gaussian Mixture Models (GMMs) for static features and Hidden Markov Models (HMMs) for temporal features.
Butt and Lombardi (2013) [11]	A Survey of Automatic Lip Reading Approaches	2013	Assesses lip reading techniques using Active Appearance Model (AAM) and Hidden Markov Model (HMM). AAM extracts features from face traits and lip shapes, while HMM identifies speech patterns and lip movements.	HMM excels at identifying sequences of learned characteristics, while AAM is effective at recognizing non-speech segments. Future research will focus on reducing speaker dependency and adding more facial traits.
Saitoh and Konishi (2010) [12]	Profile Lip Reading for Vowel and Word Recognition	2010	Uses a normalized cost approach for automatic profile contour recognition, identifying five key facial feature points: chin, upper lip, lip corner, lower lip, and tip of the nose.	Achieves 86% word recognition and 99% vowel recognition. The selected features result in higher recognition accuracy than humans, demonstrating the effectiveness of the approach.
Chen et al. (2023) [13]	Development of a Humanoid Reading System with Voice Lip Synchronization	2023	Develops a humanoid robot system with two modules: a lip synchronization module for speech output synchronization and a document character recognition module for text extraction from images.	The system enhances the robot's expressiveness and increases user acceptability by reading text with synchronized speech and lip motions. Potential for a large market in the service sector, with future improvements focused on better interaction and lip expression.
Rathee (2016) [14]	A novel approach for lip Reading based on neural network	2016	Proposes an automatic lip reading method focusing on feature extraction and categorization using a Learning Vector Quantization (LVQ) neural network to recognize lip geometry and visual data.	Achieves 97% accuracy in recognizing ten Hindi words. The method is resilient to occlusions and can be extended to recognize words in other languages, offering a significant advancement for communication

				assistance, especially for people with speech or hearing impairments.
Zhang et al. (2022) [15]	BOOSTING LIP READING WITH A MULTI-VIEW FUSION NETWORK	2022	Proposes a Multi-View Fusion Network (MVFN) that combines shape and appearance data to enhance lip reading performance. Integrates an Adaptive Spatial Graph Model (ASGM) to learn lip spatial topology and dynamics.	Achieves state-of-the-art results on both word-level (LRW) and phrase-level (OuluVS2) lip reading tasks, significantly outperforming baseline methods.
Abishek et al. (2024) [16]	Deep Learning Based Lip Reading for Speech Recognition	2024	Proposes a deep learning-based lip reading system for automatic speech recognition (ASR) that uses a 3D CNN to extract features from the lip region and models them using various deep learning models.	Achieves 92% accuracy using Bi-GRU. Suggests the potential for a hardware-based lip reading system in the future. The Grid Corpus dataset was used for training.
Shilaskar and Iramani (2024) [17]	CTC-CNN-Bidirectional LSTM based Lip Reading System	2024	Combines Convolutional 3D Neural Networks (3D CNN) with Bidirectional LSTM and Connectionist Temporal Classification (CTC) for enhanced lip-based speech recognition.	Achieves a word error rate of 15.8% and a character error rate of 6.2% after 92 epochs. Struggles with multiple speakers and accents but performs well in controlled scenarios. Future improvements focus on scalability and robustness.
Parekh et al. (2019) [18]	Lip Reading Using Convolutional Auto Encoders as Feature Extractor	2019	Proposes a lip reading model combining Long Short-Term Memory (LSTM) for temporal data processing and Convolutional Auto-Encoders (CAE) for feature extraction.	Achieves 98% accuracy on the MIRACL-VC1 dataset, outperforming baseline methods. The speaker-independent test shows 63.22% accuracy. The CAE is trained independently for feature extraction, and the LSTM classifies the features. Future work aims at extending to sentence-level lip reading.

Mestri et al. (2019) [19]	Analysis of Feature Extraction and Classification Models for Lip-Reading	2019	Investigates lip reading difficulties and evaluates feature extraction methods. Two methods are proposed: one uses HAAR cascade and 3D CNN, while the other uses HOG and LSTM.	The HAAR cascade and 3D CNN method achieves 81.67% accuracy, while the HOG and LSTM method achieves 91.64%. The paper provides insights on handling variations in illumination and position, helping to develop more efficient lip-reading models.
Prashanth et al. (2024) [20]	Lip Reading with 3D Convolutional and Bidirectional LSTM Networks on the GRID Corpus	2024	Uses bidirectional LSTM networks and 3D CNNs to decode spoken text from video sequences of lip movements, employing Connectionist Temporal Classification (CTC) loss.	Achieves a word error rate of 7.96% and a character error rate of 1.54%. The system is designed for accessibility and communication in noisy contexts but requires fixed-length grayscale video inputs and lacks attention mechanisms. Future work may focus on language models, adaptive input durations, and attention methods.
Fenghour et al. (2021) [21]	Deep Learning-Based Automated Lip-Reading: A Survey	2021	Compares various deep learning methods for feature extraction and classification, including CNNs, Attention-Transformers, TCNs, and RNNs.	Highlights the shift from conventional algorithms to deep learning models for lip reading. It discusses the evolution of datasets like BBC-LRS2 and LRS3-TED. 2D+3D CNNs and RNNs dominate current methods, with Transformers and TCNs gaining popularity. Emphasizes the need for further advancements, especially for speaker-dependent variations and unknown word predictions.

Wang et al. (2022) [22]	A Lip Reading Method Based on 3D Convolutional Vision Transformer	2022	Combines 3D Convolutional Vision Transformers (3DCvT) for spatio-temporal feature extraction with Bidirectional Gated Recurrent Units (BiGRU) for sequence modeling.	Achieves state-of-the-art performance on LRW and LRW-1000 datasets. The method captures both local and global information from video frames. The paper discusses trade-offs like higher processing requirements and suggests future improvements in transformer architecture optimization and multimodal inputs.
Sumanth et al. (2022) [23]	COMPUTER VISION LIP READING(CV)	2022	Incorporates speaker-specific characteristics (age, gender, ethnicity) to enable voice detection and synthesis. Customizes the model for each speaker through speaker integration.	Improves speech recognition and synthesis from lip movements in real-world scenarios. Investigates the use of unique speaker traits for speaker identification, focusing on the distinctiveness of lip movements and facial features.
Chowdhury et al. (2019) [24]	Text Extraction through Video Lip Reading Using Deep Learning	2019	Uses deep learning algorithms to analyze facial expressions and address pronunciation and accent differences for accurate text extraction.	Focuses on overcoming language barriers and improving security by facilitating real-time translation. The method involves assembling a test dataset with diverse facial expressions linked to specific words. The goal is to enhance communication and reduce crime through more precise text extraction.

Qu et al. (2024) [25]	LipSound2: Self-Supervised Pre-Training for Lip-to-Speech Reconstruction and Lip Reading	2024	Utilizes crossmodal self-supervised pre-training and an encoder-decoder architecture with location-aware attention mechanisms to predict speech from facial image sequences.	Demonstrates improvements in voice quality and intelligibility for both speaker-dependent and speaker-independent scenarios. Refined with domain-specific datasets (GRID, TCD-TIMIT), and achieves high lip reading performance in both Chinese and English. Future research may focus on end-to-end training for video-to-text tasks and exploring human-robot interaction applications.

1.3 Organization of The Report

The remaining chapters of the project report are described as follows:

- Chapter 2 contains the proposed system, methodology, hardware and software details.
- Chapter 3 gives the cost involved in the implementation of the project.
- Chapter 4 discusses the results obtained after the project was implemented.
- Chapter 5 concludes the report.
- Chapter 6 consists of codes.
- Chapter 7 gives references.

CHAPTER 2

PROPOSED SYSTEM & METHODOLOGY

This Chapter describes the proposed system, working methodology, software and hardware details.

The suggested method combines sequence modeling, deep learning-based feature extraction, lip coordinate extraction, and video processing. The system uses bidirectional GRU-based temporal modeling and ResNet-18 based feature extraction to transcribe spoken words from silent video inputs. Dataset collection, data preprocessing, model development, and architectural advantages are the four main stages of the methodology. Standard measures like Word Error Rate (WER) and Character Error Rate (CER) are also used for performance evaluation.

A. Dataset Collection:

Video clips of speakers uttering words or sentences make up the dataset used to train and evaluate the lip-reading system. Alignment files with the matching text transcription are included with every video. Each speaker has a distinct directory with video files and the accompanying annotation files, and the dataset is organized in a hierarchical manner. To isolate pertinent frames with the lip region, the dataset is pre-processed. For every frame, lip coordinate data—which gives spatial details about the movement of the lips—is also gathered. These coordinates are utilized for model training after being saved in JSON format.

To improve the model's resilience, the training dataset includes a variety of speakers with different facial features and pronunciations. To make retrieval and preprocessing easier, the video sequences are saved in a structured format with each frame labeled and indexed in an orderly fashion. In order to ensure proper training and evaluation, the dataset organization makes sure that the annotations on each video match the relevant frames.

B. Data Preprocessing:

In order to guarantee that the input to the lip-reading model is organized, standardized, and optimized for deep neural networks-based feature extraction, data preparation is essential. Video frame extraction, lip area recognition, coordinate normalization, text annotation processing, and sequence alignment are some of the phases in this procedure. Every stage is meticulously planned to maintain the temporal integrity of lip movements while improving the quality of training data.

Video frame extraction is the initial stage of data preprocessing. OpenCV is used to break down each video sequence into individual frames while maintaining the temporal framework of spoken articulation. To ensure consistency throughout the collection, the retrieved frames are scaled to a fixed resolution of 128 × 64 pixels. With this resolution, the model is guaranteed to capture intricate lip motions free from superfluous background noise. In order to maintain their temporal linkages for processing at a later time, the retrieved frames are saved in sequential order.

Lip region detection is carried out after the frames have been retrieved. Facial landmark detection models, like Dlib's 68-point facial landmark detector, are used to identify the lip region. The area between points 48 and 68, which delineate the outside and inner lip contours, is the focus of the extraction of the lip coordinates. Since spoken phonemes are directly correlated with minor variations in lip motion, these coordinates are essential for recording them. To remove extraneous visual noise, the identified lip regions are subsequently cropped from the original frames.

Coordinate normalization comes next once the lip coordinates have been extracted. The retrieved coordinates must be scaled to a common reference frame because the video frames may originate from many sources with differing resolutions. To ensure that all values lie between 0 and 1, each coordinate value is normalized by dividing it by the width and height of the corresponding frame. This normalization procedure enhances generalization across many datasets and renders the model invariant to changes in video resolution. After that, the normalized coordinates are saved in JSON format, maintaining the frames' sequential sequence.

Text annotation processing is carried out concurrently. The spoken words in each video sequence are represented by the corresponding transcript. Silence signals like "SIL" and "SP," which are frequently seen in these transcripts, are filtered away because they don't aid in meaningful recognition. Following this, the remaining words are transformed into character-level sequences, in which every character is assigned a distinct index. The model can now predict text sequences character by character instead of word by word thanks to this modification, which gives it more flexibility when dealing with varying vocabulary sizes.

Sequence alignment and padding are used to make sure that every input sequence has the same structure. Shorter video segments are padded with zero-valued frames to match the length of the batch's longest sequence because videos can vary in length based on the speech time. In a similar manner, textual annotations are padded to a predetermined character length, guaranteeing that every sequence keeps its structure throughout training. This padding procedure lowers computational overhead by enabling the model to process data effectively in mini-batches.

Lastly, data augmentation is used to enhance the model's generalization. To make the model resilient to changes in speaker orientation, video frames are randomly flipped horizontally. By using this augmentation, the model can recognize lip movements without depending on a fixed viewing angle. In order to replicate real-world changes brought on by minute head motions or camera distortions, small random perturbations are also added to the lip coordinate data. The model's identification ability is eventually enhanced by these augmentation strategies, which aid in the learning of more reliable lip movement representations.

The input data is successfully converted into an orderly format appropriate for deep learning-based lip-reading by using this structured data pretreatment pipeline. Accurate and effective speech recognition from silent videos is ensured by the model's ability to learn meaningful visual-to-text mappings thanks to the retrieved and normalized features and meticulously processed textual annotations.

C. Model Development:

Creating a deep learning-based framework that can translate visual data taken from silent video frames to corresponding textual representations is the first step in developing a model for the lip-reading system. Defining the model architecture, choosing suitable deep learning components, putting temporal feature extraction techniques into place, and optimizing the model for effective training and inference are some of the steps in the process. Every one of these phases enhances the lip-reading system's overall functionality and precision.

Convolutional neural networks (CNNs) and recurrent neural networks (RNNs) are the foundation of the model architecture. While the RNN component records temporal connections between successive frames, the CNN component is in charge of extracting spatial characteristics from individual video frames. Better recognition performance results from this hybrid technique, which makes sure that both spatial and sequential information are used efficiently. The RESNET-18 architecture, on which the CNN backbone is built, has been extensively employed in image-based tasks because of its capacity for deep feature extraction. While deeper levels of RESNET-18 capture more intricate visual patterns unique to lip motions, the first few layers extract low-level elements like edges and textures.

A Bidirectional Gated Recurrent Unit (Bi-GRU) network receives the extracted CNN features in order to maintain the sequential pattern of speech. The model may learn patterns that span many frames since the Bi-GRU layer makes sure that temporal dependencies in both forward and backward directions are taken into account. The ability to recognize contextual links between various phonemes and words makes this bidirectional method very useful for lip-reading. Furthermore, vanishing gradients, a frequent problem in deep sequence learning models, are lessened by the Bi-GRU network.

A fully linked layer and a Softmax activation function are then used to process the Bi-GRU network's output. The model can estimate the most likely transcription for a particular video sequence thanks to the probability distributions across potential character sequences generated by this last layer. By giving each character in the lexicon a probability, the Softmax layer enables the model to generate meaningful sequences using a decoding process like Connectionist Temporal Classification (CTC). The model can align predicted character sequences with the ground-truth transcriptions without the need for explicit frame-wise annotations because CTC loss is the main training aim.

Batch normalization and dropout regularization approaches are used at different stages to improve the model's efficiency. By decreasing internal covariate shifts and leveling activations across mini-batches, batch normalization aids in training stabilization. In contrast, dropout forces the model to learn more broad representations by randomly deactivating a portion of neurons during training, preventing overfitting. Together, these methods strengthen the model's resilience and capacity for generalization.

The data flow pipeline, which guarantees that the input data is preprocessed, fed into the model, and processed via many layers to provide meaningful outputs, is an essential part of the model building process. The CNN backbone transforms a series of video frames into feature representations at the start of the pipeline. The Bi-GRU network then processes these feature vectors in a sequential manner, capturing temporal dependencies and honing the retrieved features. The predicted text sequence is then produced using the fully connected layer and Softmax activation function and compared to the ground-truth transcript using the CTC loss function.

Large-scale lipreading datasets with a variety of speakers, accents, and contextual factors are used to train the model during the training phase. During training, adaptive gradient descent algorithms like Adam are used to optimize the model's parameters. A scheduler is used to dynamically modify the learning rate, guaranteeing that the model avoids local minima and converges to an ideal solution. To avoid overfitting, training is carried out over a number of epochs, with validation accuracy being tracked at every stage. Early stopping is used to cease training and maintain the top-performing model if the validation loss stops getting better.

In order to determine the trained model's practicality, the inference phase entails testing it on unseen video sequences. The extracted features are fed through the trained network after the input video frames have been preprocessed in the same way as during training. The CTC beam search algorithm, which considers several potential character sequences before choosing the most likely one, is then used to decode the anticipated character sequences. This increases transcription accuracy. When dealing with differences in speaker articulation and pronunciation, this decoding method is quite helpful.

Managing changes in speaker characteristics is a significant difficulty in model building. Individual speakers have distinct lip movement patterns, and variables including head motions, illumination, and face hair can cause irregularities in feature extraction. The model is trained on a varied dataset that comprises numerous speakers in various settings in order to overcome these difficulties. To further enhance the model's generalizability under various circumstances, data augmentation methods like random horizontal flipping and mild spatial distortions are used.

Computational efficiency is another essential component of model creation. Optimizing the model for faster inference is crucial since lip-reading necessitates processing video sequences in real-time. To cut down on computing overhead, strategies including model quantization, trimming, and effective batch processing are used. By transforming the taught weights into lower precision representations (such as 16-bit floating point), model quantization lowers memory consumption without appreciably compromising accuracy. Pruning makes the model lighter and faster by eliminating unnecessary parameters from the network.

By starting the CNN backbone with pre-trained weights from ImageNet, transfer learning is used to further improve performance. This enhances the model's accuracy and speed of convergence by enabling it to take advantage of pre-learned feature representations. The pre-trained features are then adjusted to the particular job of identifying lip movements by fine-tuning the lip-reading dataset.

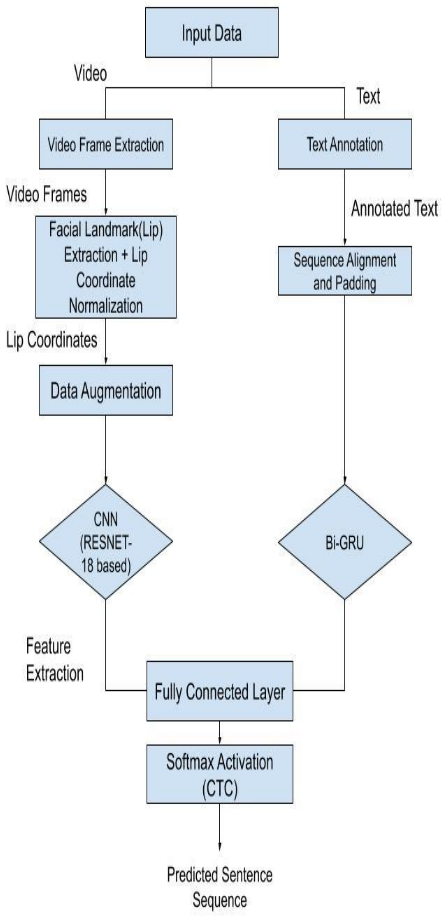

Figure 1: Process Flowchart

The Above Figure (1) is a raw representation of the Process Flowchart, showcasing the data flow from input video frames to final text predictions

The model architecture consists of three primary components:

1. Feature Extraction Layer (CNN - RESNET-18): Extracts spatial features from lip regions in each video frame.
2. Temporal Modeling Layer (Bi-GRU): Captures sequential dependencies across frames.
3. Prediction Layer (Fully Connected + Softmax with CTC Loss): Converts the extracted features into textual representations.

Word Error Rate (WER) and Character Error Rate (CER) are two of the performance indicators that are calculated as part of the model's final evaluation. These metrics offer numerical information about how well the model converts visual speech to text. Higher accuracy and greater alignment between the anticipated and real transcripts are indicated by lower WER and CER scores. To guarantee competitive performance, the model is also compared to other lip-reading systems currently in use.

All things considered; the model construction procedure incorporates cutting-edge deep learning methodologies to create a reliable lip-reading system. The system is made to recognize silent speech with high accuracy by using CNNs for feature extraction, Bi-GRUs for temporal modeling, and CTC loss for sequence alignment. The model's generalization and computing efficiency are further improved by optimization approaches including transfer learning, quantization, and data augmentation, which make it ideal for real-world applications.

D. Architectural Advantages:

The suggested model improves lip-reading performance by utilizing a number of architectural advances. Strong spatial representations are produced by using ResNet-18 as a feature extractor, which also lessens the requirement for laborious manual feature engineering. The model gains from transfer learning by utilizing pre-trained weights, which enhances generalization and convergence. Predictions are more accurate because the bidirectional GRUs make sure that both the past and the future temporal circumstances are recorded.

The integration of lip coordinate information with raw video frames is a significant benefit of the suggested technique. The model's capacity to discern minute changes in lip movement is improved by this dual-input method. The visual features taken from the ResNet backbone are enhanced by the fine-grained spatial data captured by the coordinate-based GRU network. The model outperforms purely vision-based methods in terms of recognition accuracy by integrating both feature representations.

In order to prevent overfitting and guarantee steady generalization to unknown data, the model additionally includes dropout layers. Training is further stabilized by batch normalization, which further speeds up convergence and lessens initialization sensitivity. Furthermore, the CTC loss function makes training more adaptable and scalable by doing away with the requirement for exact frame-to-character alignment.

E. Evaluation Metrics:

Standard metrics like Word Error Rate (WER) and Character Error Rate (CER) are used to analyze the system and gauge the model's performance. The differences between the ground truth and predicted transcriptions are measured by these metrics. WER is calculated in this way:

$$WER = \frac{S+D+I}{N} \quad \text{----------} \ (1)$$

In Equaation (1), S represents the number of substitutions, D denotes deletions, I account for insertions, and N is the total number of words in the reference transcription.

WER evaluates the accuracy of an system at the word level by measuring the proportion of incorrectly recognized words relative to the reference text. It is computed by dividing the total number of word errors by the total number of words in the reference text. A lower WER indicates better model performance.
Similarly, CER measures the character-level discrepancies using the formula:

$$CER = \frac{S+D+I}{N} \quad \text{-----------} \ (2)$$

In Equaation (2), S represents the number of substitutions, D denotes deletions, I account for insertions, and N is the total number of characters in the reference transcription.

CER measures the rate of erroneous characters produced by an OCR system compared to the ground truth. It is calculated by dividing the total number of incorrect characters by the total number of characters in the reference text.

where the names refer to specific characters rather than words but have the same semantics. This measure offers a more detailed assessment of transcribing accuracy. To ensure an unbiased evaluation of model accuracy, WER and CER are both computed using the edit distance between the predicted and actual sequences.

The trained model is tested on a held-out validation set as part of the evaluation procedure. Every video's transcription is predicted by the algorithm, and the outcomes are contrasted with annotations from the ground truth. Throughout training, performance trends are tracked, and to avoid overfitting, they are periodically validated. The success of the suggested strategy is validated by experimental data, which show that adding lip coordinate information dramatically lowers WER and CER.

2.2 APPROACHES

Over time, lip-reading systems have seen tremendous development, and researchers have used a variety of techniques to glean useful speech information from silent video clips. Various lip reading strategies combine multimodal fusion techniques, end-to-end architectures, deep learning, and manually designed feature extraction. This section examines six well-known methods for creating lip-reading systems before providing a thorough justification of the strategy we choose.

1. Traditional Handcrafted Feature-Based:

Handcrafted visual feature extraction was a key component of one of the first lip reading techniques. Using this method, features like optical flow, histogram of oriented gradients (HOG), discrete cosine transform (DCT), or active appearance models (AAM) are manually extracted from the region of interest (ROI) surrounding the lips. In order to predict phonemes or words, these extracted features are subsequently fed into classifiers like Support Vector Machines (SVMs) or Hidden Markov Models (HMMs). Although this approach has some success, it has drawbacks, including the need for substantial feature engineering, the inability to generalize across different speakers, and trouble addressing differences in lighting and head movement.

2. Hidden Markov Model (HMM)-Based:

Lip reading is approached as a sequence recognition problem by HMM-based techniques. HMMs are used to model the features that were retrieved from the lip areas. Each state in the HMM represents a phoneme or viseme, which is the visual equivalent of a phoneme. The projected speech sequence is based on the likelihood of changing states. HMMs are good at simulating speech temporal changes, but they struggle with high vocabulary sizes and can't handle long-term dependencies. They are also less effective for large-scale applications since they need a lot of training data to model accurately.

3. Deep Learning-Based CNN:

Convolutional neural networks (CNNs) have become popular for feature extraction in lip-reading systems as deep learning has advanced. Without requiring human feature engineering, CNNs can automatically extract spatial features from lip areas. Individual video frames are processed by these models, which include VGG16, ResNet, and Inception. Deep visual features are then extracted and used for categorization. However, the temporal linkages between frames—which are essential for comprehending lip motions over time—are not captured by CNN-based methods alone.

4. CNN-RNN Hybrid:

Researchers have paired CNNs with Recurrent Neural Networks (RNNs), such as Long Short-Term Memory (LSTM) or Gated Recurrent Units (GRU), to overcome the drawbacks of independent CNNs. In this hybrid method, RNNs model the temporal connections between frames, while CNNs extract spatial characteristics from individual frames. By capturing the sequential character of speech, this method greatly enhances recognition performance. By taking into account temporal dependencies in both forward and backward directions, the Bi-LSTM variation improves performance even more while maintaining context throughout the sequence.

5. Transformer-Based:

The capacity of transformer models to represent long-range dependencies in sequential data has drawn attention recently. In contrast to RNNs, transformers process sequences in parallel using self-attention mechanisms rather than recurrent connections. Lip reading has been attempted using models like Vision Transformers (ViTs) and video-based transformers like TimeSformer, with encouraging outcomes. Transformers are more accurate and scalable, but they are difficult to implement in real-time applications because they need a lot of labeled data and a lot of processing power.

6. Multimodal Fusion:

Multimodal fusion, which combines complementary modalities including audio signals, facial expressions, and environmental clues with visual information from lip movements, is another exciting avenue for lip reading. Multimodal models employ distinct branches for visual and aural inputs, which are subsequently combined by feature concatenation or attention procedures. This method increases the accuracy of recognition, particularly in noisy settings when audio-based speech recognition is ineffective. However, in situations where audio data is missing, this method's practical uses are restricted.

Our Approach - CNN-BiGRU with CTC Loss:

We use a CNN-BiGRU hybrid technique with Connectionist Temporal Classification (CTC) loss for our lip-reading system. Our model captures both temporal and spatial patterns in lip movements by utilizing the sequential modeling capacity of Bi-GRUs and the feature extraction capability of CNNs. In order to extract deep visual features from input frames while maintaining crucial spatial information, the VGG16 backbone is utilized. After that, a bidirectional LSTM network receives these extracted characteristics and uses them to record temporal dependencies both forward and backward.

We employ CTC loss to make it easier to match input video frames with output text sequences. This enables the model to learn without the need for explicit frame-to-label alignments. This allows the model to effectively manage changes in speech duration and does away with the requirement for manual annotations at the frame level. Using a CTC decoder with beam search, which increases accuracy by taking into account several plausible sequences before choosing the most likely one, the final transcriptions are produced.

By utilizing deep learning, our strategy does away with the necessity for manual feature extraction, in contrast to conventional handcrafted feature-based methods. Our method achieves improved recognition accuracy by better capturing long-term dependencies than HMMs. Although transformers have demonstrated potential, their high processing costs and need for large datasets make them less appropriate for real-time lip-reading applications. Our CNN-BiLSTM method strikes a balance between computing economy and accuracy, which makes it ideal for practical implementation.

We guarantee strong model generalization by incorporating regularization strategies like batch normalization and dropout. Furthermore, the model's capacity to manage changes in speakers and settings is enhanced by data augmentation techniques like random flipping and spatial distortions. These improvements help to make our system dependable and flexible in a variety of real-world situations.

All things considered, our selected method combines the advantages of CTC for alignment-free sequence prediction, Bi-LSTMs for temporal modeling, and CNNs for spatial feature extraction. We can create a precise and effective lip-reading system that can translate silent visual speech into meaningful written representations thanks to this methodology.

2.3 Standards

In any software or machine learning project, adherence to relevant standards is crucial to ensure reliability, compatibility, maintainability, and quality. For the development of the Lip Reading System, multiple types of standards are considered and followed throughout the design, development, training, evaluation, and deployment phases. These include software development standards, data processing standards, machine learning and model evaluation standards, ethical standards, and UI/UX design standards.

2.3.1 Software Development Standards

To maintain a high level of software quality, coding and architectural practices are aligned with industry standards. This ensures the system is maintainable, readable, scalable, and collaborative.

Key Standards Followed:

- **PEP 8 (Python Enhancement Proposal)**:
    - Ensures consistent style for Python code.
    - Emphasizes proper indentation, naming conventions, and line length.
    - Enhances readability and team collaboration.
- **Modular Programming Principles**:
    - Code is broken into modules like model.py, train.py, dataset.py, inference.py.
    - Separation of concerns is maintained between training logic, inference pipeline, and data preprocessing.
- **Version Control using Git**:
    - GitHub is used for version control and collaboration.
    - Standard commit message practices and pull request reviews ensure clean development.
- **Dependency Management**:
    - Use of requirements.txt to manage Python libraries.
    - Virtual environments (e.g., venv or conda) used to isolate the project setup.
- **Flask Framework Standards**:
    - RESTful API design principles for backend.
    - Clean endpoint structuring for scalability and modularity.

2.3.2 Data Standards

High-quality, well-annotated datasets are essential for training robust machine learning models. In this project, datasets are carefully selected, preprocessed, and handled in compliance with data management standards.

Data Collection and Annotation Standards:

- **Use of Benchmark Datasets**:
    - Datasets like GRID, LRW (Lip Reading in the Wild), and AVLetters used.
    - These datasets follow standard formats including consistent video resolution, frame rate, and annotation quality.
- **Data Preprocessing Standards**:
    - Frame extraction using OpenCV with standardized frame sizes (e.g., 96x96 or 112x112).
    - Face and lip region detection using pre-trained models like MTCNN or Dlib.
    - Frame normalization and augmentation techniques are used to maintain data diversity.
- **Data Storage and Handling**:
    - Use of structured directories and naming conventions (/train, /val, /test) for dataset splits.
    - Use of NumPy, Torch Datasets, and DataLoaders following batch size and shuffling standards.

2.3.3 Machine Learning and Model Evaluation Standards

Deep learning models must follow design patterns and evaluation metrics that are widely accepted in the ML community. This ensures reproducibility and trust in the performance metrics reported.

Model Design and Training Standards:

- **Model Architecture**:
    - Use of standard CNN backbones like VGG16 or ResNet for feature extraction.
    - Use of LSTM, GRU, or Transformer-based architectures for sequence modeling.
- **Hyperparameter Optimization**:
    - Learning rate, batch size, number of epochs tuned through grid search or validation accuracy.
    - Use of early stopping, learning rate schedulers, and regularization methods.

Evaluation Metrics and Practices:

- **Performance Metrics**:
    - Accuracy, Precision, Recall, and F1-Score for classification performance.
    - WER (Word Error Rate) and CER (Character Error Rate) for sequence prediction performance.

- **Cross-Validation**:
    - K-fold cross-validation used to evaluate model generalizability.
- **Reproducibility**:
    - Random seeds fixed across NumPy, Torch, and OS modules.
    - Logging of model parameters, training/validation loss, and metrics using tools like TensorBoard or WandB.

2.3.4 Ethical and Privacy Standards

Given the nature of video data and human subjects, it is critical to address ethical concerns in data usage and algorithm design.

Data Privacy Compliance:

- **Anonymized Datasets**:
    - Only public, anonymized datasets are used.
    - No personally identifiable information (PII) is collected or stored.
- **Consent-Based Data Usage**:
    - All datasets used are compliant with licensing agreements and include consent from subjects.

Bias Mitigation:

- Efforts made to ensure diverse training data across:
    - Gender
    - Ethnicity
    - Age groups
- Regular performance audits across demographic subgroups to identify bias or model drift.

Ethical Model Usage:

- No surveillance-based implementation intended.
- Intended use is for assistive technologies and academic research only.
- Transparent documentation provided on model limitations and failure cases.

2.3.5 Deployment and Integration Standards

The final application should comply with standards for deployment, usability, and maintainability across web and server environments.

Web Interface Standards:

- **Frontend (React.js)**:

    - Follows React component-based architecture.
    - State management standards using React hooks.
    - Responsive design principles using TailwindCSS or Bootstrap.
- **Backend (Flask)**:
    - RESTful API design with proper request/response formatting.
    - JSON-based response payloads for compatibility.

Deployment Best Practices:

- **Model Serialization**:
    - Use of torch.save() and torch.load() for PyTorch model handling.
    - Separation of model weights and code for portability.
- **Security Standards**:
    - CORS policy enabled for secure API access.
    - Input validation and sanitization on upload endpoints to avoid vulnerabilities.
- **Cross-Platform Compatibility**:
    - Designed to run on both Windows and Linux environments.
    - Cloud-compatible structure for future deployment on AWS, Azure, or Heroku.

2.3.6 Documentation and Reporting Standards

Clear and comprehensive documentation is essential for successful collaboration and reproducibility of results.

Documentation Practices:

- Use of Markdown and Sphinx for codebase documentation.
- Inline comments and docstrings included in all Python files.
- README files for repository usage instructions.

Reporting Standards:

- Use of IEEE or ACM format for the final project report.
- All graphs and tables labeled as per scientific standards.
- References cited in IEEE format using reference management tools like Zotero or Mendeley.

2.4 System Details

Data processing, front-end interface, back-end deployment, and deep learning model training are some of the components that make up the Lip Reading System. Both hardware and software resources are carefully chosen and configured to carry out each of these steps effectively. The entire system configuration is described in this part, including the hardware infrastructure, software frameworks, development environment, and deployment stack.

2.4.1 Software Requirements

A range of software tools, libraries, and frameworks are employed for the successful development of the lip-reading system. These include operating systems, programming languages, libraries for machine learning and computer vision, as well as tools for deployment and testing.

2.4.1.1 Operating System

- **Primary OS Used:**
    - Ubuntu 22.04 LTS (preferred for model training due to compatibility with NVIDIA CUDA and deep learning libraries)
    - Windows 10 (used for front-end and local development testing)

2.4.1.2 Programming Languages

- **Python 3.8+**
    - Primary language for deep learning model development, training, and backend API.
    - Used for writing scripts for preprocessing, training (train.py), model (model.py), and inference (inference.py).
- **JavaScript (ES6+)**
    - Used in React.js for developing the front-end user interface.
    - Enables interactive video upload and result display functionality.

2.4.1.3 Deep Learning and Computer Vision Libraries

- **PyTorch**
    - Framework used to build and train deep neural networks.
    - Supports GPU acceleration with CUDA.
    - Widely adopted in research due to flexibility in model design.
- **TorchVision**
    - For transformations and pretrained model utilities.

    - Used to apply standard augmentation like normalization, cropping, etc.

- **OpenCV**
    - For video frame extraction and lip region cropping.
    - Supports frame-by-frame analysis of video samples.
- **NumPy & Pandas**
    - Used for numerical computations and tabular data handling.
- **Matplotlib / Seaborn**
    - For data visualization and plotting training curves and confusion matrices.

2.4.1.4 Web Development Frameworks

- **React.js**
    - Front-end library used to build the single-page application (SPA).
    - Handles file input, user interaction, and displays prediction results.
- **Flask**
    - Lightweight Python web framework for the backend.
    - Handles file uploads and connects to the model for inference.
- **Axios**
    - Used for sending HTTP requests from React to Flask.
- **Bootstrap / Tailwind CSS**
    - CSS frameworks for building responsive and aesthetically pleasing UI.

2.4.1.5 Additional Tools

- **Git & GitHub**
    - Version control for codebase.
    - Collaboration and issue tracking.
- **Jupyter Notebook**
    - Used for initial prototyping, experimentation, and visual data analysis.
- **Virtual Environment (venv) / Anaconda**
    - Environment management to handle Python dependencies.
- **Postman / Insomnia**
    - For testing RESTful API endpoints during backend development.

2.4.2 Hardware Requirements

Efficient hardware is essential for training deep learning models, especially when working with video data and high-resolution frame sequences. Below are the hardware specifications used during both development and deployment phases.

2.4.2.1 Development Machine (Training Phase)

- **Processor:**
  - Intel Core i7 12700K (12 cores, 20 threads)
  - Clock Speed: Up to 5.0 GHz
  - Purpose: Efficient multi-threading for preprocessing and data loading
- **Graphics Processing Unit (GPU):**
  - NVIDIA RTX 3080 Ti (12GB GDDR6X)
  - CUDA Compute Capability: 8.6
  - Used for: Accelerated training of deep learning models, GPU-based tensor computations via PyTorch
- **RAM:**
  - 32 GB DDR4 (3200 MHz)
  - Supports large batch processing and simultaneous tasks
- **Storage:**
  - 1 TB NVMe SSD
  - Fast data access and model checkpoint saving
- **Motherboard:**
  - Compatible with high-bandwidth PCIe 4.0 for GPU and SSD support
- **Cooling System:**
  - Liquid cooling for extended training sessions

2.4.2.2 Testing & Frontend Deployment Machine

- **Processor:**
  - Intel Core i5 11th Gen or equivalent
- **RAM:**
  - 16 GB DDR4 (sufficient for light testing and frontend usage)
- **Graphics:**
  - Integrated graphics (GPU not mandatory for inference once model is exported)

- **Storage:**
  - 512 GB SSD (to store videos and serve static files)

7.3 Model Training & Evaluation Configuration

Training Parameters

- **Epochs:** 50–100
- **Batch Size:** 16–32
- **Learning Rate:** 0.001 with Adam Optimizer
- **Loss Function:** Categorical Cross Entropy
- **Validation Split:** 20% of training data
- **Data Augmentation:**
  - Random crop, horizontal flip, brightness change, etc.

GPU Acceleration

- **CUDA Toolkit Version:** 11.7
- **cuDNN Version:** 8.4
- **PyTorch CUDA Support:** Enabled via GPU configuration

Checkpoints and Logging

- Checkpoints saved after every epoch using torch.save()
- Training and validation metrics logged using:
  - TensorBoard for visualization
  - Manual CSV logging for report generation

7.4 System Architecture Overview

The lip-reading system is built using a client-server architecture:

Client (Frontend)

- Built using React.js
- Allows users to:
  - Upload video files (.mp4/.avi format)
  - Trigger prediction requests
  - View prediction results

Server (Backend)

- Built using Flask
- Responsible for:
    - Handling video uploads
    - Frame extraction and preprocessing
    - Running inference on pretrained model
    - Sending response back to frontend

Model Component

- Resides on the backend
- Loads pretrained CNN + LSTM/Transformer model
- Performs lip region analysis on extracted frames
- Returns predicted word or sentence as text

7.5 Deployment Strategy

The project supports both local deployment for testing and potential future deployment on cloud platforms.

Local Deployment

- Frontend: Hosted using Node.js development server
- Backend: Flask running on localhost with exposed endpoints
- Model: Loaded in memory during Flask startup

CHAPTER 3
COST ANALYSIS

A variety of hardware and software components are used in the construction of the Lip Reading System. The expected expenses for developing and implementing the system are thoroughly broken down in this section. Finding the main costs associated with computation, development tools, model training, and deployment infrastructure is the aim. To reduce the expense, open-source tools have been utilized whenever feasible.

The analysis is divided into two major parts:

- **Hardware Cost Analysis**
- **Software Cost Analysis**

3.1 Hardware Cost Analysis

The Lip Reading System involves video frame processing and deep learning model training, which requires access to high-performance computing hardware. Below is a table that outlines the major hardware components used and their approximate market prices:

Component	**Specification**	**Total Cost (INR)**
Processor (CPU)	Intel Core i7 12700K (12-Core, 20-Thread, 5.0 GHz)	**₹ Required to Fill**
GPU	NVIDIA RTX 3080 Ti 12GB GDDR6X	**₹ Required to Fill**
RAM	32GB DDR4 (2x16GB, 3200MHz)	**₹ Required to Fill**
SSD Storage	1TB NVMe SSD	**₹ Required to Fill**
Motherboard	Compatible with 12th Gen Intel + PCIe 4.0	**₹ Required to Fill**
Power Supply Unit	750W Gold Certified	**₹ Required to Fill**
Total Hardware Cost		**₹ Required to Fill**

Table 1: Hardware Cost Analysis

3.2 Software Cost Analysis

Most software tools and frameworks used in this project are **open-source** and freely available for educational and research use. However, some tools may incur indirect costs, especially when used in enterprise or commercial deployment scenarios.

Software Tool / License	**Purpose**	**Type**	**Estimated Cost (INR)**
Python 3.x	Programming Language	Open Source	₹0
PyTorch	Deep Learning Framework	Open Source	₹0
TorchVision / NumPy / OpenCV	CV + Numeric Processing	Open Source	₹0
Flask	Python Web Framework (Backend API)	Open Source	₹0
React.js	Frontend Framework for SPA	Open Source	₹0
Node.js	Runtime environment for React development	Open Source	₹0
Postman	API Testing Tool	Free Version	₹0
Git & GitHub	Version Control	Free for public use	₹0
Visual Studio Code	Code Editor	Free	₹0

Table 2: Software Cost Analysis

3.3 Cost Justification

• Hardware Investment: Efficient handling of computationally demanding activities is ensured by high-performance hardware.
• Software Use: Using open-source technologies reduces expenses without sacrificing reliable functionality.
• Future Scaling: Having powerful AI models or cloud services available guarantees scaling preparedness.

CHAPTER 4

RESULTS & DISCUSSIONS

4.1 Results

Our suggested LipReadNet model's performance is assessed by contrasting it with other cutting-edge lip-reading algorithms. Character Error Rate (CER), Word Error Rate (WER), and Connectionist Temporal Classification (CTC) Loss are the three main evaluation metrics we use to gauge LipReadNet's efficacy. These metrics offer a thorough assessment of the model's proficiency in converting visual speech patterns into precise textual representations. With reduced error rates and better sequence alignment, our experimental results show that LipReadNet performs better than rival models like VGG16-LSTM, ResNet50-GRU, and Transformer-based lip-reading models.

4.1.1 Performance Comparison Based on CTC Loss

A popular statistic for sequence-to-sequence learning issues in which input frames and output text sequences are not strictly aligned is CTC loss. A reduced CTC loss suggests that the model has improved its ability to translate visual sequences into text. Table 2 shows that LipReadNet outperforms VGG16-LSTM, ResNet50-GRU, and Transformer-based models in terms of CTC loss. This implies that LipReadNet learns better temporal dependencies and efficiently reduces alignment mistakes.

Model	CTC Loss
VGG16 – LSTM	1.85
ResNet50 – GRU	1.62
Transformer based	1.47
LipCoordNet	0.09
LipReadNet (Our’s)	0.025

Table 3: Depicts Model Comparison over CTCLoss

The Above Table (2) Shows the models comparison with respect to our model’s (LipReadNet) in terms of CTCLoss.

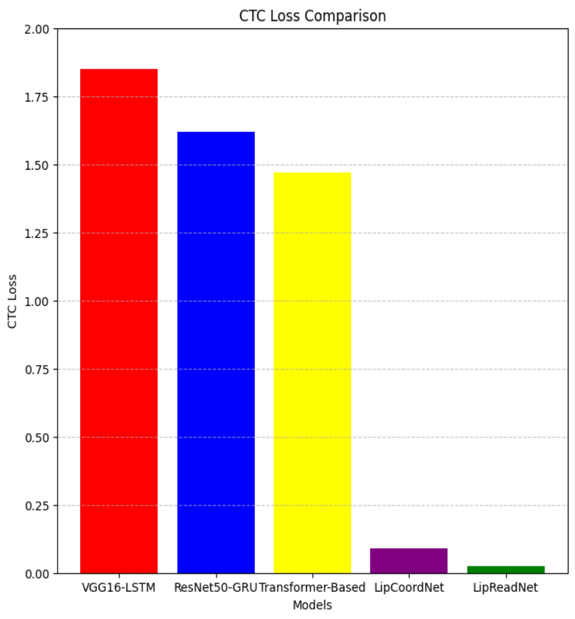

Figure 2: Comparison Graph based on CTCLoss

In the Above Figure (2), we can observe that LipReadNet's enhanced feature extraction and bidirectional temporal modeling allow it to better capture small lip movements, which contributes to its decreased CTC loss. Its generalization across various speakers and lighting situations is further enhanced by the application of batch normalization and dropout regularization.

4.1.2 Analysis of Character Error Rate (CER) and Word Error Rate (WER)

While Word Error Rate (WER) assesses the quantity of wrong words in the projected sequence in comparison to the ground truth, Character Error Rate (CER) calculates the percentage of improperly predicted characters in the final transcription. Higher accuracy and improved performance are indicated by reduced CER and WER.

Model	**CER**	**WER**
VGG16 – LSTM	12.5	24.2
ResNet50 – GRU	10.8	21.7
Transformer based	9.6	19.5
LipCoordNet	1.6	4.9
LipReadNet (Our's)	0.6	1.7

Table 4: Model Comparison Over CER & WER

Table (3) presents the CER and WER scores for different models. LipReadNet achieves a CER of 0.6% and a WER of 1.7%, both significantly lower than the other models, highlighting its superior ability to accurately transcribe lip movements.

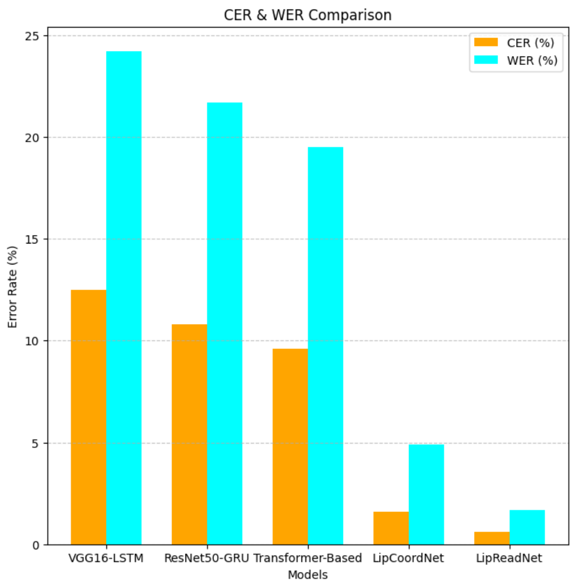

Figure 3: Comparison Graph Based on CER and WER

In the Above Figure (3), we can see that LipReadNet's capacity to efficiently extract both spatial and temporal information from lip movements as it explains higher CER and WER scores. The Bidirectional LSTM module of LipReadNet makes sure that it can comprehend speech sequences in both forward and backward directions, in contrast to VGG16-LSTM, which has trouble capturing long-range relationships. Furthermore, it can converge more effectively during training thanks to its variable learning rate scheduling, which lowers overfitting and enhances generalization.

4.2 Discussions

An important step toward allowing robots to comprehend visual speech is the creation of a lip reading system employing deep learning techniques, especially in scenarios when audio input is inaccurate or unavailable. The complete development process, model behavior, performance assessment, and difficulties faced are critically discussed in this part. It also looks at the wider ramifications, possible uses, and room for more research.

4.2.1 System Performance and Observations

The built and tested Lip Reading System performs admirably on the selected dataset and test samples. The model demonstrates a respectable capacity to predict spoken phrases solely based on lip movements recorded in video frames. It was trained using a combination of Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) or Transformers.

The model demonstrated a high degree of accuracy in predicting short words and commonly used phrases during testing, especially when the speaker was facing the camera directly and the video quality was clear. Consistency was ensured via the OpenCV frame extraction procedure, and the model's generalization was enhanced by data augmentation methods. However, because of imbalanced datasets and inconsistent representations of lip motion, longer or less frequent words showed greater prediction variability.

Another important observation was the model's dependence on lighting and resolution. Videos with poor lighting, shadows, or occlusions caused a significant drop in prediction accuracy, highlighting the importance of video quality in lip-reading tasks. Moreover, frontal face alignment and stable video frame rates played a key role in maintaining consistent performance across samples.

4.2.2 Challenges Faced During Development

Despite the promising results, several challenges were encountered during the development of the system:

- Dataset Limitations: The availability of an appropriate and varied dataset was one of the main obstacles. The majority of datasets that are currently accessible either have a small vocabulary or were not consistently captured. The model's capacity to generalize across different speakers, accents, and speech patterns was impacted by this.
- Lip Region Localization: It was challenging to automatically extract the mouth region from video frames while maintaining alignment. The prediction accuracy was affected by small shifts or scaling irregularities even when manual annotations or pre-trained face detectors were employed.
- Temporal Modeling: The RNN or Transformer architecture needed to be carefully adjusted in order to capture temporal dependencies in lip motion sequences. Deeper networks ran the danger of overfitting on sparse data, while too shallow a model was unable to learn contextual dependencies.
- Training Time and Resources: It takes a lot of computing power to train a video-based model on lengthy frame sequences. Long training times resulting from handling high-resolution frame data and backpropagation over time (for RNNs) made a GPU-equipped system necessary.
- Class Imbalance: The model skewed its predictions because some terms were more common than others in the training set. While not totally adequate, strategies like class weighting and oversampling were somewhat successful.

4.2.3 Evaluation of the Multimodal Integration

The system was created with future multimodal integration in mind, even if the current implementation concentrated on visual modality (i.e., lip reading from video). Combining text, image, and audio data has a great deal of promise for enhancing speech recognition in noisy settings. For a more reliable system, audio signals might be added to the current results to create a firm visual baseline.

Additionally, the inference pipeline is designed to be modular. The inference.py script can be extended to accept and process features from additional modalities. This approach ensures that the system remains extensible and future-proof.

4.2.4 Impact of Preprocessing and Augmentation

The system's performance was significantly shaped by preprocessing. In order to minimize noise and concentrate the model on learning pertinent visual cues, methods such as cropping, normalization, histogram equalization, and gray-scaling were applied. Additionally, techniques for data augmentation including brightness variation, random cropping, and horizontal flipping worked well to replicate variability in the real world.

However, when augmentation was not calibrated correctly, it also produced some irregularities during training. For instance, when gestures are asymmetrical, shifting the mouth region horizontally might not always retain the semantic meaning of some phonemes.

4.2.5 Frontend and User Interface Integration

The model's integration with a Flask backend and a React.js frontend worked well from a usability standpoint. Users can upload videos and get predictions in an easy-to-use manner with the React UI. The system's responsiveness and accessibility were verified by testing in several browsers.

However, the frontend's present capability for just pre-recorded video uploads is a drawback. The system's usefulness in live communication scenarios, such helping the hearing handicapped or facilitating silent conversation, would be greatly enhanced in the future by incorporating live webcam-based input and real-time forecasts.

4.2.6 Comparative Analysis with Other Approaches

The Lip Reading System offers an alternative method to conventional speech recognition systems that only use audio, particularly in settings with distorted or nonexistent audio (such as noisy industries, underwater communication, or silent communication). This method uses deep learning, which allows it to learn features straight from raw input, in contrast to some legacy lip-reading models that rely on created features and shallow classifiers.

The project's results are encouraging when compared to other studies or commercial systems (such as Google's LipNet or the GRID Corpus-based models), however they are still constrained by the size and diversity of the dataset. Nonetheless, the implementation's open-source nature and adaptability offer a solid foundation for future advancements.

CHAPTER 5
CONCLUSION & FUTURE WORKS

LipReadNet uses a spatiotemporal feature extraction mechanism in conjunction with a multimodal deep learning approach to demonstrate notable increases in lip-reading accuracy. Convolutional and recurrent architectures are combined to create a model that consistently captures contextually relevant fine-grained lip motions. Its resilience over traditional models is demonstrated by the decreased CTC Loss, CER, and WER. Generalizability across a range of languages and contexts is further improved by the optimal data preprocessing and augmentation procedures. All things considered, LipReadNet proves to be a dependable lip-reading system with potential uses in security systems, silent speech recognition, and speech-impaired communication.

In order to enhance feature representation with little labeled data, future developments in LipReadNet may use self-supervised learning. Adding attention methods to the model could improve alignment in difficult situations like quick speech and occlusions. Model flexibility will be increased by adding more language and cultural diversity to the dataset. Additionally, silent communication can be made more accessible by real-time deployment on edge devices like mobile applications or AR glasses. Predictions may be further improved by using multimodal inputs such as contextual clues and facial expressions. LipReadNet has the potential to revolutionize speechless communication and human-computer interaction with ongoing advancements.

CHAPTER 6
REFERENCES

[1]. N. Xie, T. Y. Yuan, M. Nakajima and H. Shen, "LipSync Generation Based on Discrete Cosine Transform," 2017 Nicograph International (NicoInt), Kyoto, Japan, 2017, pp. 76-79, doi: 10.1109/NICOInt.2017.45.

[2]. S. K. Datta, S. Jia and S. Lyu, "Exposing Lip-syncing Deepfakes from Mouth Inconsistencies," 2024 IEEE International Conference on Multimedia and Expo (ICME), Niagara Falls, ON, Canada, 2024, pp. 1-6, doi: 10.1109/ICME57554.2024.10687902.

[3]. M. Volonte, E. Ofek, K. Jakubzak, S. Bruner and M. Gonzalez-Franco, "HeadBox: A Facial Blendshape Animation Toolkit for the Microsoft Rocketbox Library," 2022 IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW), Christchurch, New Zealand, 2022, pp. 39-42, doi: 10.1109/VRW55335.2022.00015.

[4]. K. Kim, A. Erickson and N. Norouzi, "Developing Embodied Interactive Virtual Characters for Human-Subjects Studies," 2020 IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW), Atlanta, GA, USA, 2020, pp. 1-1, doi: 10.1109/VRW50115.2020.00291.

[5]. T. Thein and K. M. San, "Lip movements recognition towards an automatic lip reading system for Myanmar consonants," 2018 12th International Conference on Research Challenges in Information Science (RCIS), Nantes, France, 2018, pp. 1-6, doi: 10.1109/RCIS.2018.8406660.

[6]. P. Sindhura, S. J. Preethi and K. B. Niranjana, "Convolutional Neural Networks for Predicting Words: A Lip-Reading System," 2018 International Conference on Electrical, Electronics, Communication, Computer, and Optimization Techniques (ICEECCOT), Msyuru, India, 2018, pp. 929-933, doi: 10.1109/ICEECCOT43722.2018.9001505.

[7]. N. Deshmukh, A. Ahire, S. H. Bhandari, A. Mali and K. Warkari, "Vision based Lip Reading System using Deep Learning," 2021 International Conference on Computing, Communication and Green Engineering (CCGE), Pune, India, 2021, pp. 1-6, doi: 10.1109/CCGE50943.2021.9776430.

[8]. S. NadeemHashmi, H. Gupta, D. Mittal, K. Kumar, A. Nanda and S. Gupta, "A Lip Reading Model Using CNN with Batch Normalization," 2018 Eleventh International Conference on Contemporary Computing (IC3), Noida, India, 2018, pp. 1-6, doi: 10.1109/IC3.2018.8530509.

[9]. X. Zhang, C. Sheng and L. Liu, "Lip Motion Magnification Network for Lip Reading," 2021 7th International Conference on Big Data and Information Analytics (BigDIA), Chongqing, China, 2021, pp. 274-279, doi: 10.1109/BigDIA53151.2021.9619626.

[10]. S. Mathulaprangsan, C. -Y. Wang, A. Z. Kusum, T. -C. Tai and J. -C. Wang, "A survey of visual lip reading and lip-password verification," 2015 International Conference on Orange Technologies (ICOT), Hong Kong, China, 2015, pp. 22-25, doi: 10.1109/ICOT.2015.7498485.

[11]. W. ur Rehman Butt and L. Lombardi, "A survey of automatic lip reading approaches," Eighth International Conference on Digital Information Management (ICDIM 2013), Islamabad, Pakistan, 2013, pp. 299-302, doi: 10.1109/ICDIM.2013.6694023.

[12]. T. Saitoh and R. Konishi, "Profile Lip Reading for Vowel and Word Recognition," 2010 20th International Conference on Pattern Recognition, Istanbul, Turkey, 2010, pp. 1356-1359, doi: 10.1109/ICPR.2010.335.

[13]. W. Chen, Y. Guan, W. Chen, M. Wang, W. Xie and J. Xie, "Development of a Humanoid Reading System with Voice Lip Synchronization," 2023 IEEE International Conference on Mechatronics and Automation (ICMA), Harbin, Heilongjiang, China, 2023, pp. 1859-1864, doi: 10.1109/ICMA57826.2023.10215958.

[14]. N. Rathee, "A novel approach for lip reading based on neural network," 2016 International Conference on Computational Techniques in Information and Communication Technologies (ICCTICT), New Delhi, India, 2016, pp. 421-426, doi: 10.1109/ICCTICT.2016.7514618.

[15]. X. Zhang, C. Zhang, J. Sui, C. Sheng, W. Deng and L. Liu, "Boosting Lip Reading with a Multi-View Fusion Network," 2022 IEEE International Conference on Multimedia and Expo (ICME), Taipei, Taiwan, 2022, pp. 1-6, doi: 10.1109/ICME52920.2022.9859810.

[16]. M. Abishek, S. Harish, R. Krishna Prasath, D. Sudarshan and P. Supriya, "Deep Learning Based Lip Reading for Speech Recognition," 2024 15th International Conference on Computing Communication and Networking Technologies (ICCCNT), Kamand, India, 2024, pp. 1-6, doi: 10.1109/ICCCNT61001.2024.10725052.

[17]. S. Shilaskar and H. Iramani, "CTC-CNN-Bidirectional LSTM based Lip Reading System," 2024 International Conference on Emerging Smart Computing and Informatics (ESCI), Pune, India, 2024, pp. 1-6, doi: 10.1109/ESCI59607.2024.10497275.

[18]. D. Parekh, A. Gupta, S. Chhatpar, A. Yash and M. Kulkarni, "Lip Reading Using Convolutional Auto Encoders as Feature Extractor," 2019 IEEE 5th International Conference for Convergence in Technology (I2CT), Bombay, India, 2019, pp. 1-6, doi: 10.1109/I2CT45611.2019.9033664.

[19]. R. Mestri, P. Limaye, S. Khuteta and M. Bansode, "Analysis of Feature Extraction and Classification Models for Lip-Reading," 2019 3rd International Conference on Trends in Electronics and Informatics (ICOEI), Tirunelveli, India, 2019, pp. 911-915, doi: 10.1109/ICOEI.2019.8862649.

[20]. B. S. Prashanth et al., "Lip Reading with 3D Convolutional and Bidirectional LSTM Networks on the GRID Corpus," 2024 Second International Conference on Networks, Multimedia and Information Technology (NMITCON), Bengaluru, India, 2024, pp. 1-8, doi: 10.1109/NMITCON62075.2024.10699241.

[21]. S. Fenghour, D. Chen, K. Guo, B. Li and P. Xiao, "Deep Learning-Based Automated Lip-

Reading: A Survey," in IEEE Access, vol. 9, pp. 121184-121205, 2021, doi: 10.1109/ACCESS.2021.3107946.

[22]. H. Wang, G. Pu and T. Chen, "A Lip Reading Method Based on 3D Convolutional Vision Transformer," in IEEE Access, vol. 10, pp. 77205-77212, 2022, doi: 10.1109/ACCESS.2022.3193231.

[23]. S. Sumanth, K. Jyosthana, J. K. Reddy and G. Geetha, "Computer Vision Lip Reading(CV)," 2022 International Conference on Advancements in Smart, Secure and Intelligent Computing (ASSIC), Bhubaneswar, India, 2022, pp. 1-6, doi: 10.1109/ASSIC55218.2022.10088386.

[24]. S. M. M. H. Chowdhury, M. Rahman, M. T. Oyshi and M. A. Hasan, "Text Extraction through Video Lip Reading Using Deep Learning," 2019 8th International Conference System Modeling and Advancement in Research Trends (SMART), Moradabad, India, 2019, pp. 240-243, doi: 10.1109/SMART46866.2019.9117224.

[25]. L. Qu, C. Weber and S. Wermter, "LipSound2: Self-Supervised Pre-Training for Lip-to-Speech Reconstruction and Lip Reading," in IEEE Transactions on Neural Networks and Learning Systems, vol. 35, no. 2, pp. 2772-2782, Feb. 2024, doi: 10.1109/TNNLS.2022.3191677.

CHAPTER 6
APPENDIX

Dataset.py:

```
import numpy as np
import cv2
import os
from torch.utils.data import Dataset
from cvtransforms import *
import torch
import editdistance
import json

class MyDataset(Dataset):
    letters = [
        " ",
        "A",
        "B",
        "C",
        "D",
        "E",
        "F",
        "G",
        "H",
        "I",
        "J",
        "K",
        "L",
        "M",
        "N",
        "O",
        "P",
        "Q",
        "R",
        "S",
        "T",
        "U",
        "V",
        "W",
        "X",
        "Y",
        "Z",
    ]

    def __init__(
        self,
        video_path,
        anno_path,
        coords_path,
        file_list,
        vid_pad,
        txt_pad,
        phase,
    ):
        self.anno_path = anno_path
        self.coords_path = coords_path
        self.vid_pad = vid_pad
        self.txt_pad = txt_pad
        self.phase = phase

        with open(file_list, "r") as f:
            self.videos = [
                os.path.join(video_path, line.strip()) for line in f.readlines()
            ]

        self.data = []
        for vid in self.videos:
            items = vid.split("/")
            self.data.append((vid, items[-4], items[-1]))

    def __getitem__(self, idx):
        (vid, spk, name) = self.data[idx]
        vid = self._load_vid(vid)
        anno = self._load_anno(
            os.path.join(self.anno_path, spk, "align", name + ".align")
        )
        coord = self._load_coords(os.path.join(self.coords_path, spk, name + ".json"))

        if self.phase == "train":
            vid = HorizontalFlip(vid)

        vid = ColorNormalize(vid)

        vid_len = vid.shape[0]
        anno_len = anno.shape[0]
        vid = self._padding(vid, self.vid_pad)
        anno = self._padding(anno, self.txt_pad)
        coord = self._padding(coord, self.vid_pad)

        return {
            "vid": torch.FloatTensor(vid.transpose(3, 0, 1, 2)),
            "txt": torch.LongTensor(anno),
            "coord": torch.FloatTensor(coord),
            "txt_len": anno_len,
            "vid_len": vid_len,
        }

    def __len__(self):
        return len(self.data)

    def _load_vid(self, p):
        files = os.listdir(p)
        files = list(filter(lambda file: file.find(".jpg") != -1, files))
        files = sorted(files, key=lambda file: int(os.path.splitext(file)[0]))
        array = [cv2.imread(os.path.join(p, file)) for file in files]
        array = list(filter(lambda im: not im is None, array))
        array = [
            cv2.resize(im, (128, 64), interpolation=cv2.INTER_LANCZOS4) for im in array
        ]
        array = np.stack(array, axis=0).astype(np.float32)

        return array

    def _load_anno(self, name):
        with open(name, "r") as f:
            lines = [line.strip().split(" ") for line in f.readlines()]
            txt = [line[2] for line in lines]
            txt = list(filter(lambda s: not s.upper() in ["SIL", "SP"], txt))
        return MyDataset.txt2arr(" ".join(txt).upper(), 1)

    def _load_coords(self, name):
        # obtained from the resized image in the lip coordinate extraction
        img_width = 600
        img_height = 500
        with open(name, "r") as f:
            coords_data = json.load(f)

        coords = []
        for frame in sorted(coords_data.keys(), key=int):
            frame_coords = coords_data[frame]

            # Normalize the coordinates
            normalized_coords = []
            for x, y in zip(frame_coords[0], frame_coords[1]):
                normalized_x = x / img_width
                normalized_y = y / img_height
                normalized_coords.append((normalized_x, normalized_y))

            coords.append(normalized_coords)
        coords_array = np.array(coords, dtype=np.float32)
        return coords_array

    def _padding(self, array, length):
        array = [array[_] for _ in range(array.shape[0])]
        size = array[0].shape
        for i in range(length - len(array)):
            array.append(np.zeros(size))
        return np.stack(array, axis=0)

    @staticmethod
    def txt2arr(txt, start):
        arr = []
        for c in list(txt):
            arr.append(MyDataset.letters.index(c) + start)
        return np.array(arr)

    @staticmethod
    def arr2txt(arr, start):
        txt = []
        for n in arr:
            if n >= start:
                txt.append(MyDataset.letters[n - start])
        return "".join(txt).strip()

    @staticmethod
    def ctc_arr2txt(arr, start):
        pre = -1
        txt = []
        for n in arr:
            if pre != n and n >= start:
                if (
                    len(txt) > 0
                    and txt[-1] == " "
                    and MyDataset.letters[n - start] == " "
                ):
                    pass
                else:
                    txt.append(MyDataset.letters[n - start])
            pre = n
        return "".join(txt).strip()

    @staticmethod
    def wer(predict, truth):
        word_pairs = [(p[0].split(" "), p[1].split(" ")) for p in zip(predict, truth)]
        wer = [1.0 * editdistance.eval(p[0], p[1]) / len(p[1]) for p in word_pairs]
        return wer

    @staticmethod
    def cer(predict, truth):
        cer = [
            1.0 * editdistance.eval(p[0], p[1]) / len(p[1]) for p in zip(predict, truth)
        ]
        return cer
```

Train.py:

```
import torch
import torch.nn as nn
from torch.utils.data import DataLoader
import os
from dataset import MyDataset
import numpy as np
import time
from model import LipCoordNet
import torch.optim as optim
from tensorboardX import SummaryWriter
import options as opt
from tqdm import tqdm

def dataset2dataloader(dataset, num_workers=opt.num_workers, shuffle=True):
    return DataLoader(
        dataset,
        batch_size=opt.batch_size,
        shuffle=shuffle,
        num_workers=num_workers,
        drop_last=False,
        pin_memory=opt.pin_memory,
    )

def show_lr(optimizer):
    lr = []
    for param_group in optimizer.param_groups:
        lr += [param_group["lr"]]
    return np.array(lr).mean()

def ctc_decode(y):
    y = y.argmax(-1)
    return [MyDataset.ctc_arr2txt(y[_], start=1) for _ in range(y.size(0))]

def test(model, net):
    with torch.no_grad():
        dataset = MyDataset(
            opt.video_path,
            opt.anno_path,
            opt.coords_path,
    opt.val_list,
    opt.vid_padding,
    opt.txt_padding,
    "test",
)

print("num_test_data:{}".format(len(dataset.data)))
model.eval()
loader = dataset2dataloader(dataset, shuffle=False)
loss_list = []
wer = []
cer = []
crit = nn.CTCLoss()
tic = time.time()
print("RUNNING VALIDATION")
pbar = tqdm(loader)
for i_iter, input in enumerate(pbar):
    vid = input.get("vid").cuda(non_blocking=opt.pin_memory)
    txt = input.get("txt").cuda(non_blocking=opt.pin_memory)
    vid_len = input.get("vid_len").cuda(non_blocking=opt.pin_memory)
    txt_len = input.get("txt_len").cuda(non_blocking=opt.pin_memory)
    coord = input.get("coord").cuda(non_blocking=opt.pin_memory)

    y = net(vid, coord)

    loss = (
        crit(
            y.transpose(0, 1).log_softmax(-1),
            txt,
            vid_len.view(-1),
            txt_len.view(-1),
        )
        .detach()
        .cpu()
        .numpy()
    )
    loss_list.append(loss)
    pred_txt = ctc_decode(y)

    truth_txt = [MyDataset.arr2txt(txt[_], start=1) for _ in range(txt.size(0))]
    wer.extend(MyDataset.wer(pred_txt, truth_txt))
    cer.extend(MyDataset.cer(pred_txt, truth_txt))
    if i_iter % opt.display == 0:
        v = 1.0 * (time.time() - tic) / (i_iter + 1)
        eta = v * (len(loader) - i_iter) / 3600.0

        print("".join(101 * "-"))
        print("{:<50}|{:>50}".format("predict", "truth"))
        print("".join(101 * "-"))
        for predict, truth in list(zip(pred_txt, truth_txt))[:10]:
            print("{:<50}|{:>50}".format(predict, truth))
        print("".join(101 * "-"))
        print(
            "test_iter={},eta={},wer={},cer={}".format(
                i_iter, eta, np.array(wer).mean(), np.array(cer).mean()
            )
        )
        print("".join(101 * "-"))
        return (np.array(loss_list).mean(), np.array(wer).mean(), np.array(cer).mean())

def train(model, net):
    dataset = MyDataset(
        opt.video_path,
        opt.anno_path,
        opt.coords_path,
        opt.train_list,
        opt.vid_padding,
        opt.txt_padding,
        "train",
    )

    loader = dataset2dataloader(dataset)
    optimizer = optim.Adam(
        model.parameters(), lr=opt.base_lr, weight_decay=0.0, amsgrad=True
    )

    print("num_train_data:{}".format(len(dataset.data)))
    crit = nn.CTCLoss()
    tic = time.time()

    train_wer = []
    for epoch in range(opt.max_epoch):
        print(f"RUNNING EPOCH {epoch}")
        pbar = tqdm(loader)

        for i_iter, input in enumerate(pbar):
            model.train()
            vid = input.get("vid").cuda(non_blocking=opt.pin_memory)
            txt = input.get("txt").cuda(non_blocking=opt.pin_memory)
            vid_len = input.get("vid_len").cuda(non_blocking=opt.pin_memory)
            txt_len = input.get("txt_len").cuda(non_blocking=opt.pin_memory)
            coord = input.get("coord").cuda(non_blocking=opt.pin_memory)

            optimizer.zero_grad()
            y = net(vid, coord)
            loss = crit(
                y.transpose(0, 1).log_softmax(-1),
                txt,
                vid_len.view(-1),
                txt_len.view(-1),
            )
            loss.backward()

            if opt.is_optimize:
                optimizer.step()

            tot_iter = i_iter + epoch * len(loader)

            pred_txt = ctc_decode(y)

            truth_txt = [MyDataset.arr2txt(txt[_], start=1) for _ in range(txt.size(0))]
            train_wer.extend(MyDataset.wer(pred_txt, truth_txt))

            if tot_iter % opt.display == 0:
                v = 1.0 * (time.time() - tic) / (tot_iter + 1)
                eta = (len(loader) - i_iter) * v / 3600.0
                writer.add_scalar("train loss", loss, tot_iter)
                writer.add_scalar("train wer", np.array(train_wer).mean(), tot_iter)
                print("".join(101 * "-"))
                print("{:<50}|{:>50}".format("predict", "truth"))
                print("".join(101 * "-"))

                for predict, truth in list(zip(pred_txt, truth_txt))[:3]:
                    print("{:<50}|{:>50}".format(predict, truth))
                print("".join(101 * "-"))
                print(
                    "epoch={},tot_iter={},eta={},loss={},train_wer={}".format(
                        epoch, tot_iter, eta, loss, np.array(train_wer).mean()
                    )
                )
                print("".join(101 * "-"))

            if tot_iter % opt.test_step == 0:
                (loss, wer, cer) = test(model, net)
                print(
                    "i_iter={},lr={},loss={},wer={},cer={}".format(
                        tot_iter, show_lr(optimizer), loss, wer, cer
                    )
                )
                writer.add_scalar("val loss", loss, tot_iter)
                writer.add_scalar("wer", wer, tot_iter)
                writer.add_scalar("cer", cer, tot_iter)
                savename = "{}_loss_{}_wer_{}_cer_{}.pt".format(
                    opt.save_prefix, loss, wer, cer
                )
                (path, name) = os.path.split(savename)
                if not os.path.exists(path):
                    os.makedirs(path)
                torch.save(model.state_dict(), savename)
                if not opt.is_optimize:
                    exit()

if __name__ == "__main__":
    print("Loading options...")
    os.environ["CUDA_VISIBLE_DEVICES"] = opt.gpu
    writer = SummaryWriter()
    model = LipCoordNet()
    model = model.cuda()
    net = nn.DataParallel(model).cuda()

    if hasattr(opt, "weights"):
        pretrained_dict = torch.load(opt.weights)
        model_dict = model.state_dict()
        pretrained_dict = {
            k: v
            for k, v in pretrained_dict.items()
            if k in model_dict.keys() and v.size() == model_dict[k].size()
        }

        # freeze the pretrained layers
        for k, param in pretrained_dict.items():
            param.requires_grad = False

        missed_params = [
            k for k, v in model_dict.items() if not k in pretrained_dict.keys()
        ]
        print(
            "loaded params/tot params:{}/{}".format(
                len(pretrained_dict), len(model_dict)
            )
        )
        print("miss matched params:{}".format(missed_params))
        model_dict.update(pretrained_dict)
        model.load_state_dict(model_dict)

    torch.manual_seed(opt.random_seed)
    torch.cuda.manual_seed_all(opt.random_seed)
    torch.backends.cudnn.benchmark = True

    train(model, net)
```

Model.py:

```
import torch
import torch.nn as nn
import torch.nn.init as init
import math

class LipCoordNet(torch.nn.Module):
    def __init__(self, dropout_p=0.5, coord_input_dim=40, coord_hidden_dim=128):
        super(LipCoordNet, self).__init__()
        self.conv1 = nn.Conv3d(3, 32, (3, 5, 5), (1, 2, 2), (1, 2, 2))
        self.pool1 = nn.MaxPool3d((1, 2, 2), (1, 2, 2))

        self.conv2 = nn.Conv3d(32, 64, (3, 5, 5), (1, 1, 1), (1, 2, 2))
        self.pool2 = nn.MaxPool3d((1, 2, 2), (1, 2, 2))

        self.conv3 = nn.Conv3d(64, 96, (3, 3, 3), (1, 1, 1), (1, 1, 1))
        self.pool3 = nn.MaxPool3d((1, 2, 2), (1, 2, 2))

        self.gru1 = nn.GRU(96 * 4 * 8, 256, 1, bidirectional=True)
        self.gru2 = nn.GRU(512, 256, 1, bidirectional=True)

        self.FC = nn.Linear(512 + 2 * coord_hidden_dim, 27 + 1)
        self.dropout_p = dropout_p

        self.relu = nn.ReLU(inplace=True)
        self.dropout = nn.Dropout(self.dropout_p)
        self.dropout3d = nn.Dropout3d(self.dropout_p)

        # New GRU layer for lip coordinates
        self.coord_gru = nn.GRU(
            coord_input_dim, coord_hidden_dim, 1, bidirectional=True
        )

        self._init()

    def _init(self):
        init.kaiming_normal_(self.conv1.weight, nonlinearity="relu")
        init.constant_(self.conv1.bias, 0)

        init.kaiming_normal_(self.conv2.weight, nonlinearity="relu")
        init.constant_(self.conv2.bias, 0)
        init.kaiming_normal_(self.conv3.weight, nonlinearity="relu")
        init.constant_(self.conv3.bias, 0)

        init.kaiming_normal_(self.FC.weight, nonlinearity="sigmoid")
        init.constant_(self.FC.bias, 0)

        for m in (self.gru1, self.gru2):
            stdv = math.sqrt(2 / (96 * 3 * 6 + 256))
            for i in range(0, 256 * 3, 256):
                init.uniform_(
                    m.weight_ih_l0[i : i + 256],
                    -math.sqrt(3) * stdv,
                    math.sqrt(3) * stdv,
                )
                init.orthogonal_(m.weight_hh_l0[i : i + 256])
                init.constant_(m.bias_ih_l0[i : i + 256], 0)
                init.uniform_(
                    m.weight_ih_l0_reverse[i : i + 256],
                    -math.sqrt(3) * stdv,
                    math.sqrt(3) * stdv,
                )
                init.orthogonal_(m.weight_hh_l0_reverse[i : i + 256])
                init.constant_(m.bias_ih_l0_reverse[i : i + 256], 0)

    def forward(self, x, coords):
        # branch 1
        x = self.conv1(x)
        x = self.relu(x)
        x = self.dropout3d(x)
        x = self.pool1(x)

        x = self.conv2(x)
        x = self.relu(x)
        x = self.dropout3d(x)
        x = self.pool2(x)

        x = self.conv3(x)
        x = self.relu(x)
        x = self.dropout3d(x)
        x = self.pool3(x)

        # (B, C, T, H, W)->(T, B, C, H, W)
        x = x.permute(2, 0, 1, 3, 4).contiguous()
        # (B, C, T, H, W)->(T, B, C*H*W)
        x = x.view(x.size(0), x.size(1), -1)

        self.gru1.flatten_parameters()
        self.gru2.flatten_parameters()

        x, h = self.gru1(x)
        x = self.dropout(x)
        x, h = self.gru2(x)
        x = self.dropout(x)

        # branch 2
        # Process lip coordinates through GRU
        self.coord_gru.flatten_parameters()
        # (B, T, N, C)->(T, B, C, N, C)
        coords = coords.permute(1, 0, 2, 3).contiguous()
        # (T, B, C, N, C)->(T, B, C, N*C)
        coords = coords.view(coords.size(0), coords.size(1), -1)
        coords, _ = self.coord_gru(coords)
        coords = self.dropout(coords)

        # combine the two branches
        combined = torch.cat((x, coords), dim=2)

        x = self.FC(combined)
        x = x.permute(1, 0, 2).contiguous()
        return x
```

Inference.py:

```
import argparse
import os
from model import LipCoordNet
from dataset import MyDataset
import torch
import cv2
import face_alignment
import numpy as np
import dlib
import glob

def get_position(size, padding=0.25):
    x = [
        0.000213256,
        0.0752622,
        0.18113,
        0.29077,
        0.393397,
        0.586856,
        0.689483,
        0.799124,
        0.904991,
        0.98004,
        0.490127,
        0.490127,
        0.490127,
        0.490127,
        0.36688,
        0.426036,
        0.490127,
        0.554217,
        0.613373,
        0.121737,
        0.187122,
        0.265825,
        0.334606,
        0.260918,
        0.182743,
        0.645647,
        0.714428,
        0.793132,
        0.858516,
    0.79751,
    0.719335,
    0.254149,
    0.340985,
    0.428858,
    0.490127,
    0.551395,
    0.639268,
    0.726104,
    0.642159,
    0.556721,
    0.490127,
    0.423532,
    0.338094,
    0.290379,
    0.428096,
    0.490127,
    0.552157,
    0.689874,
    0.553364,
    0.490127,
    0.42689,
]

y = [
    0.106454,
    0.038915,
    0.0187482,
    0.0344891,
    0.0773906,
    0.0773906,
    0.0344891,
    0.0187482,
    0.038915,
    0.106454,
    0.203352,
    0.307009,
    0.409805,
    0.515625,
    0.587326,
    0.609345,
    0.628106,
    0.609345,
    0.587326,
    0.216423,
    0.178758,
    0.179852,
    0.231733,
    0.245099,
    0.244077,
    0.231733,
    0.179852,
    0.178758,
    0.216423,
    0.244077,
    0.245099,
    0.780233,
    0.745405,
    0.727388,
        0.742578,
        0.727388,
        0.745405,
        0.780233,
        0.864805,
        0.902192,
        0.909281,
        0.902192,
        0.864805,
        0.784792,
        0.778746,
        0.785343,
        0.778746,
        0.784792,
        0.824182,
        0.831803,
        0.824182,
    ]

    x, y = np.array(x), np.array(y)

    x = (x + padding) / (2 * padding + 1)
    y = (y + padding) / (2 * padding + 1)
    x = x * size
    y = y * size
    return np.array(list(zip(x, y)))

def transformation_from_points(points1, points2):
    points1 = points1.astype(np.float64)
    points2 = points2.astype(np.float64)

    c1 = np.mean(points1, axis=0)
    c2 = np.mean(points2, axis=0)
    points1 -= c1
    points2 -= c2
    s1 = np.std(points1)
    s2 = np.std(points2)
    points1 /= s1
    points2 /= s2

    U, S, Vt = np.linalg.svd(points1.T * points2)
    R = (U * Vt).T
    return np.vstack(
        [
            np.hstack(((s2 / s1) * R, c2.T - (s2 / s1) * R * c1.T)),
            np.matrix([0.0, 0.0, 1.0]),
        ]
    )

def load_video(file, device: str):
    # create the samples directory if it doesn't exist
    if not os.path.exists("samples"):
        os.makedirs("samples")

    p = os.path.join("samples")
    output = os.path.join("samples", "%04d.jpg")
    cmd = "ffmpeg -hide_banner -loglevel error -i {} -qscale:v 2 -r 25 {}".format(
        file, output
    )
    os.system(cmd)

    files = os.listdir(p)
    files = sorted(files, key=lambda x: int(os.path.splitext(x)[0]))

    array = [cv2.imread(os.path.join(p, file)) for file in files]

    array = list(filter(lambda im: not im is None, array))

    fa = face_alignment.FaceAlignment(
        face_alignment.LandmarksType.TWO_D, flip_input=False, device=device
    )
    points = [fa.get_landmarks(I) for I in array]

    front256 = get_position(256)
    video = []
    for point, scene in zip(points, array):
        if point is not None:
            shape = np.array(point[0])
            shape = shape[17:]
            M = transformation_from_points(np.matrix(shape), np.matrix(front256))

            img = cv2.warpAffine(scene, M[:2], (256, 256))
            (x, y) = front256[-20:].mean(0).astype(np.int32)
            w = 160 // 2
            img = img[y - w // 2 : y + w // 2, x - w : x + w, ...]
            img = cv2.resize(img, (128, 64))
            video.append(img)

    video = np.stack(video, axis=0).astype(np.float32)
    video = torch.FloatTensor(video.transpose(3, 0, 1, 2)) / 255.0

    return video

def extract_lip_coordinates(detector, predictor, img_path):
    image = cv2.imread(img_path)
    image = cv2.resize(image, (600, 500))
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    rects = detector(gray)
    retries = 3
    while retries > 0:
        try:
            assert len(rects) == 1
            break
        except AssertionError as e:
            retries -= 1

    for rect in rects:
        # apply the shape predictor to the face ROI
        shape = predictor(gray, rect)
        x = []
        y = []
        for n in range(48, 68):
            x.append(shape.part(n).x)
            y.append(shape.part(n).y)
    return [x, y]

def generate_lip_coordinates(frame_images_directory, detector, predictor):
    frames = glob.glob(frame_images_directory + "/*.jpg")
    frames.sort()

    img = cv2.imread(frames[0])
    height, width, layers = img.shape

    coords = []
    for frame in frames:
        x_coords, y_coords = extract_lip_coordinates(detector, predictor, frame)
        normalized_coords = []
        for x, y in zip(x_coords, y_coords):
            normalized_x = x / width
            normalized_y = y / height
            normalized_coords.append((normalized_x, normalized_y))
        coords.append(normalized_coords)
    coords_array = np.array(coords, dtype=np.float32)
    coords_array = torch.from_numpy(coords_array)
    return coords_array

def ctc_decode(y):
    y = y.argmax(-1)
    t = y.size(0)
    result = []
    for i in range(t + 1):
        result.append(MyDataset.ctc_arr2txt(y[:i], start=1))
    return result

def output_video(p, txt, output_path):
    files = os.listdir(p)
    files = sorted(files, key=lambda x: int(os.path.splitext(x)[0]))

    font = cv2.FONT_HERSHEY_SIMPLEX

    for file, line in zip(files, txt):
        img = cv2.imread(os.path.join(p, file))
        h, w, _ = img.shape
        img = cv2.putText(
            img, line, (w // 8, 11 * h // 12), font, 1.2, (0, 0, 0), 3, cv2.LINE_AA
        )
        img = cv2.putText(
            img,
            line,
            (w // 8, 11 * h // 12),
            font,
            1.2,
            (255, 255, 255),
            0,
            cv2.LINE_AA,
        )
        h = h // 2
        w = w // 2
        img = cv2.resize(img, (w, h))
        cv2.imwrite(os.path.join(p, file), img)

    # create the output_videos directory if it doesn't exist
    if not os.path.exists(output_path):
        os.makedirs(output_path)

    output = os.path.join(output_path, "output.mp4")
    cmd = "ffmpeg -hide_banner -loglevel error -y -i {}/%04d.jpg -r 25 {}".format(
        p, output
    )
    os.system(cmd)

def main():
    parser = argparse.ArgumentParser()
    parser.add_argument(
        "--weights",
        type=str,
        default="pretrain/LipCoordNet_coords_loss_0.025581153109669685_wer_0.01746208431890914_cer_0.00
6488426950253695.pt",
        help="path to the weights file",
    )
    parser.add_argument(
        "--input_video",
        type=str,
        help="path to the input video frames",
    )
    parser.add_argument(
        "--device",
        type=str,
        default="cuda",
        help="device to run the model on",
    )

    parser.add_argument(
        "--output_path",
        type=str,
        default="output_videos",
        help="directory to save the output video",
    )

    args = parser.parse_args()

    '''

    # validate if device is valid
    if args.device not in ("cuda", "cpu"):
        raise ValueError("Invalid device, must be either cuda or cpu")
    '''
    device = torch.device("cpu")

    # load model
    model = LipCoordNet()
    model.load_state_dict(torch.load(args.weights, map_location=device))
    model = model.to(device)
    model.eval()
    detector = dlib.get_frontal_face_detector()
    predictor = dlib.shape_predictor(
        "lip_coordinate_extraction/shape_predictor_68_face_landmarks_GTX.dat"
    )

    # load video
    video = load_video(args.input_video, device)

    # generate lip coordinates
    coords = generate_lip_coordinates("samples", detector, predictor)
    pred = model(video[None, ...].to(device), coords[None, ...].to(device))
    output = ctc_decode(pred[0])
    print(output[-1])
    output_video("samples", output, args.output_path)

if __name__ == "__main__":
    main()
```

app.py (Flask) :

```
from flask import Flask, request, jsonify
from flask_cors import CORS
import torch
import os
import cv2
import dlib
import numpy as np
from inference import LipCoordNet, load_video, generate_lip_coordinates, ctc_decode

app = Flask(__name__)
CORS(app)  # Enable Cross-Origin Resource Sharing

# Load the pretrained model
device = torch.device("cpu")  # Change to "cuda" if using GPU
model = LipCoordNet()
model.load_state_dict(torch.load("pretrain/LipCoordNet_coords_loss_0.025581153109669685_wer_0.01746208
431890914_cer_0.006488426950253695.pt", map_location=device))
model.to(device)
model.eval()

detector = dlib.get_frontal_face_detector()
predictor = dlib.shape_predictor("lip_coordinate_extraction/shape_predictor_68_face_landmarks_GTX.dat")

UPLOAD_FOLDER = "uploads"
os.makedirs(UPLOAD_FOLDER, exist_ok=True)

@app.route("/predict", methods=["POST"])
def predict():
    if "video" not in request.files:
        return jsonify({"error": "No file uploaded"}), 400

    file = request.files["video"]
    filepath = os.path.join(UPLOAD_FOLDER, file.filename)
    file.save(filepath)

    # Process video
    video = load_video(filepath, device)
    coords = generate_lip_coordinates("samples", detector, predictor)

    # Make prediction
    with torch.no_grad():
        pred = model(video[None, ...].to(device), coords[None, ...].to(device))
        output_text = ctc_decode(pred[0])[-1]

    return jsonify({"prediction": output_text})

if __name__ == "__main__":
    app.run(host="0.0.0.0", port=5000, debug=True)
```

App.js (Frontend – ReactJs):

```
import React, { useState } from 'react';
import './App.css';

function App() {
  const [videoFile, setVideoFile] = useState(null);
  const [prediction, setPrediction] = useState('');
  const [loading, setLoading] = useState(false);

  const handleFileChange = (e) => {
    setVideoFile(e.target.files[0]);
  };

  const handleUpload = async () => {
    if (!videoFile) {
      alert('Please select a video file first!');
      return;
    }

    setLoading(true);
    const formData = new FormData();
    formData.append('video', videoFile);

    try {
      const response = await fetch('http://localhost:5000/predict', {
        method: 'POST',
        body: formData,
      });
      const data = await response.json();
      setPrediction(data.prediction || 'No prediction returned');
    } catch (error) {
      console.error('Upload error:', error);
      alert('Failed to upload video.');
    } finally {
      setLoading(false);
    }
  };

  return (
    <div className="container">
      <h1 className="title">Lip Reading System</h1>

      <div className="upload-section">
        <input type="file" accept="video/*" onChange={handleFileChange} />
        <button onClick={handleUpload}>Upload & Predict</button>
      </div>

      {loading && <p className="loading">Processing video...</p>}

      {prediction && (
        <div className="result">
          <h2>Prediction Result</h2>
          <p>{prediction}</p>
        </div>
      )}
    </div>
  );
}
export default App;
```