Lip Reading Using Deep Learning

Contributors:

Samarth Gayakhe

Ashish Singh

Isha Narwaria

G Nishika

Content

1. Introduction

1.1 Project Overview

The project involves developing an end-to-end machine learning solution to detect spoken words from video sequences focusing on analyzing lip movements. The proposed solution integrates deep learning algorithms such as LSTM and Neural Networks to accurately predict word outputs based on lip movements captured in the video data.

1.2 Purpose

The purpose of this project is to leverage machine learning, specifically deep learning techniques, for lip reading to achieve several significant outcomes:

1. **Improved Speech Recognition:** Lip reading acts as a complementary component to audio-based speech recognition systems. It enhances accuracy and robustness, especially in noisy environments or scenarios where audio signals are unclear.
2. **Elimination of Audio Data Dependency:** Unlike traditional speech recognition models that rely on transcribed audio data for training, an end-to-end lip reading system can be trained exclusively on video data. This eliminates the need for expensive and time-consuming transcribed audio data acquisition.
3. **Multi-Modal Applications:** Integrating lip reading with audio-based systems creates multi-modal applications. For instance, in video conferencing, this fusion can significantly improve real-time communication by providing more accurate transcriptions.
4. **Accessibility for Hearing-Impaired Individuals:** A precise lip reading system serves as a crucial communication tool for individuals with hearing impairments. It enhances their ability to understand spoken language and actively participate in conversations, promoting inclusivity and accessibility.

2. LITERATURE SURVEY

2.1 Existing Problem:

In the realm of lip reading using deep learning, several challenges persist despite technological advancements. Some of the existing problems include:

Limited Datasets: Availability of comprehensive and diverse datasets for training deep learning models remains a challenge. Insufficient data could lead to model biases and limited generalization to real-world scenarios.

Complexity of Lip Movements:The intricate and variable nature of lip movements presents a challenge for accurate interpretation. Variations in pronunciation, accents, and contextual lip shapes add complexity to the recognition process.

Real-Time Processing: Achieving real-time lip reading capabilities is a challenge due to the computational demands of deep learning models. Implementing efficient algorithms for swift video analysis without compromising accuracy is an ongoing concern.

2.2 References:

Yannis M. Assael , Brendan Shillingford , Shimon Whiteson & Nando de Freitas(2017). LIPNET: END-TO-END SENTENCE-LEVEL LIPREADING

Petridis, S., Pantic, M. (2018). End-to-End Visual Speech Recognition with LSTMs. IEEE Transactions on Multimedia, 20(6), 1386-1396.

Chung, J. S., Zisserman, A. (2016). Lip Reading Sentences in the Wild. Computer Vision and Pattern Recognition.

Wand, M., Gurban, M., Wu, Y. (2017). Lipreading with Long Short-Term Memory. European Conference on Computer Vision.

2.3 Problem Statement Definition:

The problem statement for this project revolves around creating an accurate and efficient end-to-end machine learning system for lip reading using deep learning techniques. The key components of the problem statement include:

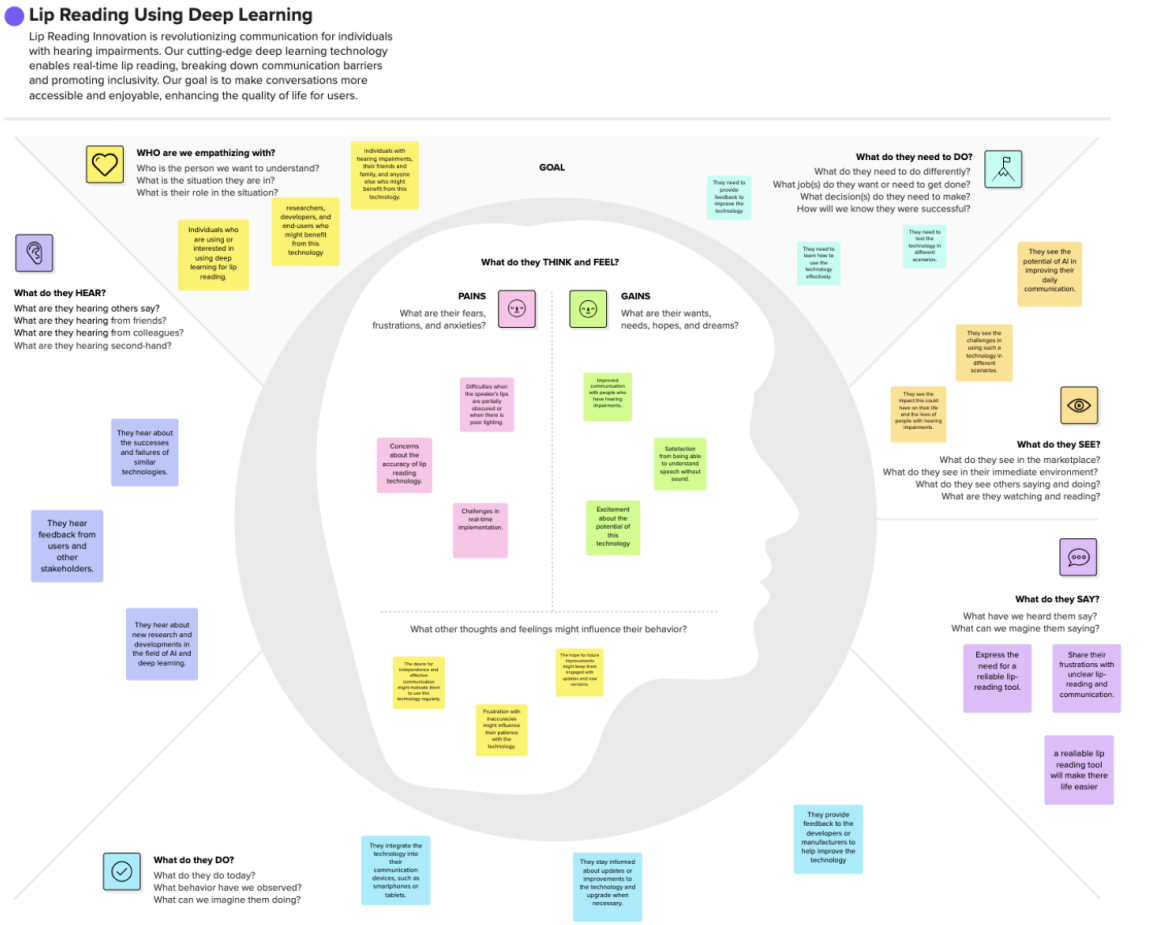
Data Availability: Gathering and utilizing comprehensive datasets to train deep learning models for accurate lip reading.

Model Robustness: Developing models capable of handling variations in lip movements, accents, and pronunciation for improved accuracy across diverse contexts.

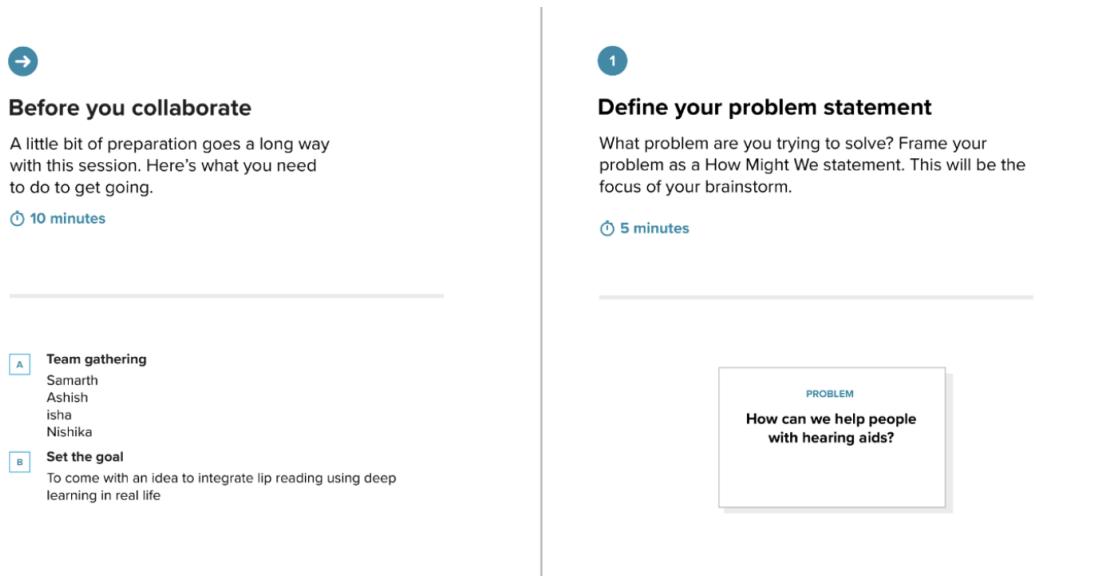
Real-Time Processing: Designing algorithms and architectures that balance accuracy and speed to enable real-time lip reading capabilities without compromising on performance.

4. IDEATION & PROPOSED SOLUTION

4.1 Empathy Map Canvas



4.2 Ideation and Brainstorming



2

Brainstorm

Write down any ideas that come to mind that address your problem statement.

10 minutes

TIP

You can select a sticky note and hit the pencil [switch to sketch] icon to start drawing!

Samarth

- Lip Reading Assistant
- Lip Reading in Noisy Environments

Ashish

- Low-Light and Infrared Lip Reading
- Lip Reading Web App

Isha

- Smart Assistants for the Deaf
- Lip-Reading Training Platform

Nishika

- Multilingual Lip Reading
- Improving Hearing Aids

3

Group ideas

Take turns sharing your ideas while clustering similar or related notes as you go. Once all sticky notes have been grouped, give each cluster a sentence-like label. If a cluster is bigger than six sticky notes, try and see if you and break it up into smaller sub-groups.

20 minutes

TIP

Add customizable tags to sticky notes to make it easier to find, browse, organize, and categorize important ideas as themes within your mural.

Lip Reading Technologies

- Lip Reading Assistant
- Lip Reading in Noisy Environments
- Low-Light and Infrared Lip Reading
- Lip Reading Web App

Lip Reading Education and Training

- Multilingual Lip Reading
- Lip-Reading Training Platform

Accessibility Solutions for the Deaf

- Smart Assistants for the Deaf
- Improving Hearing Aids

4

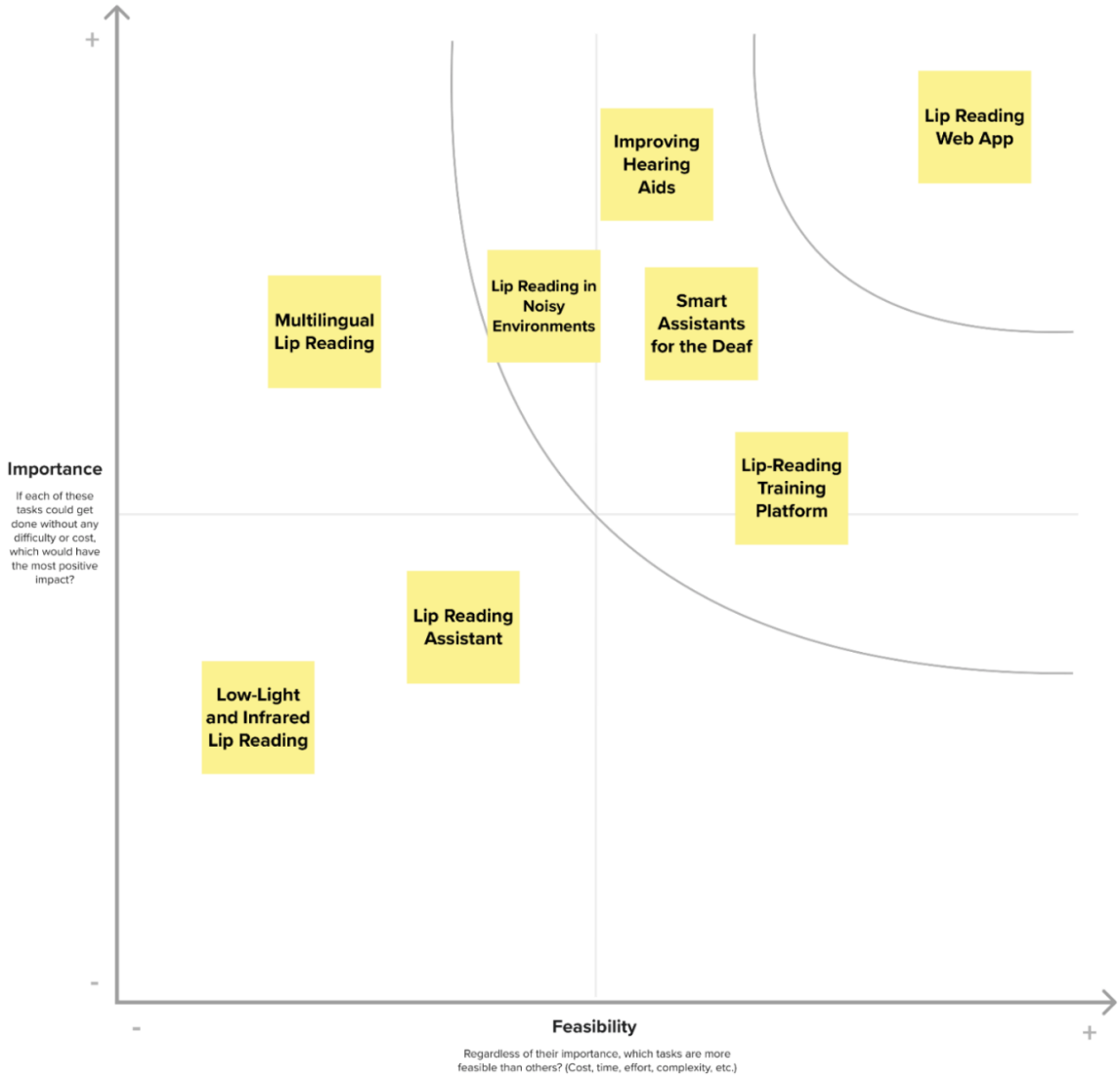
Prioritize

Your team should all be on the same page about what's important moving forward. Place your ideas on this grid to determine which ideas are important and which are feasible.

20 minutes

TIP

Participants can use their cursors to point at where sticky notes should go on the grid. The facilitator can confirm the spot by using the laser pointer holding the **H key** on the keyboard.



4. Requirement Analysis

4.1 Functional Requirements:

1. Data Collection and Training:
- Scalable Data Collection: The solution should be able to collect and annotate a large and diverse dataset, accommodating additional data as more users and languages are introduced.
The machine learning model's training infrastructure should be scalable to handle more data and potentially larger, more complex models as needed.

2. Real-time Processing:
- Scalable Processing Power: Ensure that the system can scale horizontally to handle more concurrent real-time processing requests. This may involve load balancing and using cloud-based resources.

3. User Interface and Deployment:
- Scalable Deployment: Design the user interface and deployment architecture to handle a growing number of users and different platforms (e.g., mobile, web, desktop).
- Internationalization: Scalability should also include the ability to add support for additional languages and regional variations in lip movements.

4. Personalization and Adaptation:
- Scalable Personalization: As more users opt for personalization, the system should be able to adapt to individual preferences without sacrificing performance.

5. Privacy and Security:
- Scalable Security Measures: As the user base expands, scaling security measures to protect user data and privacy is essential.

4.2 Non-Functional Requirements

1. Feedback and Improvement Loop:

- Scalable Feedback Handling: Manage and process user feedback efficiently as the user base grows, as this is crucial for continuous learning and model improvement.

2. Customer Support:

- Scalable Customer Support: Plan for scalable customer support infrastructure to assist users and address their questions or issues.

3. Infrastructure and Cost Considerations:

- Cost Management: Scaling infrastructure and resources can come with increased costs. Ensure that the business model can support the scaling requirements and that revenue growth aligns with the increased operational costs.

5. PROJECT DESIGN

5.1 Data Flow Diagrams & User Stories

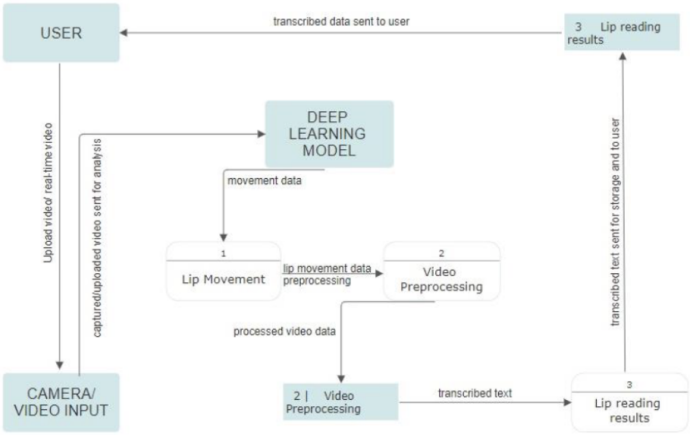


Fig.: DATA FLOW DIAGRAM OF LIP READING USING DEEP LEARNING

User Stories

Use the below template to list all the user stories for the product.

User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Release
Customer (Mobile user)	Registration	USN-1	As a user, I can create an account with a username and password.	I can access my account/dashboard.	High	Sprint-1
		USN-2	As a user, I will receive a confirmation emailonce I have created my account.	I can receive a confirmation email.	High	Sprint-1
		USN-3	As a user, I can create my account through other social accounts.	I can register and access thedashboard with other social accounts' logins.	Low	Sprint-2
		USN-4	As a user, I can register for the application through Gmail.		Medium	Sprint-1
	Login	USN-5	As a user, I can log into the application by entering my email and password.	I can log in securely to access the lip reading system.	High	Sprint-1
	Dashboard	USN-6	As a user, I want to upload a video with clear lip movements for analysis.	I can upload videos to be analyzed.	High	Sprint-2
		USN-7	As a user, I want to see the transcribed text from lip movement analysis.	I can view the transcribed text of my uploaded videos.	High	Sprint –2
		USN-8	As a user, I should be able to choose a specific video from my uploaded videos for lip reading.	I can select at my convenience a particular uploaded video for analysis.	Medium	Sprint-2
		USN-9	As a user, I should be able to access previously generated lip-reading results for a specific video.	I can view previously analyzed results of my uploaded videos.	Medium	Sprint-3
		USN-10	As a user, I should be able to download or share the lip-reading results in various formats (text, subtitles, etc.).	I can export lip reading results.	Low	Sprint-5
		USN-11	As a user, I should be able to use a webcam to perform real-time lip reading on live conversations.	I can perform Real-time lip reading.	High	Sprint-2
		USN-12	As a user, I should be able to control the speed of video playback for more accurate lip-reading results.	I can adjust video playback speed.	Medium	Sprint-3
		USN-13	As a user, I should be able to request a re-analysis of a previously uploaded video if	I can request re-analysis of a previously uploaded video.	Low	Sprint-4

			the initial results were inaccurate.			
		USN-14	As a user. I should be able to integrate the lip-reading system with other applications through an API.	I can access Lip Reading API.	High	Sprint-4
		USN-15	As a user, I should be able to see the confidence score or accuracy rating for the lip-reading results.	I can view Lip reading accuracy.	Medium	Sprint-8
		USN-16	As a user, I should be able to update my user profile information, including email, password, and profile picture.	I can manage my user profile efficiently.	Medium	Sprint-1
		USN-17	As a user, I should be able to access a knowledge base, FAQs, or contact customer support for assistance.	I can access help and support when need be.	Medium	Sprint-5
Administrator	User Management	USN-18	As an administrator, I should be able to add user accounts.	I can manage user account by adding them.	High	Sprint-6
		USN-19	As an administrator, I should be able to suspend or delete user accounts.	I can efficiently manage redundant and dead traffic by deleting user accounts.	High	Sprint-6
	System Maintenance	USN-20	As an administrator, I should be able to perform routine maintenance tasks to ensure the system's reliability and performance.	I can look after the system's health via routine maintenance checkups.	High	Sprint-6
	Monitor usage	USN-21	As an administrator, I should be able to Track system usage and generate usage reports for analysis.	I can monitor the traffic and usage.	High	Sprint-5
	Manage content	USN-22	As an administrator, I should be able to remove inappropriate or copyright-violating content uploaded by users.	I can manage content to ensure reliable content.	High	Sprint-7
Developer	API Documentation	USN-23	As a developer, I should be able to access comprehensive documentation for the lip-reading system's API.	I can manage API Documentation.	High	Sprint-3
	Integration Support	USN-24	As a developer, I should be able to receive technical support for integrating the lip-reading system with external applications.	I can integrate support for the lip-reading system with external applications.	High	Sprint-7
	Access Training Data	USN-25	As a developer, I should be able to utilize a labeled dataset for training and fine-tuning the deep-learning model.	I can access labeled dataset for better training of the deep-learning model.	High	Sprint-7
	Model Update	USN-26	As a developer, I should be able to update the deep learning model periodically to improve accuracy and adapt to new languages or accents.	I can update the model periodically to improve accuracy or add new languages or accents.	Medium	Sprint-8

5.2 Solution Architecture

Solution architecture is a complex process – with many sub-processes – that bridges the gap between business problems and technology solutions. Its goals are to:

• Find the best tech solution to solve existing business problems. Solution:

1. Tech Solution: The most promising approach to address the current limitations of traditional lip-reading methods involves leveraging deep learning techniques. Specifically, we propose the utilization of Convolutional Neural Networks (CNN) for feature extraction and Recurrent Neural Networks (RNN) for sequence-to- sequence mapping. This combination allows the system to capture and interpret intricate lip movements and translate them into spoken words with a high degree of accuracy.
2. Training Data: The success of our approach hinges on the availability of a substantial and diverse dataset. We intend to train the model on a comprehensive dataset comprising lip movements and their corresponding textual transcriptions. This training process enables the deep learning model to learn the intricate relationship between lip patterns and spoken language. As the dataset grows and becomes more diverse, the model's ability to accurately interpret lip movements will continually improve.
3. PotentialBenefits:Theadoptionofdeeplearningtechniquesoffersawealthof potential benefits. It can significantly enhance the accuracy of lip reading, making it a practical solution for real-world applications. Whether used to aid communication for individuals with hearing impairments or to improve security and surveillance systems, this approach holds the potential to revolutionize how we understand and utilize lip movements for communication.

• Describe the structure, characteristics, behavior, and other aspects of the software to project stakeholders.

Solution:

1. Architectural Visualization: Clear and visually engaging architectural diagrams and visuals will be developed to illustrate the software's structure. These diagrams will highlight the integral components, including the CNN and RNN layers, highlighting how they work together to interpret lip movements and convert them into text.
2. Key Characteristics: Defining key characteristics is crucial in ensuring that project stakeholders have a comprehensive understanding of the software. These characteristics include accuracy, the ability to process data in real-time, and scalability to accommodate a growing user base.
3. Behavior and Interaction: To provide stakeholders with a vivid picture of the software's functionality, we will employ real-world use cases and user stories. These scenarios will help illustrate how the system interprets lip movements and transcribes spoken words,

and how it interacts with users and other systems.

4. Stakeholder Engagement: Effective communication with project stakeholders is paramount. To facilitate discussions, address questions, and gather valuable feedback, regular meetings and presentations will be conducted. This open line of communication ensures that the software aligns with stakeholders' expectations and requirements.
5. Documentation Maintenance: To maintain clarity and alignment throughout the project's lifecycle, we commit to ongoing updates and maintenance of project documentation. This includes architectural diagrams, characteristic definitions, and use cases to reflect any changes or enhancements made during the project.

• Define features, development phases, and solution requirements.
Solution:

1. Comprehensive Feature Specification: A detailed feature specification document will be created to encompass all required functionalities. These features range from real-time transcription, enabling multi-modal integration with audio, to providing accessibility features for hearing-impaired individuals. Each feature will be exhaustively defined and prioritized based on its significance to the project.

a. Real-timetranscription:Thesystemwillbeabletotranscribelip movements into text in real time.
b. Multi-modal integration with audio: The system will be able to integrate with existing audio-based speech recognition systems to improve accuracy.
c. Accessibility features for hearing-impaired individuals: The system will be accessible to hearing-impaired individuals by providing features such as text-to-speech and visual feedback.

2. Distinct Development Phases: To ensure a well-structured and organized development process, the project will be broken down into distinct phases. Clear milestones and deliverables will be defined for each phase, providing a roadmap for project progress. These phases include:

a. Datacollection:Collectingofalargeanddiversedatasetoflip movements and corresponding text.
b. Model training: Training a deep learning model on the collected dataset to learn the relationship between lip movements and spoken words.

c. Integration: Integrating the trained model with other systems, such as audio-based speech recognition systems, to create a complete lip- reading system.

d. Testing:Testingthelip-readingsystemonaheld-outdatasettoevaluate its performance.

3. Solution Requirements: A comprehensive set of solution requirements will be documented. These requirements will cover both technical and non-technical aspects, including data sources, accuracy thresholds, hardware and software dependencies, and compliance with industry standards. By explicitly defining these requirements, we set clear expectations for the project's success.

a. Accuracy: Achieving a high degree of accuracy in transcribing lip movements into text.

b. Real-time processing: Transcribing lip movements into text in real time.

c. Scalability: Scalability to handle many concurrent users.

d. Accessibility:Accessibletohearing-impairedindividuals.

4. Scope Management: Clearly outlining the project's boundaries is essential to prevent scope creep. This prevents any uncontrolled changes or additions during the project, ensuring that it stays aligned with its initial goals and objectives.

5. Project Tracking: Utilizing project management tools is crucial for tracking and managing the progress of each development phase. This ensures that the project remains on schedule and within budget, making any necessary adjustments along the way.

• Provide specifications according to which the solution is defined, managed, and delivered.
Solution:

1. Technical Specifications: Detailed technical specifications will be meticulously crafted, outlining precisely how the deep learning models will be built and configured to achieve accurate lip reading. This includes specifics about data preprocessing, model architecture, and training algorithms.

I. The system will be developed using deep learning and computer vision technologies.
II. The system will be able to transcribe lip movements into text in real time, with minimal latency.
III. The system will be able to integrate with existing audio-based speech recognition systems to improve accuracy.
IV. The system will be able to operate in a variety of environments, including noisy environments and low-light conditions.

2. Quality Assurance Standards: Stringent quality assurance standards and procedures will be put in place to ensure that the solution consistently meets predefined quality benchmarks. This includes extensive testing and validation at various stages of development.

a. The system will be tested on a large and diverse dataset of lip movement recordings.

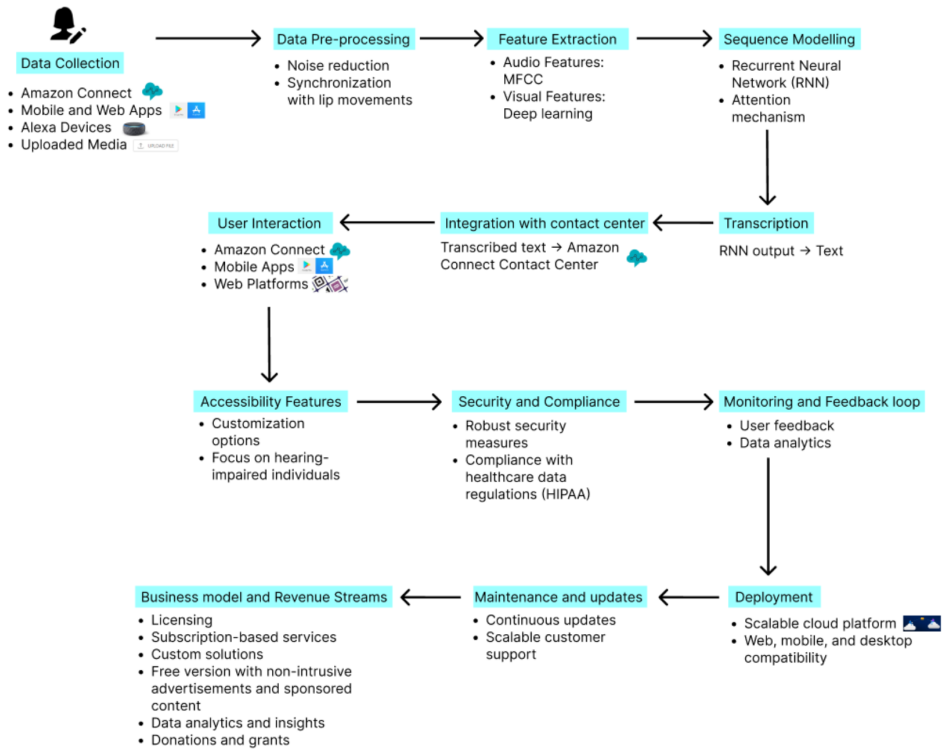
3. Project Management Guidelines: Clearly defined project management guidelines will be established, delineating roles and responsibilities within the project team. This will set timelines, allocate resources, and establish communication protocols to ensure a well-organized and efficient development process.

a. The Project team will hold regular standup meetings, planning meetings, and review meetings to track progress and identify any potential roadblocks

4. ChangeControlProcedures:Aneffectivechangecontrolprocesswillbe implemented to manage any modifications to the solution throughout the project's lifecycle. This includes documenting change requests, evaluating their impact, and implementing them with thorough testing.

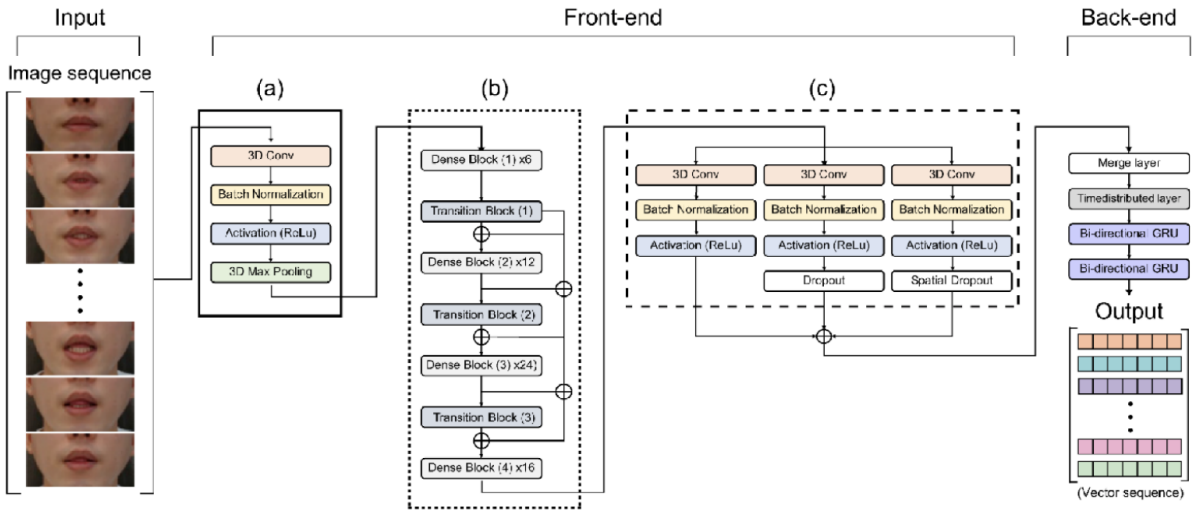
5. Regular Updates and Communication: Maintaining open and regular communication with the project team is crucial to ensure that the project aligns with the defined specifications. This ongoing dialogue allows for quick adaptation to any changes or refinements.

Example - Solution Architecture Diagram:



6. PROJECT PLANNING & SCHEDULING

6.1 Technical Architecture



6.2 Sprint Planning & Estimation

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint-1	Registration	USN-1	As a user, I can create an account with a username and password.	2	High	Isha
Sprint-1		USN-2	As a user, I will receive a confirmation emailonce I have created my account.	2	High	Samarth
Sprint-2		USN-3	As a user, I can create my account through other social accounts.	1	Low	Nishika
Sprint-1		USN-4	As a user, I can register for the applicationthrough Gmail.	1	Medium	Ashish
Sprint-1	Login	USN-5	As a user, I can log into the application byentering my email and password.	3	High	Nishika
Sprint-2	Dashboard	USN-6	As a user, I want to upload a video with clear lip movements for analysis.	4	High	Samarth
Sprint-2		USN-7	As a user, I want to see the transcribed text from lip movement analysis.	4	High	Isha
Sprint-2		USN-8	As a user, I should be able to choose a specific video from my uploaded videos for lip reading.	2	Medium	Ashish
Sprint-3		USN-9	As a user, I should be able to access previously generated lip-reading results for a specific video.	2	Medium	Samarth

Sprint-5		USN-10	As a user, I should be able to download or share the lip-reading results in various formats (text, subtitles, etc.).	1	Low	Isha
Sprint-2		USN-11	As a user, I should be able to use a webcam to perform real-time lip reading on live conversations.	4	High	Nishika
Sprint-3		USN-12	As a user, I should be able to control the speed of video playback for more accurate lip-reading results.	2	Medium	Samarth
Sprint-4		USN-13	As a user, I should be able to request a re-analysis of a previously uploaded video if the initial results were inaccurate.	1	Low	Isha
Sprint-4		USN-14	As a user. I should be able to integrate the lip-reading system with other applications through an API.	3	High	Samarth
Sprint-8		USN-15	As a user, I should be able to see the confidence score or accuracy rating for the lip-reading results.	2	Medium	Ashish
Sprint-1		USN-16	As a user, I should be able to update my user profile information, including email, password, and profile picture.	2	Medium	Nishika
Sprint-5		USN-17	As a user, I should be able to access a knowledge base, FAQs, or contact customer support for assistance.	2	Medium	Ashish
Sprint-6	User Management	USN-18	As an administrator, I should be able to add user accounts.	3	High	Samarth
Sprint-6		USN-19	As an administrator, I should be able to suspend or delete user accounts.	3	High	Nishika
Sprint-6	System Maintenance	USN-20	As an administrator, I should be able to perform routine maintenance tasks to ensure the system's reliability and performance.	3	High	Isha
Sprint-5	Monitor usage	USN-21	As an administrator, I should be able to Track system usage and generate usage reports for analysis.	2	High	Ashish
Sprint-7	Manage content	USN-22	As an administrator, I should be able to remove inappropriate or copyright-violating content uploaded by users.	2	High	Nishika
Sprint-3	API Documentation	USN-23	As a developer, I should be able to access comprehensive documentation for the lip-reading system's API.	4	High	Samarth
Sprint-7	Integration Support	USN-24	As a developer, I should be able to receive technical support for integrating the lip-reading system with external applications.	3	High	Isha
Sprint-7	Access Training Data	USN-25	As a developer, I should be able to utilize a labeled dataset for training and fine-tuning the deep-learning model.	4	High	Samarth
Sprint-8	Model Update	USN-26	As a developer, I should be able to update the deep learning model periodically to improve accuracy and adapt to new languages or accents.	2	Medium	Ashish

7. CODING & SOLUTIONING

7.1importing necessary libraries

```
import os
import cv2
import tensorflow as tf
import numpy as np
from typing import List
from matplotlib import pyplot as plt
import imageio
```

7.2Define the required functions:

1. load_video()

```
def load_video(path:str) -> List[float]:

    cap = cv2.VideoCapture(path)
    frames = []
    for _ in range(int(cap.get(cv2.CAP_PROP_FRAME_COUNT))):
        ret, frame = cap.read()
        frame = tf.image.rgb_to_grayscale(frame)
        frames.append(frame[190:236,80:220,:])
    cap.release()

    mean = tf.math.reduce_mean(frames)
    std = tf.math.reduce_std(tf.cast(frames, tf.float32))
    return tf.cast((frames - mean), tf.float32) / std
```

2. char_to_num() and num_to_char():

```
vocab = [x for x in "abcdefghijklmnopqrstuvwxyz'?!123456789 "]
char_to_num = tf.keras.layers.StringLookup(vocabulary=vocab, oov_token="")
num_to_char = tf.keras.layers.StringLookup(
    vocabulary=char_to_num.get_vocabulary(), oov_token="", invert=True
)
```

3.load_allignments():

```
def load_alignments(path:str) -> List[str]:
    with open(path, 'r') as f:
        lines = f.readlines()
    tokens = []
    for line in lines:
        line = line.split()
        if line[2] != 'sil':
            tokens = [*tokens,' ',line[2]]
    return char_to_num(tf.reshape(tf.strings.unicode_split(tokens, input_encoding='UTF-8'), (-1)))[1:]
```

4.load_data():

```
def load_data(path: str):
    path = bytes.decode(path.numpy())
    file_name = path.split('/')[-1].split('.')[0]
    video_path = os.path.join('/kaggle','input','lipreading','data','s1',f'{file_name}.mpg')
    alignment_path = os.path.join('/kaggle','input','lipreading','data','alignments','s1',f'{file_name}.align')
    frames = load_video(video_path)
    alignments = load_alignments(alignment_path)

    return frames, alignments
```

now lets test these functions and convert a test input into tensor:

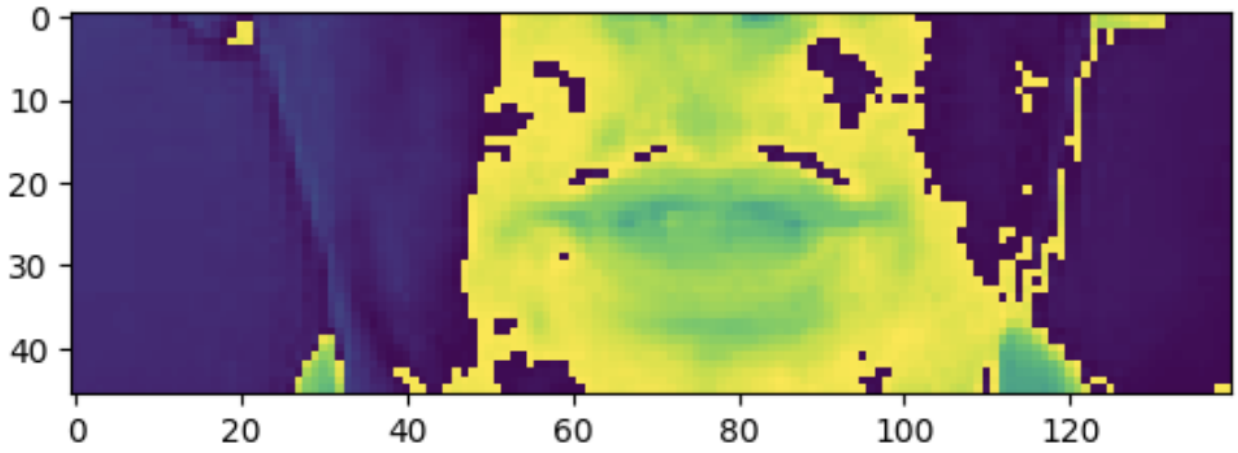
```
test_path = '/kaggle/input/lipreading/data/s1/bbaf2n.mpg'
tf.convert_to_tensor(test_path).numpy().decode('utf-8').split('/')[-1].split('.')[0]
frames, alignments = load_data(tf.convert_to_tensor(test_path))
frames
```

the output will be something like:

```
[9]: <tf.Tensor: shape=(75, 46, 140, 1), dtype=float32, numpy=
     array([[[[1.4991664 ],
              [1.4991664 ],
              [1.4616872 ],
              ...,
              [0.41227075],
              [0.41227075],
              [0.41227075]],

             [[1.4991664 ],
              [1.4991664 ],
              [1.4616872 ],
```

7.3 visualize a frame of the video:

```
plt.imshow(frames[35])
```



7.4 Spliting the data into training and testing

first we will need a mappable function:

```
def mappable_function(path:str) ->List[str]:
    result = tf.py_function(load_data, [path], (tf.float32, tf.int64))
    return result
```

next we can use the code shown below to split into train and test equally:

```
data = tf.data.Dataset.list_files('/kaggle/input/lipreading/data/s1/*.mpg')
data = data.shuffle(500, reshuffle_each_iteration=False)
data = data.map(mappable_function)
data = data.padded_batch(2, padded_shapes=([75,None,None,None],[40]))
data = data.prefetch(tf.data.AUTOTUNE)
# Added for split
train = data.take(450)
test = data.skip(450)
```

7.5 Building a model

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv3D, LSTM, Dense, Dropout
from tensorflow.keras.layers import Bidirectional, MaxPool3D, Activation, Reshape
from tensorflow.keras.layers import SpatialDropout3D, BatchNormalization
from tensorflow.keras.layers import TimeDistributed, Flatten
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ModelCheckpoint, LearningRateScheduler
```

```
model = Sequential()
model.add(Conv3D(128, 3, input_shape=(75,46,140,1), padding='same'))
model.add(Activation('relu'))
model.add(MaxPool3D((1,2,2)))

model.add(Conv3D(256, 3, padding='same'))
model.add(Activation('relu'))
model.add(MaxPool3D((1,2,2)))

model.add(Conv3D(75, 3, padding='same'))
model.add(Activation('relu'))
model.add(MaxPool3D((1,2,2)))

model.add(TimeDistributed(Flatten()))

model.add(Bidirectional(LSTM(128, kernel_initializer='Orthogonal', return_sequences=True)))
model.add(Dropout(.5))

model.add(Bidirectional(LSTM(128, kernel_initializer='Orthogonal', return_sequences=True)))
model.add(Dropout(.5))

model.add(Dense(char_to_num.vocabulary_size()+1, kernel_initializer='he_normal', activation='softmax'))
```

7.5 defining the callback functions, loss function and compiling the model

```
def scheduler(epoch, lr):
    if epoch < 30:
        return lr
    else:
        return lr * tf.math.exp(-0.1)
```

```
def CTCLoss(y_true, y_pred):
    batch_len = tf.cast(tf.shape(y_true)[0], dtype="int64")
    input_length = tf.cast(tf.shape(y_pred)[1], dtype="int64")
    label_length = tf.cast(tf.shape(y_true)[1], dtype="int64")

    input_length = input_length * tf.ones(shape=(batch_len, 1), dtype="int64")
    label_length = label_length * tf.ones(shape=(batch_len, 1), dtype="int64")

    loss = tf.keras.backend.ctc_batch_cost(y_true, y_pred, input_length, label_length)
    return loss
```

```python
class ProduceExample(tf.keras.callbacks.Callback):
    def __init__(self, dataset) -> None:
        self.dataset = dataset.as_numpy_iterator()

    def on_epoch_end(self, epoch, logs=None) -> None:
        data = self.dataset.next()
        yhat = self.model.predict(data[0])
        decoded = tf.keras.backend.ctc_decode(yhat, [75,75], greedy=False)[0][0].numpy()
        for x in range(len(yhat)):
            print('Original:', tf.strings.reduce_join(num_to_char(data[1][x])).numpy().decode('utf-8'))
            print('Prediction:', tf.strings.reduce_join(num_to_char(decoded[x])).numpy().decode('utf-8'))
            print('~'*100)
```

```python
model.compile(optimizer=Adam(learning_rate=0.0001), loss=CTCLoss)
```

```python
checkpoint_callback = ModelCheckpoint(os.path.join('models','checkpoint'), monitor='loss', save_weights_only=True)
```

```python
schedule_callback = LearningRateScheduler(scheduler)
```

7.6 training the model:

```python
model.fit(train, validation_data=test, epochs=100, callbacks=[checkpoint_callback, schedule_callback, example_callback])
```

```
Epoch 1/100
450/450 [==============================] - ETA: 0s - loss: 85.5234
[mpeg1video @ 0x793cdc021ec0] ac-tex damaged at 22 17
[mpeg1video @ 0x793cdc021ec0] Warning MVs not available
[mpeg1video @ 0x793cc406d580] ac-tex damaged at 22 17
[mpeg1video @ 0x793cc406d580] Warning MVs not available
1/1 [==============================] - 2s 2s/step
Original: place blue in o six please
Prediction: l e e e e n
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Original: set red with v three again
Prediction: l e e e e e n
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
450/450 [==============================] - 785s 2s/step - loss: 85.5234 - val_loss: 72.2875 - lr: 1.0000e-04
Epoch 2/100
111/450 [======>.....................] - ETA: 4:02 - loss: 72.8648
[mpeg1video @ 0x793cc4057100] ac-tex damaged at 22 17
[mpeg1video @ 0x793cc4057100] Warning MVs not available
450/450 [==============================] - ETA: 0s - loss: 70.7376
[mpeg1video @ 0x7940db18c180] ac-tex damaged at 22 17
[mpeg1video @ 0x7940db18c180] Warning MVs not available
1/1 [==============================] - 0s 122ms/step
Original: bin red at g one again
Prediction: la e e e e o
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Original: bin blue by z eight now
Prediction: la e e e e o
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
450/450 [==============================] - 551s 1s/step - loss: 70.7376 - val_loss: 65.0712 - lr: 1.0000e-04
Epoch 3/100
141/450 [========>.....................] - ETA: 3:39 - loss: 68.1182
[mpeg1video @ 0x794278075100] ac-tex damaged at 22 17
[mpeg1video @ 0x794278075100] Warning MVs not available
450/450 [==============================] - ETA: 0s - loss: 66.5817
[mpeg1video @ 0x793cd883b600] ac-tex damaged at 22 17
[mpeg1video @ 0x793cd883b600] Warning MVs not available
1/1 [==============================] - 0s 117ms/step
Original: lay green in l seven again
Prediction: la e e e e ea
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Original: bin white in t zero please
Prediction: la e e e e ea
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

now wait for the model to get train for upto 100 epoches

8. PERFORMANCE TESTING:

now since we have saved the model at the location "models/checkpoint"

so first load the weights:

```
model.load_weights('models/checkpoint')
```

now lets load a video to prerdict

```
sample = load_data(tf.convert_to_tensor('/kaggle/input/lipreading/data/s1/bbas2p.mpg'))
```

lets see the real text

```
print('~'*100, 'REAL TEXT')
[tf.strings.reduce_join([num_to_char(word) for word in sentence]) for sentence in [sample[1]]]
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ REAL TEXT
[<tf.Tensor: shape=(), dtype=string, numpy=b'bin blue at s two please'>]
```

Now lets predict using the model

```
yhat = model.predict(tf.expand_dims(sample[0], axis=0))
```

```
1/1 [==============================] - 5s 5s/step
```

```
decoded = tf.keras.backend.ctc_decode(yhat, input_length=[75], greedy=True)[0][0].numpy()
```

```
print('~'*100, 'PREDICTIONS')
[tf.strings.reduce_join([num_to_char(word) for word in sentence]) for sentence in decoded]
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ PREDICTIONS
[<tf.Tensor: shape=(), dtype=string, numpy=b'bin blue at s two please'>]
```

we can see that the predicted text matches exactly with the real text.

now lets test the accurracy and Levenshtein Distance of the model

Accuracy: In the context of classification, accuracy measures the proportion of correctly classified instances among the total instances evaluated. It's calculated as the number of correct predictions divided by the total number of predictions made.

Levenshtein Distance: Levenshtein Distance quantifies the minimum number of single-character edits (insertions, deletions, substitutions) required to transform one string into another. It measures the dissimilarity or distance between two strings by counting the minimum number of operations needed to make them identical.

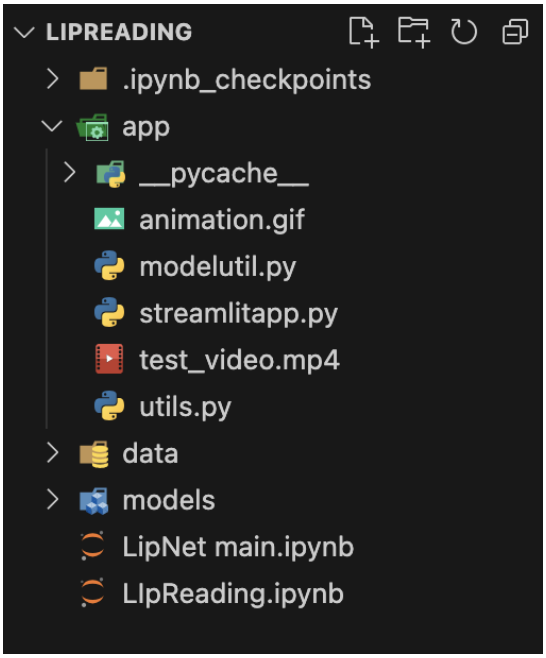
```
from pathlib import Path

predicted_sequences = []
ground_truth_sequences = []
directory_path = Path('/kaggle/input/lipreading/data/s1')  # Replace this with your directory path

# Get all files with a specific extension (e.g., '.mpg') in the directory
file_paths = list(directory_path.glob('*.mpg'))  # Convert the generator to a list to get the length

er = ''
# Use tqdm to create a progress bar for the iteration
for file_path in file_paths:
    # Process each file or load data as needed
    er = file_path
    print(er)
    sample = load_data(tf.convert_to_tensor(str(file_path)))
    yhat = model.predict(tf.expand_dims(sample[0], axis=0))
    decoded = tf.keras.backend.ctc_decode(yhat, input_length=[75], greedy=True)[0][0].numpy()
    # Perform operations using 'sample'
    predicted_sequences.append([tf.strings.reduce_join([num_to_char(word) for word in sentence]) for sentence in decoded][0].numpy().decode('utf-8'))
    ground_truth_sequences.append([tf.strings.reduce_join([num_to_char(word) for word in sentence]) for sentence in [sample[1]]][0].numpy().decode('utf-8'))
```

```
import Levenshtein
# Initialize lists to accumulate individual accuracy and Levenshtein distances
accuracies = []
lev_distances = []

# Loop through each pair of predicted and ground truth sequences
for pred_seq, gt_seq in zip(predicted_sequences, ground_truth_sequences):
    # Calculate accuracy based on string equality
    accuracy = 1.0 if pred_seq == gt_seq else 0.0
    accuracies.append(accuracy)

    # Calculate Levenshtein distance
    levenshtein_distance = Levenshtein.distance(pred_seq, gt_seq)
    max_length = max(len(pred_seq), len(gt_seq))
    normalized_distance = levenshtein_distance / max_length
    lev_distances.append(normalized_distance)

# Compute overall statistics across all sequences
overall_accuracy = sum(accuracies) / len(accuracies)
average_lev_distance = sum(lev_distances) / len(lev_distances)

print("Overall Accuracy:", overall_accuracy)
print("Average Levenshtein Distance:", average_lev_distance)
```

```
Overall Accuracy: 0.9115577889447236
Average Levenshtein Distance: 0.006815311039102414
```

9.Integration with the Web App

the file structure should look something like this:



As you can see from the file structure above we need to create thrree python files inside app folder

1. modelutil.py
2. utils.py
3. streamlitapp.py

Inside of modelutil.py file we will need to add the model:

```python
import os
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv3D, LSTM, Dense, Dropout
from tensorflow.keras.layers import Bidirectional, MaxPool3D, Activation, Reshape
from tensorflow.keras.layers import SpatialDropout3D, BatchNormalization
from tensorflow.keras.layers import TimeDistributed, Flatten
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ModelCheckpoint, LearningRateScheduler

def load_model() -> Sequential:
    model = Sequential()

    model.add(Conv3D(128, 3, input_shape=(75,46,140,1), padding='same'))
    model.add(Activation('relu'))
    model.add(MaxPool3D((1,2,2)))

    model.add(Conv3D(256, 3, padding='same'))
    model.add(Activation('relu'))
    model.add(MaxPool3D((1,2,2)))

    model.add(Conv3D(75, 3, padding='same'))
    model.add(Activation('relu'))
    model.add(MaxPool3D((1,2,2)))

    model.add(TimeDistributed(Flatten()))

    model.add(Bidirectional(LSTM(128, kernel_initializer='Orthogonal', return_sequences=True)))
    model.add(Dropout(.5))

    model.add(Bidirectional(LSTM(128, kernel_initializer='Orthogonal', return_sequences=True)))
    model.add(Dropout(.5))

    model.add(Dense(41, kernel_initializer='he_normal', activation='softmax'))

    model.load_weights(os.path.join('..','models','checkpoint'))

    return model
```

Inside of modelutil.py file we will need to add all the user defined functions which are char_to_num, num_to_char, load_video(), load_alignments(), load_data() functions:

```python
import tensorflow as tf
from typing import List
import cv2
import os

vocab = [x for x in "abcdefghijklmnopqrstuvwxyz'?!123456789 "]
char_to_num = tf.keras.layers.StringLookup(vocabulary=vocab, oov_token="")
# Mapping integers back to original characters
num_to_char = tf.keras.layers.StringLookup(
    vocabulary=char_to_num.get_vocabulary(), oov_token="", invert=True
)

def load_video(path:str) -> List[float]:
    #print(path)
    cap = cv2.VideoCapture(path)
    frames = []
    for _ in range(int(cap.get(cv2.CAP_PROP_FRAME_COUNT))):
        ret, frame = cap.read()
        frame = tf.image.rgb_to_grayscale(frame)
        frames.append(frame[190:236,80:220,:])
    cap.release()

    mean = tf.math.reduce_mean(frames)
    std = tf.math.reduce_std(tf.cast(frames, tf.float32))
    return tf.cast((frames - mean), tf.float32) / std

def load_alignments(path:str) -> List[str]:
    #print(path)
    with open(path, 'r') as f:
        lines = f.readlines()
    tokens = []
    for line in lines:
        line = line.split()
        if line[2] != 'sil':
            tokens = [*tokens,' ',line[2]]
    return char_to_num(tf.reshape(tf.strings.unicode_split(tokens, input_encoding='UTF-8'), (-1)))[1:]

def load_data(path: str):
    path = bytes.decode(path.numpy())
    file_name = path.split('/')[-1].split('.')[0]
    # File name splitting for windows
    # file_name = path.split('\\')[-1].split('.')[0]
    video_path = os.path.join('..','data','s1',f'{file_name}.mpg')
    alignment_path = os.path.join('..','data','alignments','s1',f'{file_name}.align')
    frames = load_video(video_path)
    alignments = load_alignments(alignment_path)

    return frames, alignments
```

Inside streamlitapp.py we will add the logic to host our app on the internet:

first let's import all the necessary libraries:

```
# Import all of the dependencies
import streamlit as st
import os
import imageio

import tensorflow as tf
from utils import load_data, num_to_char
from modelutil import load_model
```

Next set the layout of the web app

```
# Set the layout to the streamlit app as wide
st.set_page_config(layout='wide')
```

let's cofigure the sidebar now:

```
# Setup the sidebar
with st.sidebar:
    st.image('https://www.onepointltd.com/wp-content/uploads/2020/03/inno2.png')
    st.title('Lip Reading')
    st.title('Contributors:')
    st.info('Samarth')
    st.info('Ashish')
    st.info('Isha')
    st.info('Nishika')
```

Now add a little and create a drop down menu for choosing the input:

```
st.title('Lipreading Using Deep Learning')
# Generating a list of options or videos
options = os.listdir(os.path.join('/Users/samarthgayakhe/Documents/projects/LipReading/data/s1'))
selected_video = st.selectbox('Choose video', options)
```

Now create two columns and the configure it:

```
# Generate two columns
col1, col2 = st.columns(2)

if options:

    # Rendering the video
    with col1:
        st.info('The video below displays the converted video in mp4 format')
        file_path = os.path.join('/Users/samarthgayakhe/Documents/projects/LipReading/data/s1', selected_video)
        os.system(f'ffmpeg -i {file_path} -vcodec libx264 test_video.mp4 -y')

        # Rendering inside of the app
        video = open('test_video.mp4', 'rb')
        video_bytes = video.read()
        st.video(video_bytes)


    with col2:
        st.info('This is all the machine learning model sees when making a prediction')
        video, annotations = load_data(tf.convert_to_tensor(file_path))
        # imageio.mimsave('animation.gif', video, fps=10)
        st.image('animation.gif', width=400)

        st.info('This is the output of the machine learning model as tokens')
        model = load_model()
        yhat = model.predict(tf.expand_dims(video, axis=0))
        decoder = tf.keras.backend.ctc_decode(yhat, [75], greedy=True)[0][0].numpy()
        st.text(decoder)

        # Convert prediction to text
        st.info('Decode the raw tokens into words')
        converted_prediction = tf.strings.reduce_join(num_to_char(decoder)).numpy().decode('utf-8')
        st.text(converted_prediction)
```

col1 will display the selected video and play it.

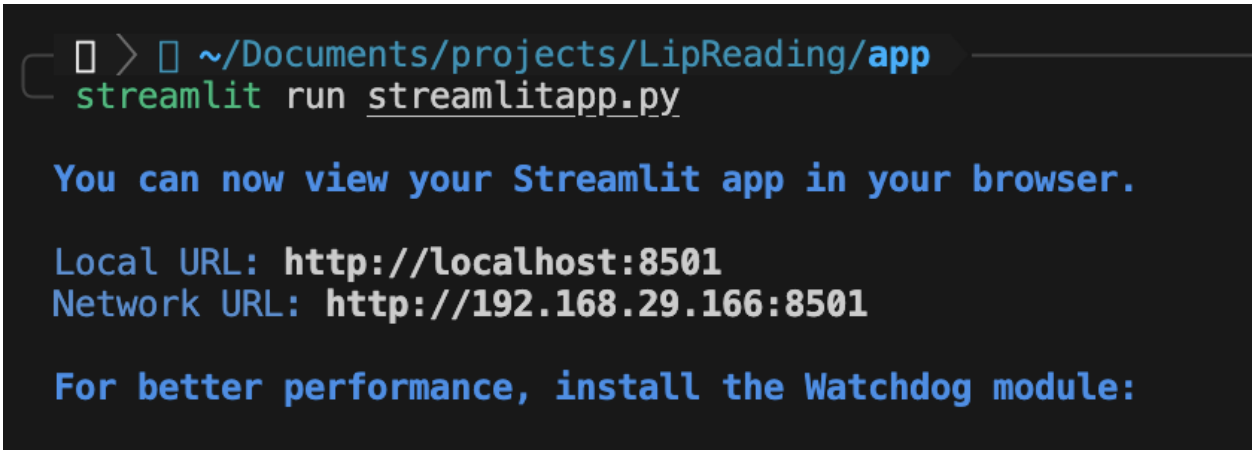
col2 will display :

➔ animation of lips.

➔ raw output of your model

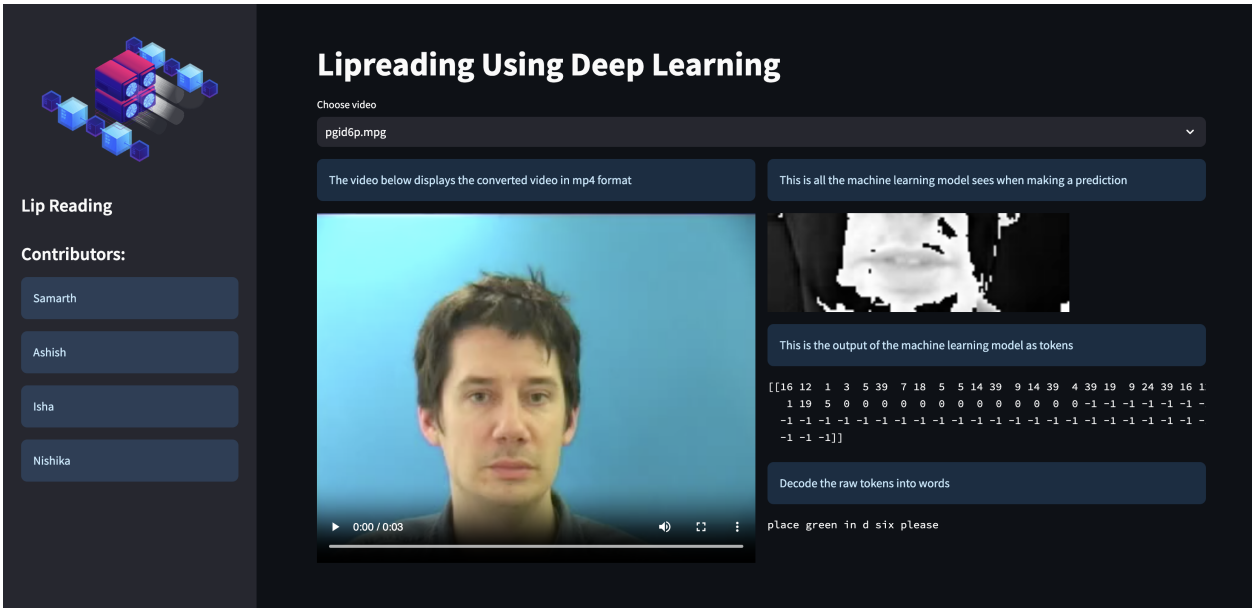
➔ result prediction

Now you can host the webapp by executing the following command in the terminal:



and my clicking on the link we can launch the web app

the web app will look something like this:



10. ADVANTAGES & DISADVANTAGES

Advantages:

- Improved Speech Recognition
- Elimination of Audio Data Dependency
- Multi-Modal Applications
- vAccessibility for Hearing-Impaired Individuals

Disadvantages:

- Limited Datasets
- Complexity of Lip Movements
- Real-Time Processing Challenges

11. CONCLUSION

The lip reading project outlined in this documentation represents a comprehensive and innovative approach to leveraging deep learning for improving speech recognition through lip movements. The project encompasses various phases, from ideation and design to implementation and integration into a web application. Below is a summary of key points:

Key Achievements:

1. End-to-End Lip Reading System: The project successfully develops an end-to-end machine learning solution for lip reading, utilizing deep learning techniques, including Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN).
2. Real-Time Processing: The system is designed to transcribe lip movements into text in real time, enhancing its practical usability in various applications.
3. Scalability and Accessibility: The solution incorporates scalability considerations to handle growing datasets, users, and languages. Additionally, it aims to be accessible to hearing-impaired individuals through features like text-to-speech and visual feedback.
4. Web Application Integration: The deployment of a user-friendly web application allows users to interact with the lip reading system, view lip movements, and obtain real-time predictions.

Challenges and Considerations:

1. Data Availability: The project emphasizes the importance of comprehensive and diverse datasets for training deep learning models. Data limitations could impact model performance and generalization.
2. Model Robustness: Addressing variations in lip movements, accents, and pronunciation is crucial for achieving robust and accurate lip reading across diverse contexts.
3. Real-Time Processing Complexity: Achieving real-time lip reading capabilities introduces computational challenges that need careful consideration.

12. FUTURE SCOPE

The future scope of the lip reading project involves ongoing enhancements and adaptations to meet evolving challenges and user requirements. Potential areas for future development include:

1. Data Enrichment: Continuously expanding and diversifying the dataset to improve model robustness and accuracy, addressing variations in lip movements and spoken language.
2. Model Optimization: Exploring advanced model architectures and optimization techniques to achieve higher accuracy and real-time processing capabilities.
3. Multilingual Support: Extending the system's capabilities to support multiple languages and regional variations in lip movements, broadening its applicability on a global scale.
4. Incorporating Feedback Mechanisms: Implementing robust feedback mechanisms to collect user input and continuously improve the model based on real-world usage and challenges.
5. Enhanced Accessibility Features: Further enhancing accessibility features for individuals with hearing impairments, including additional visual feedback and customization options.
6. Deployment in Diverse Environments: Conducting extensive testing and adaptations to ensure the system's effectiveness in various environments, including noisy conditions and low-light scenarios.