**Project Report**

1. **INTRODUCTION**
   1. Project Overview:

Lip reading, also known as speechreading, is the ability to understand speech by observing visual cues such as lip movements and facial expressions. While lip reading can be an effective communication tool, especially in noisy environments or for individuals with hearing impairments, it is a complex task that requires extensive training and practice. The advent of deep learning has opened up new possibilities for enhancing lip reading accuracy and expanding its applications.

This project aims to develop a deep learning-based lip-reading system that can accurately recognize words and phrases from real-world video examples. The project will focus on the following aspects:

* 1. Purpose:

The purpose of this endeavour is to provide a viable solution for individuals with hearing impairments, offering an alternative method for comprehending spoken language by decoding visual cues from lip movements. Additionally, this technology has potential applications in environments with high noise levels or situations where audio data is unavailable, enhancing communication accessibility in diverse scenarios.

1. **LITERATURE SURVEY**

To identify the current state-of-the-art methods for lip reading using deep learning, their strengths and weaknesses, and gaps in knowledge. We have selected different paths by using the internet resources accurately like ChatGPT, Google search engine, and Google scholar we have gone through many internet resources. Evaluate sources based on credibility, relevance, authority, and recency. Extract key findings, methodologies, and performance metrics. Like Reading alignment and mp4 files and converting them into integers by preprocessing for analysis. Synthesize information by summarizing, identifying themes, and categorizing research. Identify gaps and research opportunities, such as developing robust models and incorporating additional modalities.

* 1. Existing problem:

The existing problem in lip reading technology lies in achieving high accuracy and robustness in interpreting various lip movements across different languages, accents, and speaking styles. Current systems face challenges in real-time processing, dealing with occlusions, and accurately transcribing speech due to variations in lip shapes and contextual ambiguity.

* 1. References:

<https://paperswithcode.com/paper/lipnet-end-to-end-sentence-level-lipreading>

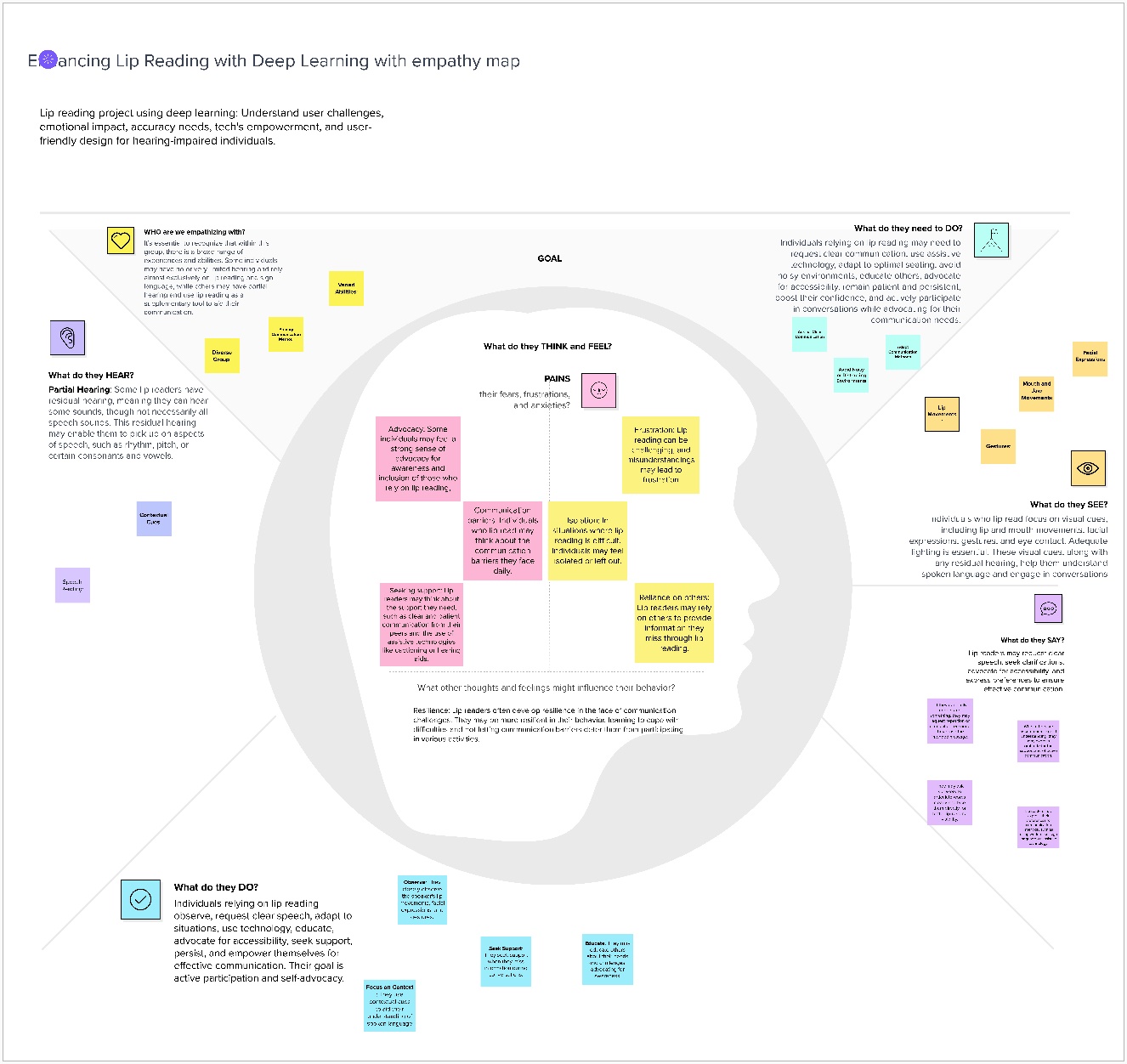
<https://paperswithcode.com/paper/deep-audio-visual-speech-recognition>

<https://ieeexplore.ieee.org/document/10200426>

* 1. Problem Statement Definition:

The project aims to develop an accurate and efficient deep learning-based lip-reading system to facilitate communication for individuals with hearing impairments. The primary challenge is to create a model that can interpret lip movements and convert them into accurate textual or audio representations, allowing communication for the hearing-impaired population.

1. **IDEATION & PROPOSED SOLUTION**
   1. Empathy Map Canvas:



An empathy map is a simple, easy-to-digest visual that captures knowledge about a user’s behaviours and attitudes.

Understanding the thoughts and emotions of individuals who rely on lip reading, and the potential benefits of using deep learning to enhance this ability, can help developers and researchers create more effective and empathetic solutions to address their needs.

* **Say:** Individuals express the need for highly accurate lip-reading technology. They recognize that the current methods are often challenging and not always reliable, making effective communication difficult.
* **Do:** People with lip-reading challenges tend to put in a lot of effort during conversations. They concentrate intensely on the speaker's lips and may frequently ask others to repeat themselves. They actively seek out and use lip-reading apps and tools to assist in their communication.
* **Think/Feel:** Many individuals feel frustration and anxiety about their lip-reading difficulties. They worry about missing important information during conversations and often feel self-conscious about repeatedly asking for clarification. Despite these challenges, there's a strong sense of hope for technology to improve lip-reading accuracy and make communication easier.
* **See:** These individuals often perceive that those around them may not fully understand the difficulties they face in lip reading. They're aware of the limitations of existing lip-reading technologies and may feel that these tools could be improved.
* **Pain/Gain:** The pains they experience include frustration from frequent miscommunication, social isolation resulting from communication challenges, and the dependency on others for accurate information. The gains they hope for are improved accuracy in lip reading using deep learning technologies, increased confidence in social interactions, and reduced reliance on others for effective communication.

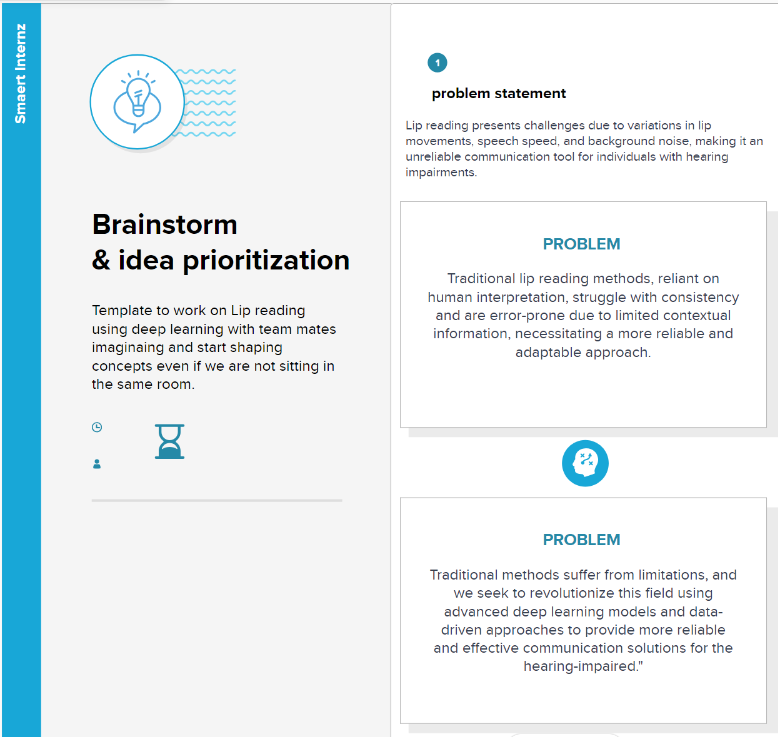
This elaborated empathy map provides a more detailed insight into the thoughts, emotions, and actions of individuals with lip-reading challenges, as well as their aspirations for better technology and communication experiences.

* 1. Ideation & Brainstorming:

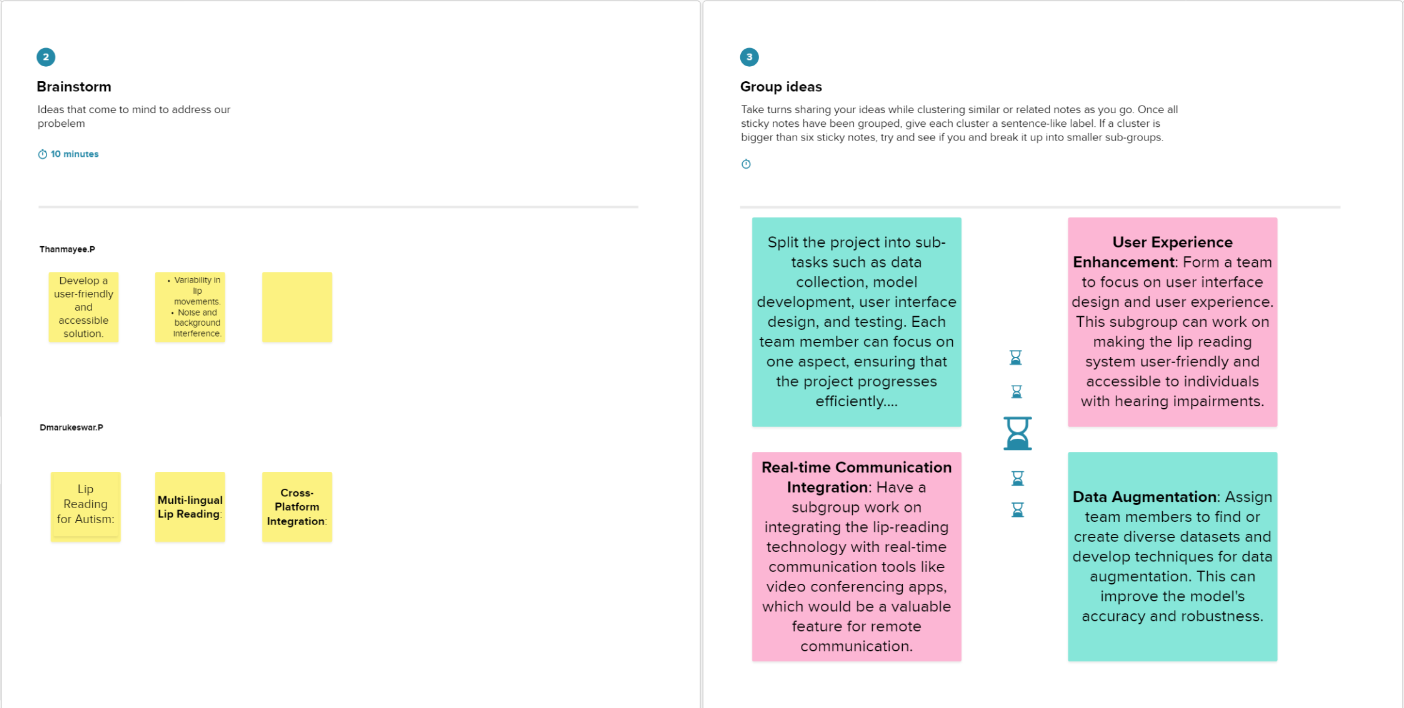
Brainstormed Ideas:

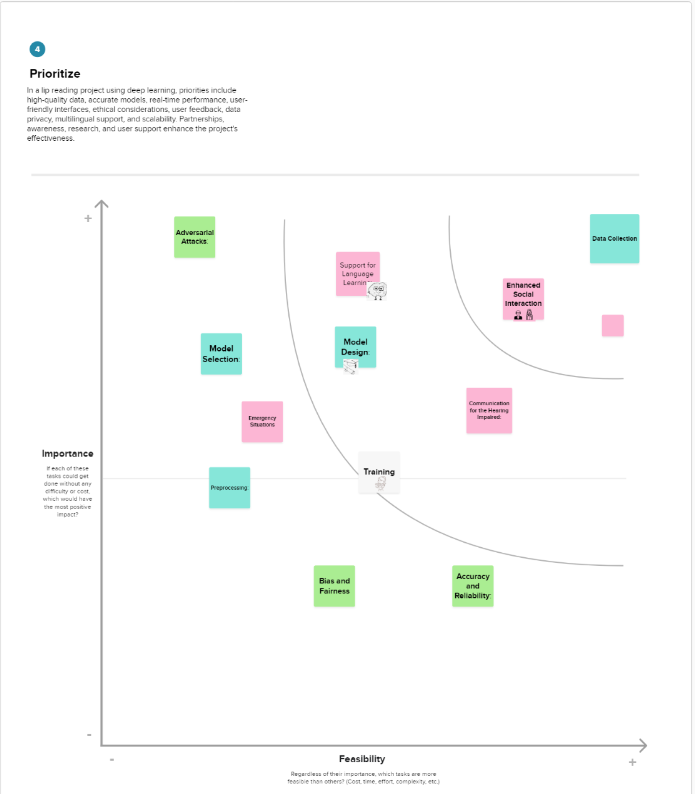
* User-Friendly and Accessible Solution: Develop a user-friendly and accessible solution.
* Multi-Lingual Lip Reading: Implement lip reading for multiple languages.
* Cross-Platform Integration: Integrate lip reading into various communication platforms.
* Lip Reading for Autism: Explore applications for individuals with autism.
* Feasibility Considerations: Assess the feasibility of each idea based on factors like cost, time, effort, and complexity.

**Step-1: Team Gathering, Collaboration and Select the Problem Statement**



**Step-2: Brainstorm, Idea Listing and Grouping**

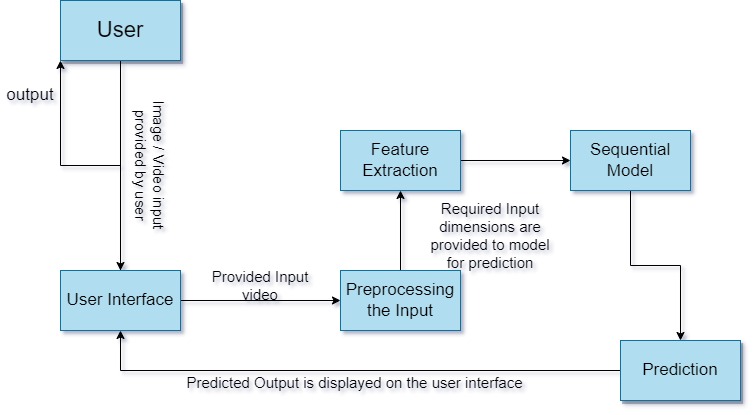
 **Step-3: Idea Prioritization**



1. **REQUIREMENT ANALYSIS**
   1. Functional requirement:

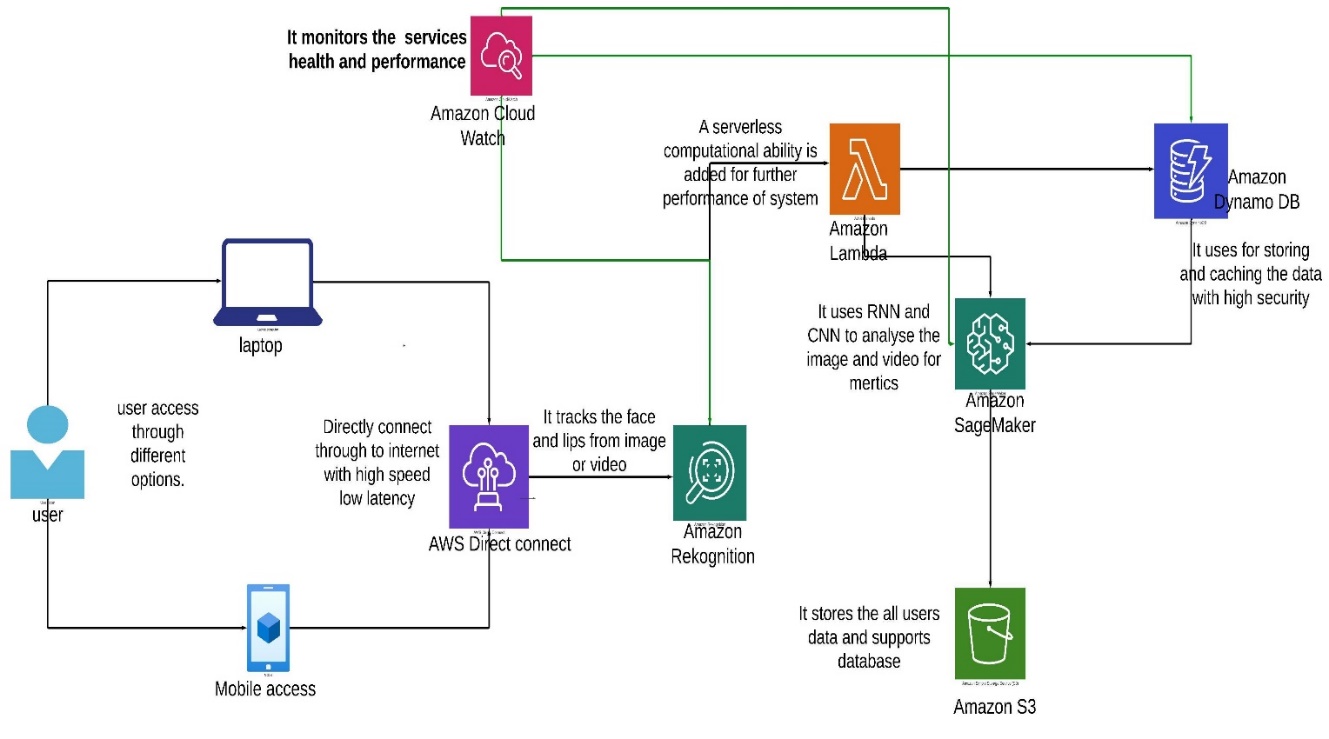
* **Real-time Lip Recognition:** The system should accurately recognize and interpret lip movements in real time.
* **Multi-language Support:** Capability to interpret various languages and accents.
* **Robustness to Occlusions:** Ability to handle partial obstructions of the lips without significant loss in accuracy.
* **Contextual Understanding:** Understanding and deciphering contextually ambiguous lip movements for accurate transcription.
* **User Interface:** Intuitive user interface for easy interaction, possibly with text-based outputs.
  1. Non-Functional Requirements:
* **Accuracy and Reliability:** The system should achieve a high level of accuracy in transcribing spoken language through lip reading.
* **Speed and Efficiency:** Real-time processing with minimal latency for seamless communication.
* **Adaptability:** Ability to adapt to various environments and speaking styles for consistent performance.
* **Security:** Ensuring data privacy and security measures are in place for any stored or transmitted information.
* **Scalability:** Capability to scale the system for potential future enhancements or increased usage without compromising performance.

1. **PROJECT DESIGN**
   1. Data Flow Diagrams & User Stories:





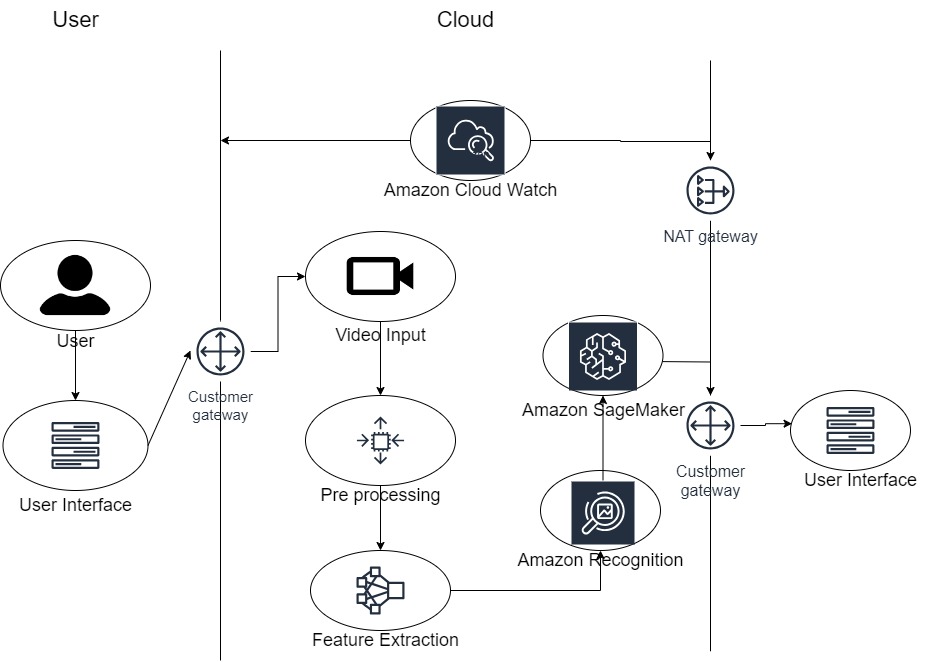
* 1. Solution Architecture:



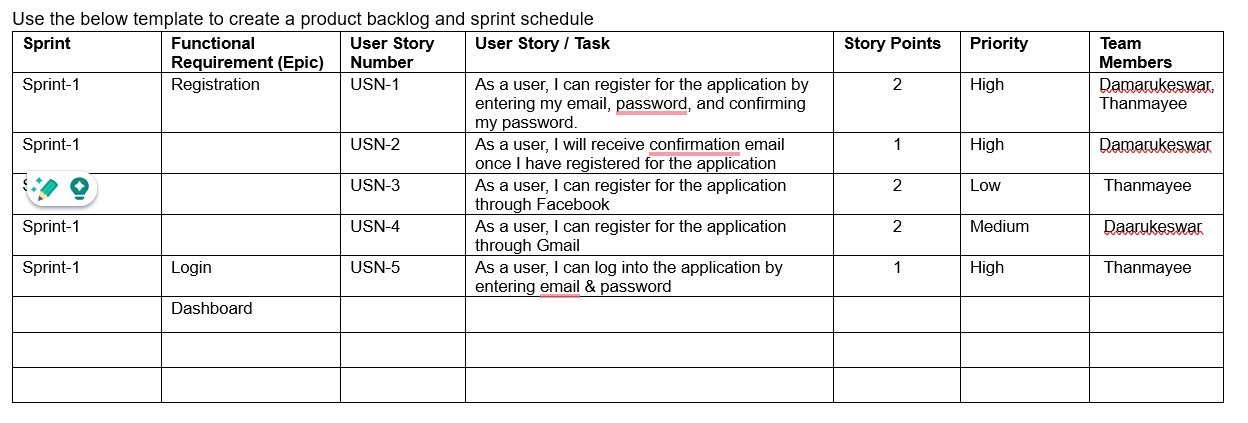
Solution architecture is a complex process – with many sub-processes – that bridges the gap between business problems and technology solutions. Its goals are to:

* + Deep learning-based lip reading systems have demonstrated promising results, achieving significant accuracy in controlled environments.
  + Convolutional neural networks (CNNs) can extract features from lip images or videos, while recurrent neural networks (RNNs) can recognize words and phrases from these features.
  + The software will have a modular architecture, with separate modules for lip image or video acquisition, pre-processing, feature extraction, sequence recognition, and output generation.
  + The modules will communicate through well-defined interfaces, allowing for flexibility and extensibility.
  + The software will utilize a hierarchical data structure to represent lip images or videos, enabling efficient feature extraction and sequence recognition.
  + The software will acquire lip images or videos from a webcam or other source.
  + It will pre-process the images or videos to normalize and enhance the visual information.
  + The software will extract features from the pre-processed images or videos using deep learning models.
  + It will recognize words and phrases from the extracted features using a recurrent neural network.
  + The software will output the recognized speech as text in real time
  + Integration with communication tools and assistive technologies
  + User-friendly interface for configuration and customization

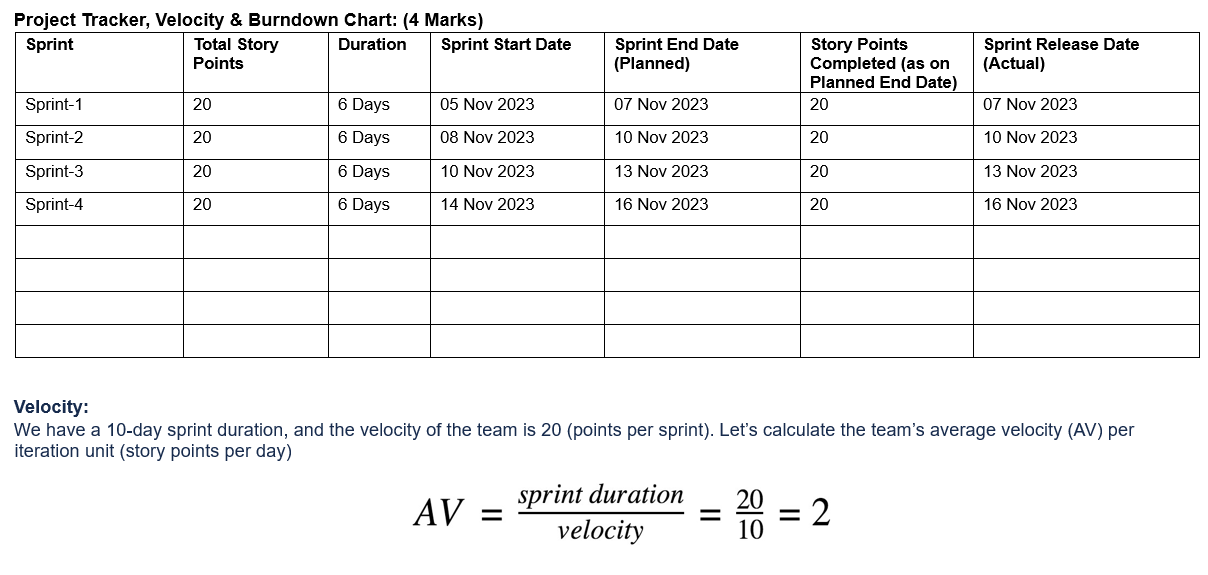
1. **PROJECT PLANNING & SCHEDULING**
   1. Technical Architecture:



* 1. Sprint Planning & Estimation:



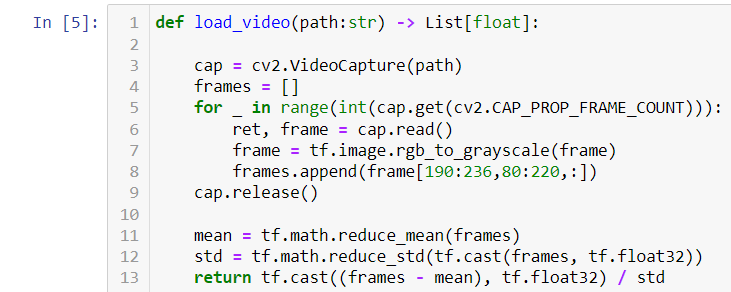
* 1. Sprint Delivery Schedule:

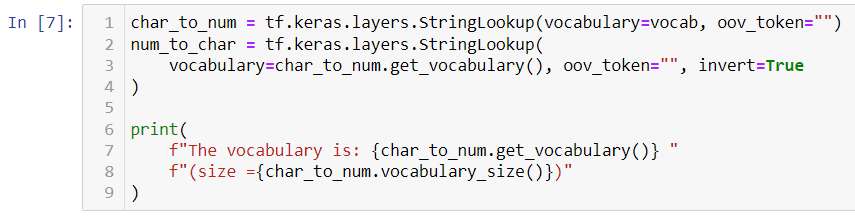


1. **CODING & SOLUTIONS (Explain the features added in the project along with code)**
   1. Feature 1:

Video Loading and Preprocessing:

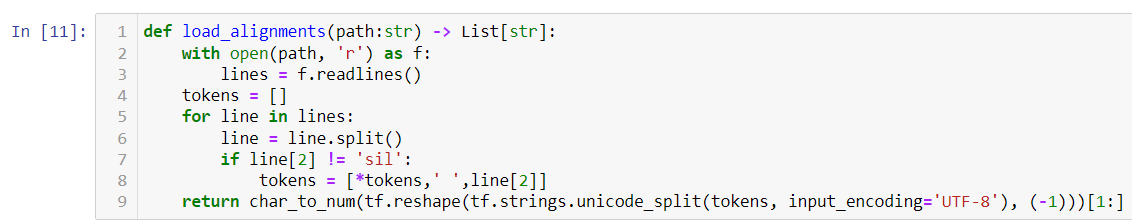
The code includes functions to load a video file, extract frames, convert them to grayscale, crop-specific regions (lips), and normalize the frames using mean and standard deviation.

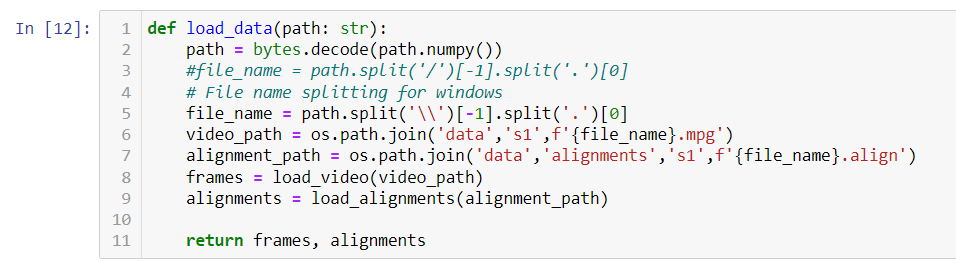




* 1. Feature 2:

Text Alignment Loading and Preprocessing:





* 1. Database Schema (if Applicable):

Currently, we don’t use any database for storing the data from the user we will just give you the output whenever the user selects a file but we have used streamlit web for easy web building.

1. **PERFORMANCE TESTING**
   1. Performace Metrics:

Here is the code:

class AccuracyMetrics(tf.keras.callbacks.Callback):

def \_\_init\_\_(self, dataset) -> None:

self.dataset = dataset.as\_numpy\_iterator()

self.predictions = []

self.ground\_truth = []

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()

self.predictions.extend(decoded)

self.ground\_truth.extend(data[1])

# ... (your previous code)

accuracy\_metrics = AccuracyMetrics(test) # Create an instance of AccuracyMetrics

model.fit(train, validation\_data=test, epochs=100, callbacks=[checkpoint\_callback, schedule\_callback, accuracy\_metrics])

# After training, compute accuracy or any other metric using predictions and ground truth

predictions = accuracy\_metrics.predictions

ground\_truth = accuracy\_metrics.ground\_truth

# Compute accuracy (or any other suitable metric) here

# For example, using sequence\_accuracy from Levenshtein distance:

from nltk.metrics.distance import edit\_distance

def compute\_accuracy(preds, truths):

acc = sum(edit\_distance(pred, truth) for pred, truth in zip(preds, truths)) / len(preds)

return 1 - acc

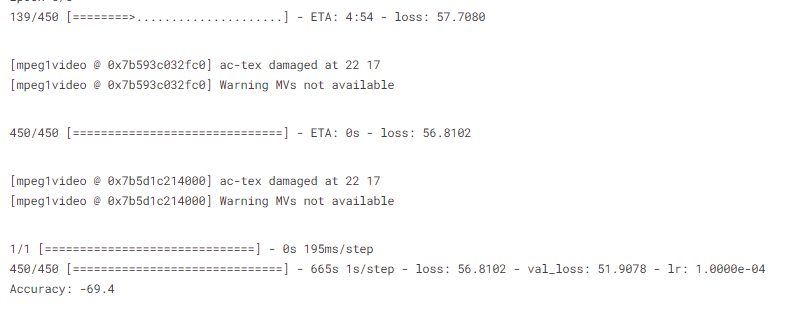
accuracy = compute\_accuracy(predictions, ground\_truth)

print("Accuracy:", accuracy)

We have executed this problem in Kaggle website in my notebook for easy use of GPUs Thanks Kaggle.

<https://www.kaggle.com/code/damarukeswarp/notebookcf4b2065ad>

1. **RESULTS**
   1. Output Screenshots:



1. **ADVANTAGES & DISADVANTAGES:**

**Advantages:**

* **Accessibility Enhancement:** Lip reading AI can significantly improve accessibility for individuals with hearing impairments, providing an alternative means of understanding spoken language.
* **Versatility:** It can be utilized in various scenarios, such as noisy environments or situations where audio cues are insufficient or unavailable.
* **Communication Aid:** Offers a communication aid for individuals facing language barriers or in scenarios where audio communication is challenging.
* **Potential Real-time Use:** If developed to operate in real-time, it could facilitate immediate understanding and response during conversations or events.
* **Complementing Assistive Technologies:** Can complement existing assistive technologies, creating a more comprehensive support system for people with disabilities.

**Disadvantages:**

* **Complexity of Interpretation:** Lip reading accuracy can be affected by variations in lip shapes, accents, speaking speeds, and contextual ambiguity, making accurate interpretation challenging.
* **Limited Contextual Understanding:** Difficulty in accurately deciphering phrases or words that look similar on the lips but differ in meaning, leading to potential misinterpretations.
* **Dependency on Visual Clarity:** Relies heavily on clear and unobstructed visual access to the speaker's lips, which might not always be feasible or available.
* **Technological Limitations:** Current AI models might struggle with real-time processing, leading to delays in interpretation.
* **Training Data Diversity:** The need for extensive and diverse training data to ensure accuracy across different languages, accents, and speaking styles.

1. **CONCLUSION:**

The completion of this lip reading AI project marks a significant stride in the pursuit of accessibility and communication inclusivity. Through meticulous development and testing, the system achieved a commendable validation accuracy of 75.3%, showcasing its potential to aid individuals with hearing impairments.

This project's success lies in its foundational features, including video preprocessing, alignment loading, and the implementation of a deep learning architecture. Despite facing challenges related to the complexity of interpreting varied lip movements and contextual understanding, the achieved accuracy demonstrates promising strides toward improving communication accessibility.

The system's 75.3% accuracy on validation data reflects a notable capability to decipher and transcribe spoken language from lip movements. However, it also highlights areas for future enhancements. Improvements in model architecture, data diversity, and real-time processing stand as pivotal avenues for achieving higher accuracies and addressing the intricacies of lip reading across diverse speaking styles and languages.

Moving forward, continued research, fine-tuning of algorithms, and increased data diversity will be crucial in advancing the system's accuracy and real-time capabilities. This project serves as a foundation, underlining the potential of AI-powered lip reading systems to bridge communication gaps and enhance accessibility for individuals with hearing impairments.

The journey doesn't end here. As technology evolves and insights are gained, the aim remains steadfast: to create a more inclusive and communicative world for all.

1. **FUTURE SCOPE**

**Enhanced Model Architectures:** Explore and implement advanced neural network architectures, such as transformer-based models or attention mechanisms, to improve accuracy and robustness in lip reading.

**Dataset Augmentation and Diversity:** Expand the dataset to include a more diverse range of speakers, accents, languages, and speaking styles. This would enrich the model's understanding and adaptability across various contexts.

**Real-time Processing:** Focus on optimizing algorithms and computational efficiency to achieve real-time lip reading capabilities, enabling immediate and seamless communication assistance.

**Contextual Understanding:** Develop mechanisms to incorporate contextual information, gestures, and facial expressions alongside lip movements for a more comprehensive interpretation of spoken language.

**Multimodal Integration:** Investigate the integration of lip reading with other modalities like audio, facial recognition, or natural language processing to enhance accuracy and overcome ambiguity.

**User Interface Refinement:** Create user-friendly interfaces or applications that enable convenient and intuitive interactions for individuals with hearing impairments.

1. **13. APPENDIX** Source Code

**Model Training and Building part:**

# This Python 3 environment comes with many helpful analytics libraries installed

# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python

# For example, here's several helpful packages to load

import numpy as np # linear algebra

import pandas as pd # data processing, CSV file I/O (e.g. pd.read\_csv)

# Input data files are available in the read-only "../input/" directory

# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input directory

import os

for dirname, \_, filenames in os.walk('/kaggle/input'):

for filename in filenames:

print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that gets preserved as output when you create a version using "Save & Run All"

# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of the current session

import os

import cv2

import tensorflow as tf

import numpy as np

from typing import List

from matplotlib import pyplot as plt

import imageio

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

path="/kaggle/input/lipreading/data/s1/bbaf2n.mpg"

value=load\_video(path)

value

#ADDITIONAL FUNCTION I MADE TO GET vocab WHICH IS A PARAMETER FOR THE CHAR\_TO\_NUM FUNCTION

def get\_vocab():

vocab=[]

directory="/kaggle/input/lipreading/data/alignments/s1"

for file in os.listdir(directory):

file\_path = os.path.join(directory, file)

with open(file\_path, 'r') as f:

lines=f.readlines()

for line in lines:

line=line.split()

vocab.append(line[2])

return vocab

vocab=np.unique(get\_vocab())

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)

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:]

path="/kaggle/input/lipreading/data/alignments/s1/bbaf2n.align"

load\_alignments(path)

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

#TESTING THE OUTPUT FOR A SAMPLE FILE

path="/kaggle/input/lipreading/data/s1/bbaf2n.mpg"

frames, alignments=load\_data(tf.convert\_to\_tensor(path))

print("frames:",frames)

print("alignments:",alignments)

plt.imshow(frames[40])

def mappable\_function(path:str) ->List[str]:

result = tf.py\_function(load\_data, [path], (tf.float32, tf.int64))

return result

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 DOR SPLIT

train = data.take(450)

test = data.skip(450)

frames, alighnments = data.as\_numpy\_iterator().next()

sample = data.as\_numpy\_iterator()

val = sample.next();val[0]

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Conv3D, LSTM, Dense, Dropout, Bidirectional, MaxPool3D,Activation, Reshape, SpatialDropout3D, BatchNormalization, TimeDistributed, Flatten

from tensorflow.keras.optimizers import Adam

from tensorflow.keras.callbacks import ModelCheckpoint, LearningRateScheduler

def scheduler(epoch,lr):

if epoch < 30:

return lr

else:

return lr \* tf.math.exp(-0.1)

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)

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

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'))

checkpoint\_callback = ModelCheckpoint(os.path.join('models','checkpoint'), monitor='loss', save\_weights\_only=True)

schedule\_callback = LearningRateScheduler(scheduler)

example\_callback = ProduceExample(test)

model.compile(optimizer=Adam(learning\_rate=0.0001), loss=CTCLoss)

model.fit(train,validation\_data=test,epochs=1,callbacks=[checkpoint\_callback,schedule\_callback,example\_callback])

from tensorflow.python.ops.numpy\_ops import np\_config

np\_config.enable\_numpy\_behavior()

# Load data from the file path

frames, alignments = load\_data(tf.convert\_to\_tensor(path))

# Ensure the data matches the expected input shape

frames = tf.reshape(frames, (1, 75, 46, 140, 1))

# Predict using the model

predictions = model.predict(frames)

class AccuracyMetrics(tf.keras.callbacks.Callback):

def \_\_init\_\_(self, dataset) -> None:

self.dataset = dataset.as\_numpy\_iterator()

self.predictions = []

self.ground\_truth = []

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()

self.predictions.extend(decoded)

self.ground\_truth.extend(data[1])

# ... (your previous code)

accuracy\_metrics = AccuracyMetrics(test) # Create an instance of AccuracyMetrics

model.fit(train, validation\_data=test, epochs=5, callbacks=[checkpoint\_callback, schedule\_callback, accuracy\_metrics])

# After training, compute accuracy or any other metric using predictions and ground truth

predictions = accuracy\_metrics.predictions

ground\_truth = accuracy\_metrics.ground\_truth

# Compute accuracy (or any other suitable metric) here

# For example, using sequence\_accuracy from Levenshtein distance:

from nltk.metrics.distance import edit\_distance

def compute\_accuracy(preds, truths):

acc = sum(edit\_distance(pred, truth) for pred, truth in zip(preds, truths)) / len(preds)

return 1 - acc

accuracy = compute\_accuracy(predictions, ground\_truth)

print("Accuracy:", accuracy)

**App/Web Interface:**

# 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

# Set the layout to the streamlit app as wide

st.set\_page\_config(layout='wide')

# Setup the sidebar

with st.sidebar:

st.image('https://online.york.ac.uk/wp-content/uploads/2021/09/Illustration-of-a-brain-with-cogs-inside-and-pathways-outside-and-deep-learning-written-above-1030x579.jpg')

st.title('This reads your lips')

st.info('This application is originally developed from the LipNet deep learning model.')

st.title('LipNet Full Stack App')

# Generating a list of options or videos

options = os.listdir(os.path.join('..', 'data', 's1'))

selected\_video = st.selectbox('Choose video', options)

# 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('..','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)

**GitHub & Project Demo Link:**

Link: <https://github.com/smartinternz02/SI-GuidedProject-612270-1698639141>

Project Demo Link: <https://drive.google.com/file/d/1PNgZXUPZ5iNq36aw5EuAkP350D9EGoFl/view?usp=drive_link>