Autonomous IoT-Enabled Hazard Detection and Communication System for Deaf Drivers.

24-25J-132

SLIIT
FACULTY OF COMPUTING

Our Team

Dr.Kapila Dissanayaka
(Supervisor)

Ms.Ishara Weerathunga
(Co-Supervisor)

Rathnayake R.M.S.N
IT21388316

Fernando W.T.R.P
IT21278280

Thathsara S.M.K
IT21219566

Iroshan G.H.M
IT21229084

Introduction

Deaf drivers face challenges due to the inability to perceive auditory cues like emergency sirens and car honking, leading to delayed responses and increased accident risk. Current vehicle safety systems often overlook the needs of deaf drivers, resulting in a lack of reliable alerts and communication methods during critical situations.

This research aims to develop an "Autonomous IoT Enabled Hazard Detection and Communication System for Deaf Drivers" which will:

- Provide real-time visual and haptic alerts.
- Use advanced sensor technology, machine learning algorithms, and a mobile application.
- Bridge the communication gap.
- Enhance driving safety for deaf drivers.

Research Question

- **How can an autonomous IoT-enabled system effectively detect emergency vehicle sirens and other critical sounds in real-time for deaf drivers?**
- **What types of sensor technologies and machine learning algorithms are most effective for identifying and processing auditory cues on the road?**
- **How can visual and haptic feedback mechanisms be optimized to ensure deaf drivers receive timely and clear notifications?**
- **What are the challenges in integrating such a system with existing vehicle infrastructure and how can they be overcome?**
- **How can a mobile application be designed to support real-time hazard detection and provide a seamless communication interface for deaf drivers?**
- **How can the system ensure reliable performance in diverse driving environments and conditions?**

OBJECTIVES

MAIN OBJECTIVE

The main objective of this research is to develop a robust and accessible hazard detection and communication system specifically designed for deaf drivers.

SUB OBJECTIVE

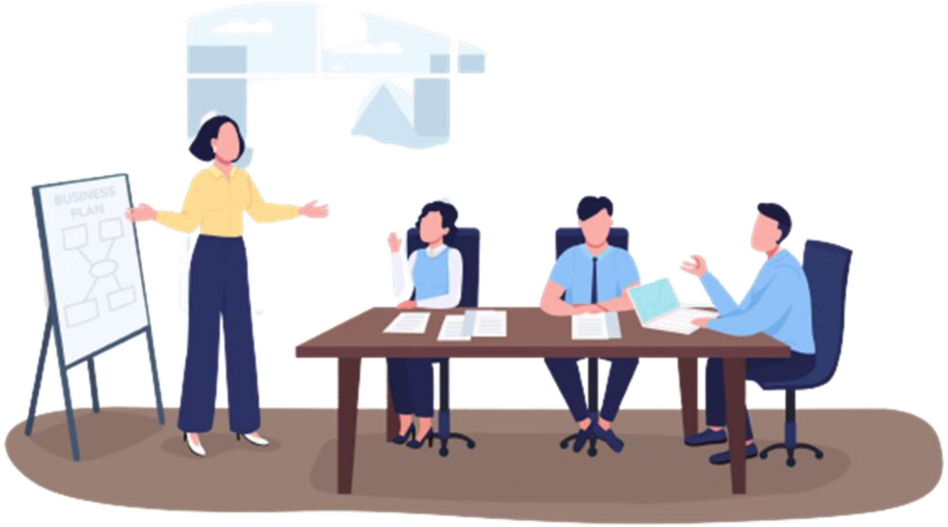

Emergency Vehicle Detection and Notification System


Continued Vehicle Horn Detection and Alert System


Comprehensive Driver Behavior Monitoring and Reporting System


Integrated Emergency Support System for Deaf Drivers

Commercialization



Market Potential:

Partnerships with automotive manufacturers.

Targeted marketing towards the deaf community.

Integration with existing vehicle safety systems.

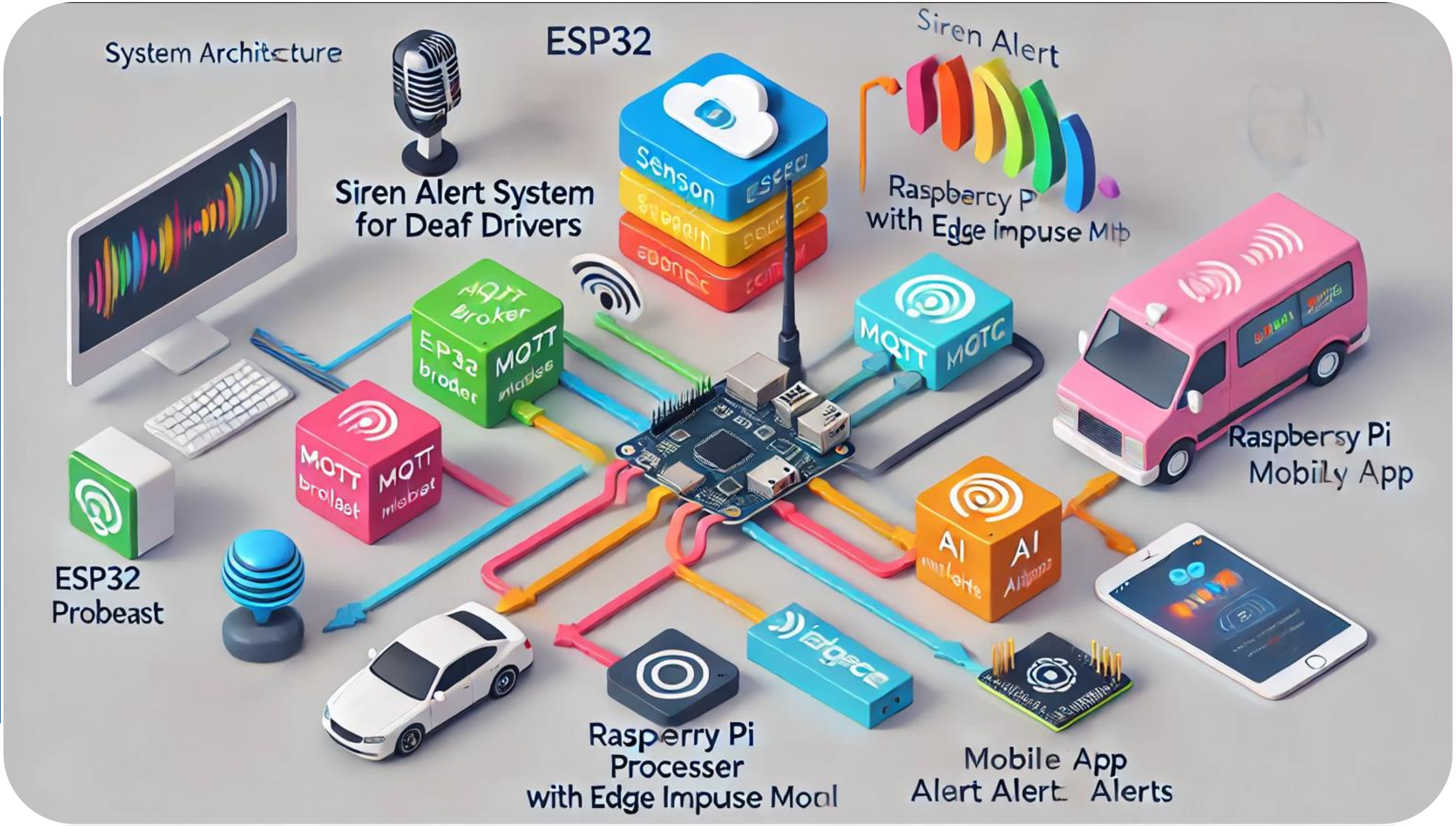
Potential for government or organizational funding support.

Rathnayake R.M.S.N
IT21388316

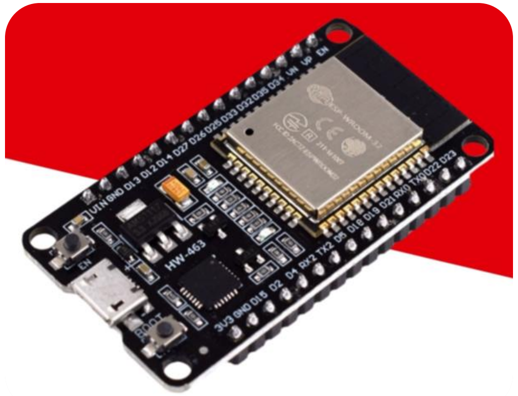
SOFTWARE ENGINEERING

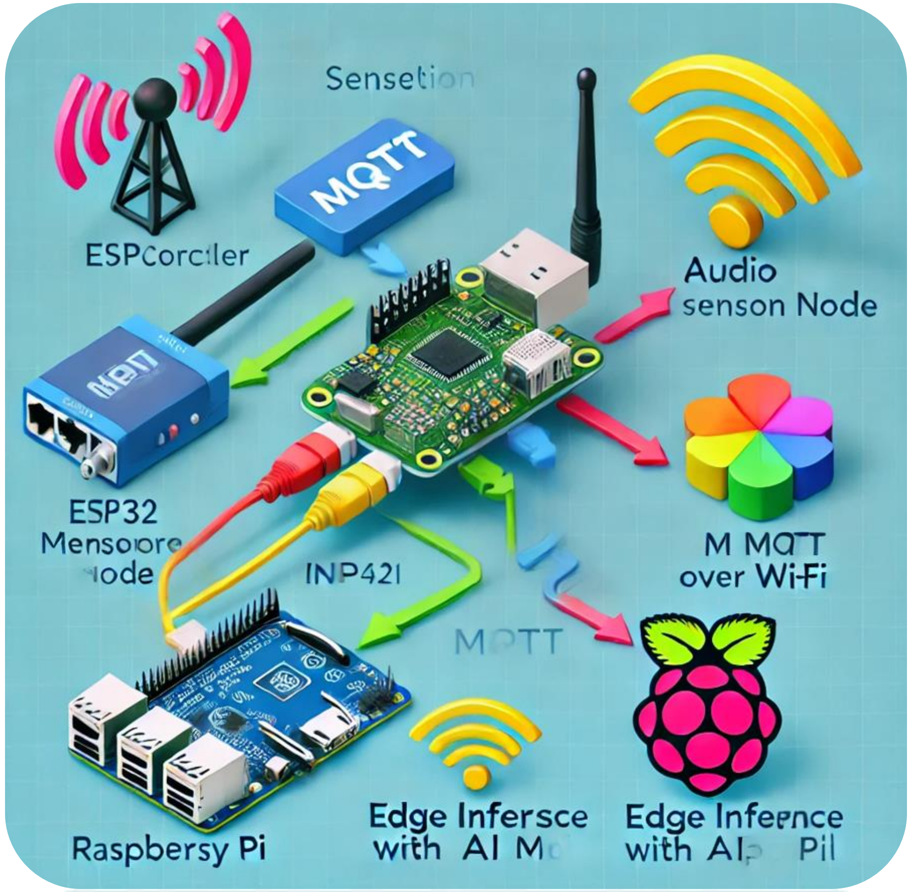

FIRE
AMBULANCE
AMBULANCE
POLICE
Emergency Vehicle Detection and Notification System

System Overview Diagram

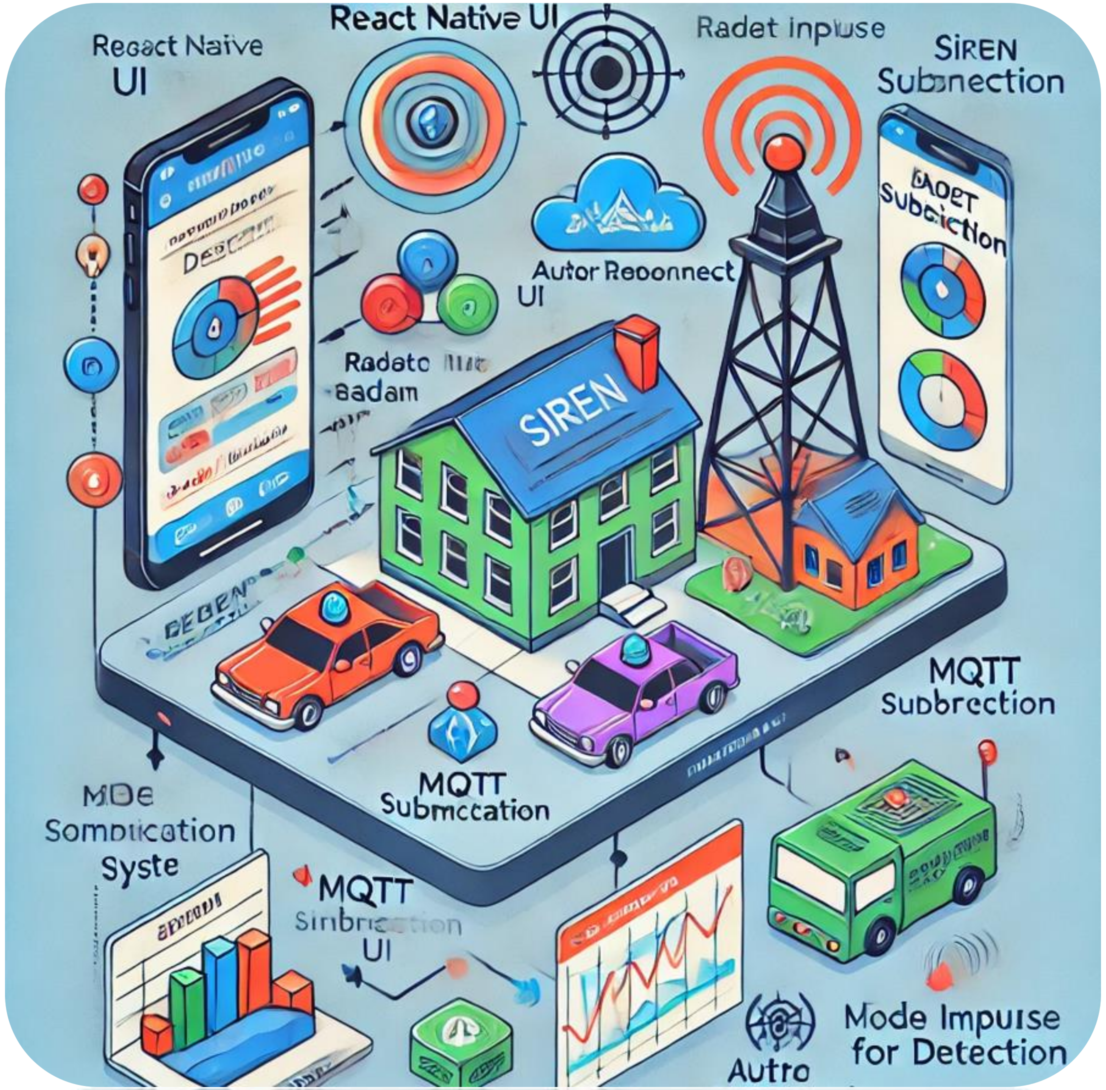


Hardware Architecture

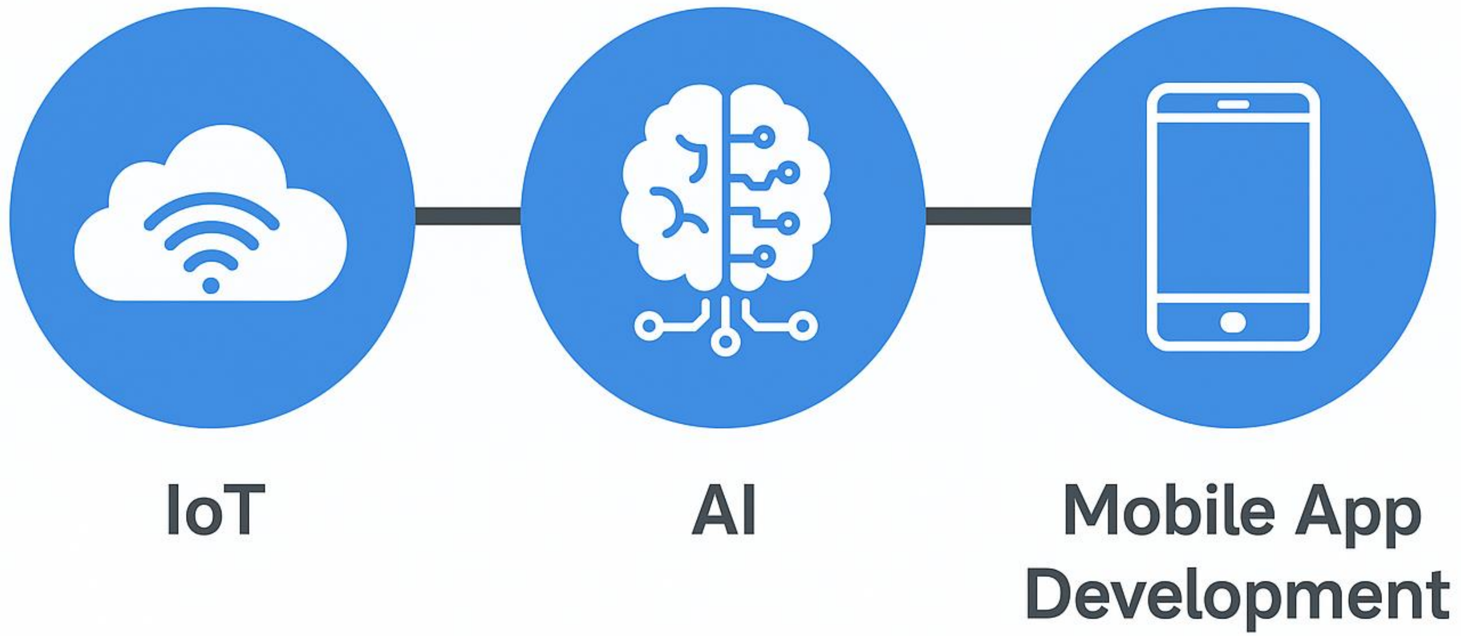




Software Architecture



Covered Areas



Sources of Audio Data

- **Pre-existing Datasets**

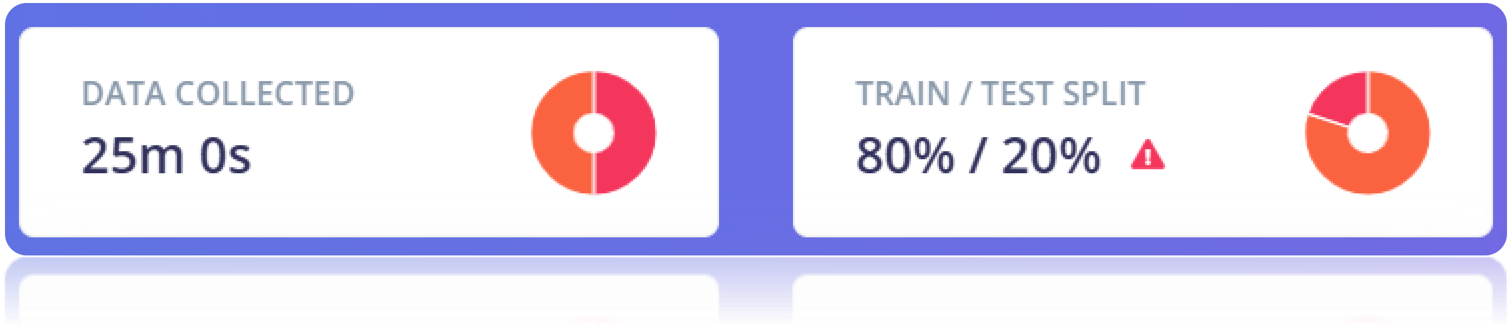
  UrbanSound8K: Contains environmental sound clips, including sirens.

- **Manual Collection**

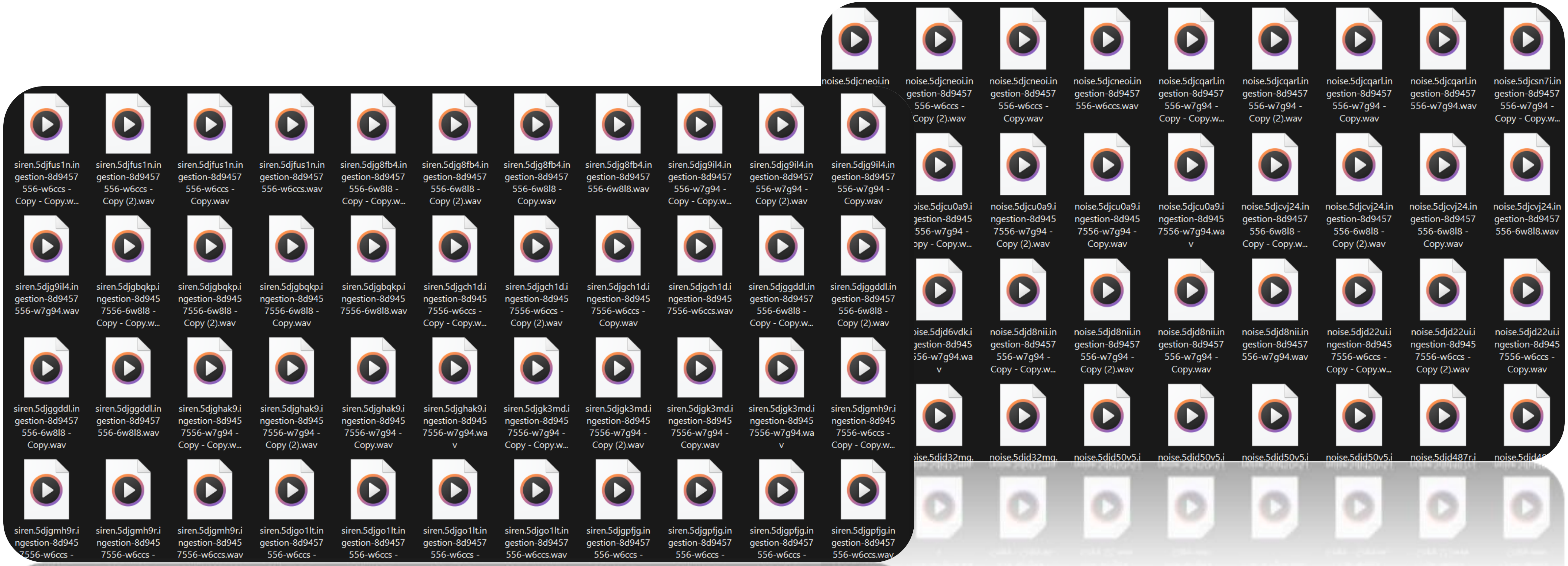
  Emergency vehicles like ambulances, police cars, and fire trucks.
  Different scenarios (urban, rural, heavy traffic).

- **Publicly Available Recordings**

  Extract sounds from YouTube videos.

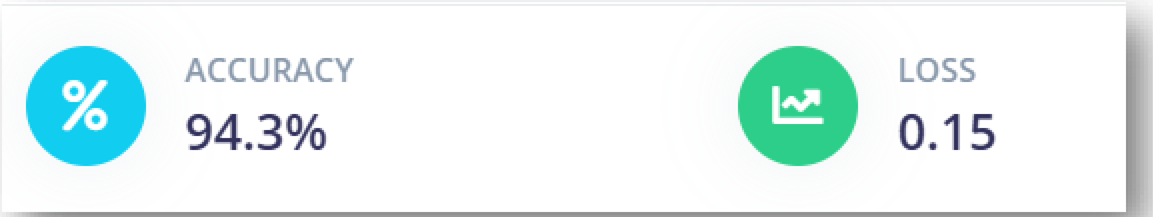


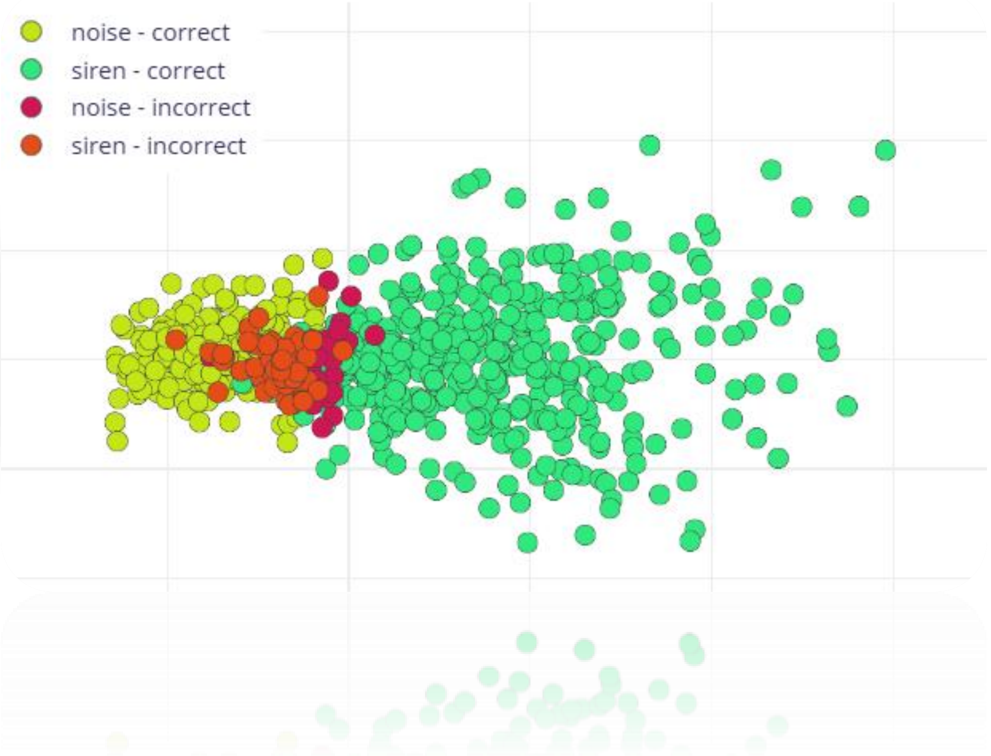
Data Acquisition



Deep Learning Model

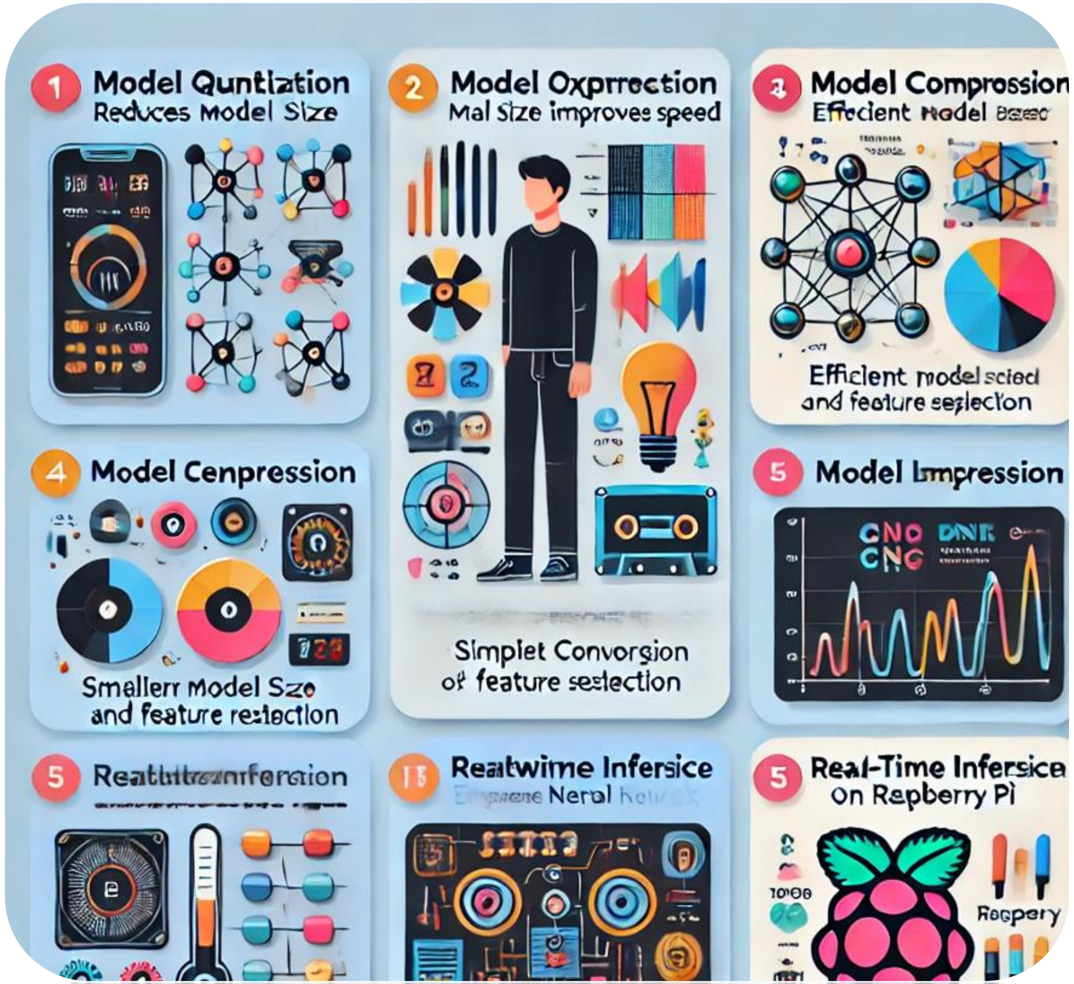
- **Architecture :** Convolutional Neural Network (CNN) architecture
- **Number of training cycles(epochs)** : 300
- **Training processor :** CPU
- **Add Data augmentation**



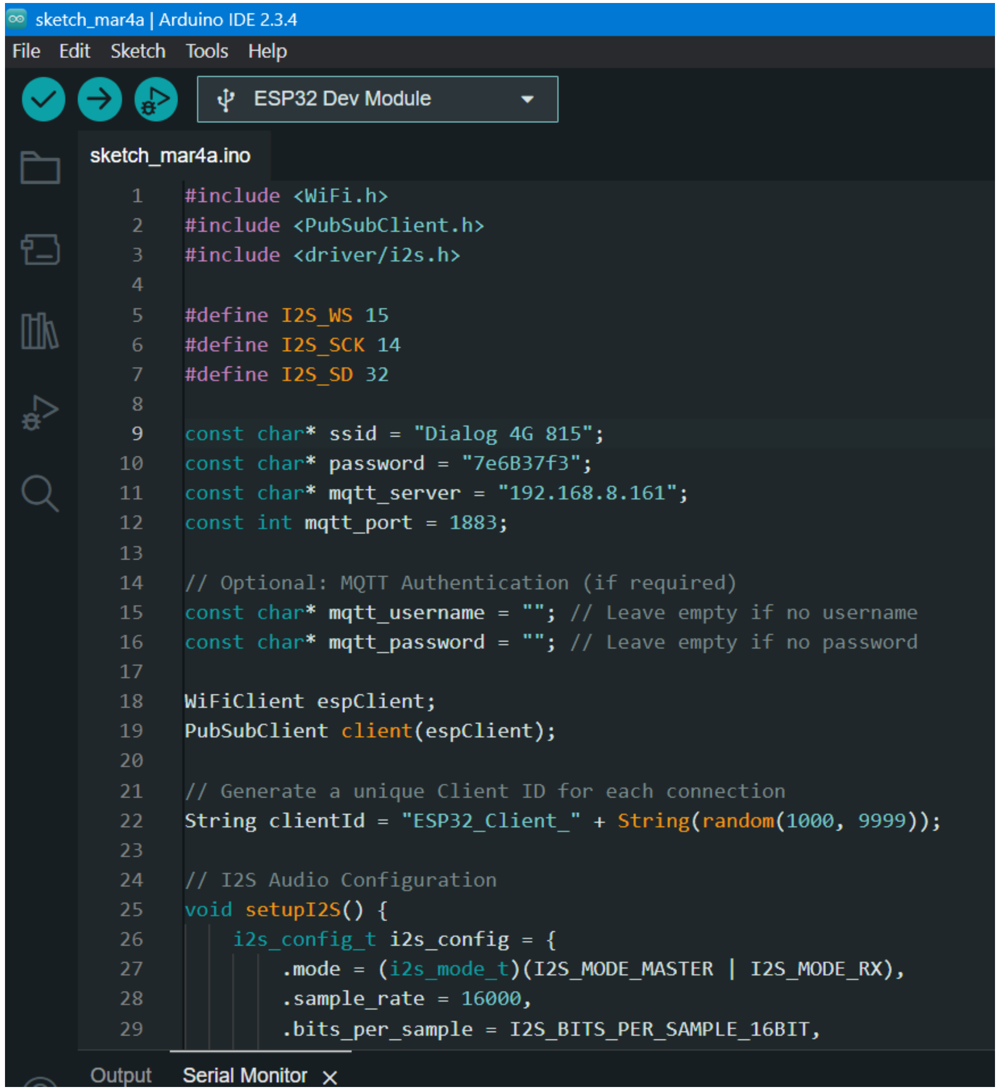


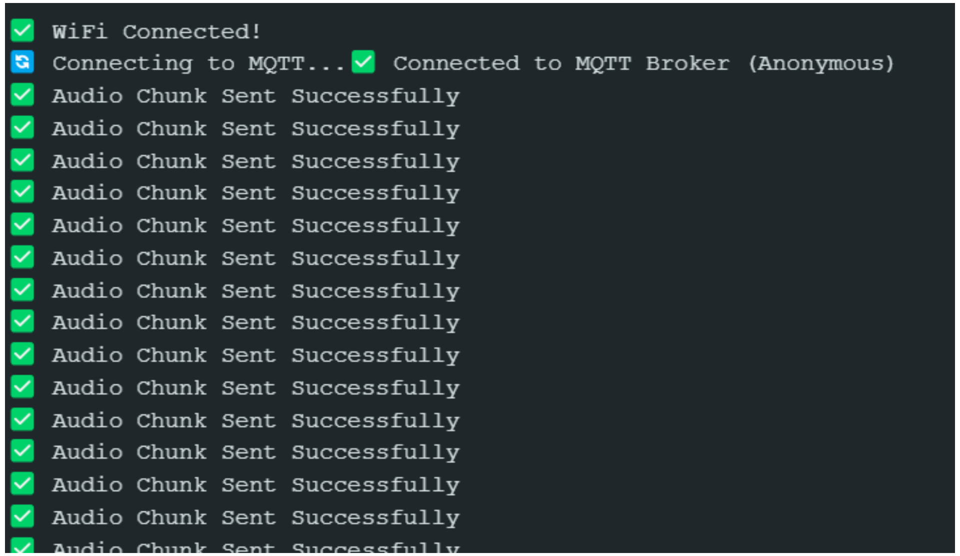
Model Optimization Techniques

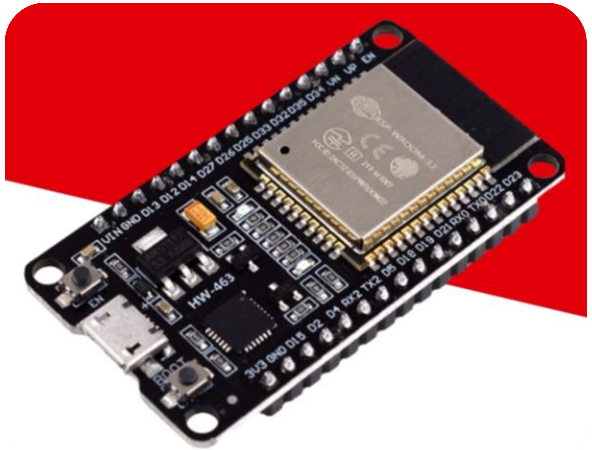
- **Quantization** — Reduces size, improves speed
- **used dropout** — to prevent overfitting
- **MFCC Features** — Efficient audio representation



ESP32 Nodemcu Board Processing

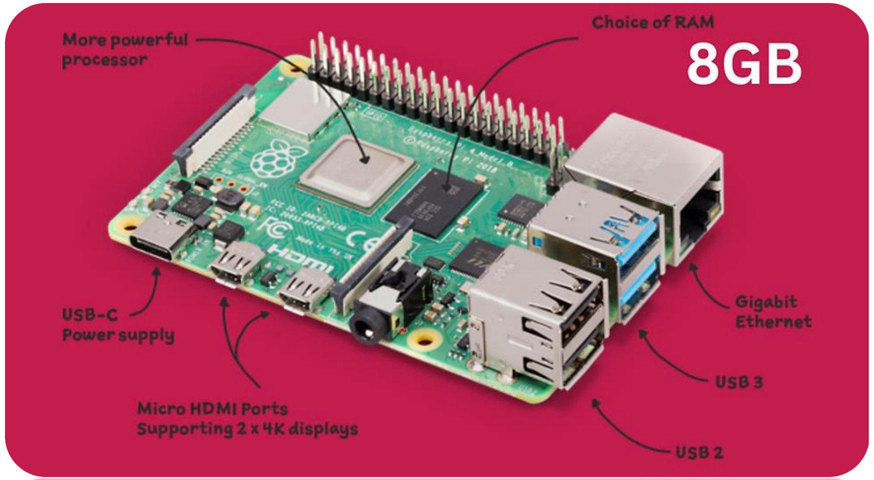


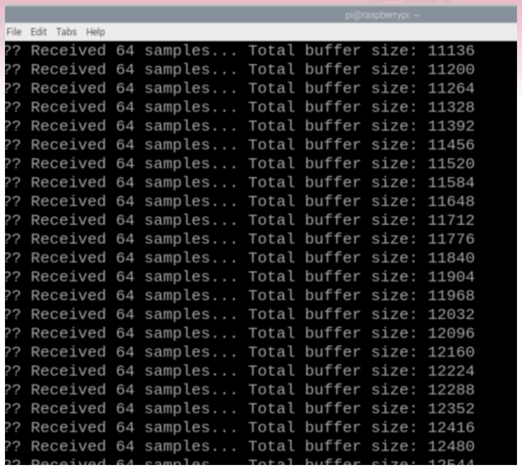




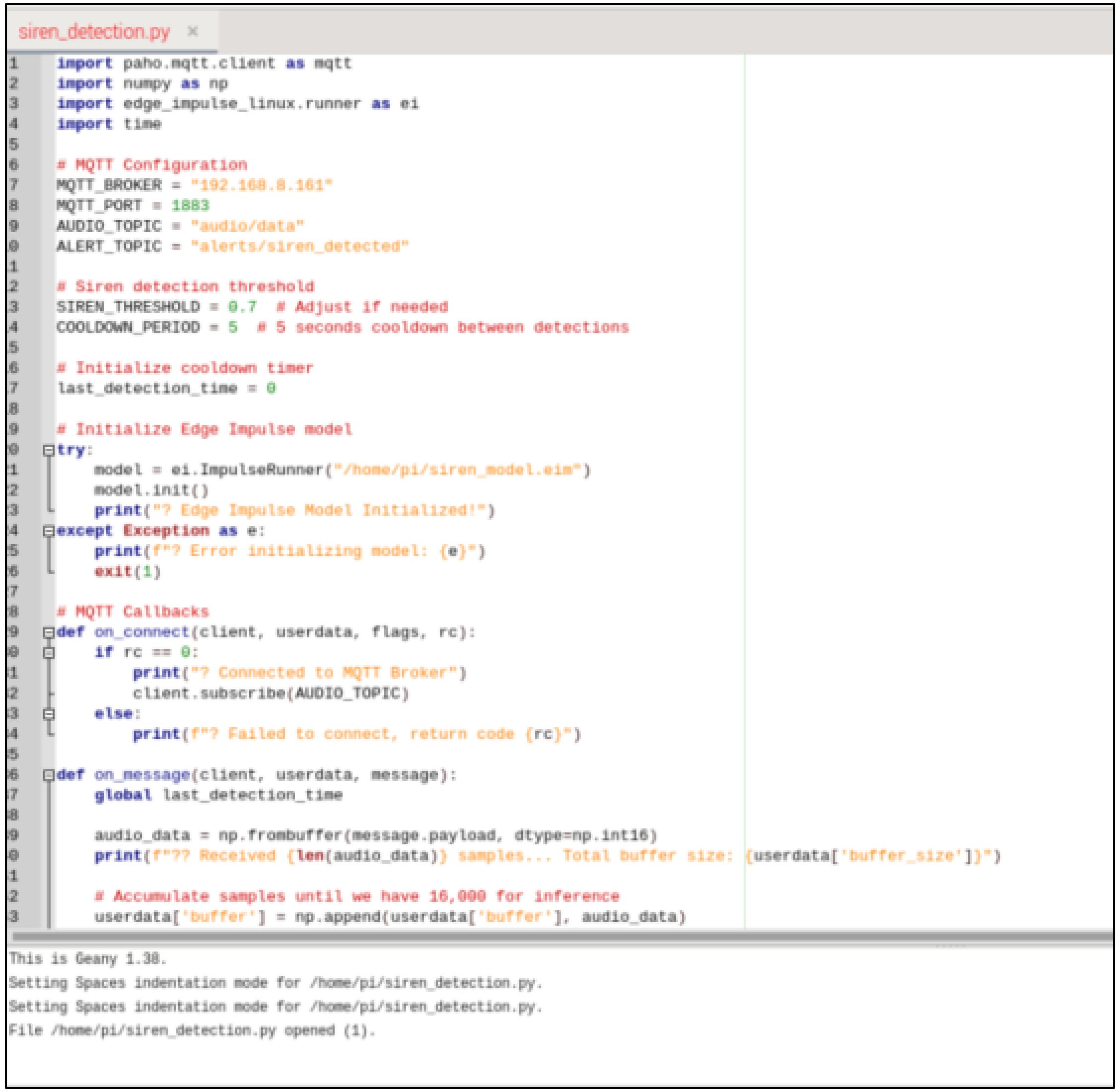
Raspberry pi Board Processing

Siren_detection.py





```
?? Received 64 samples... Total buffer size: 11136
?? Received 64 samples... Total buffer size: 11200
?? Received 64 samples... Total buffer size: 11264
?? Received 64 samples... Total buffer size: 11328
?? Received 64 samples... Total buffer size: 11392
?? Received 64 samples... Total buffer size: 11456
?? Received 64 samples... Total buffer size: 11520
?? Received 64 samples... Total buffer size: 11584
?? Received 64 samples... Total buffer size: 11648
?? Received 64 samples... Total buffer size: 11712
?? Received 64 samples... Total buffer size: 11776
?? Received 64 samples... Total buffer size: 11840
?? Received 64 samples... Total buffer size: 11904
?? Received 64 samples... Total buffer size: 11968
?? Received 64 samples... Total buffer size: 12032
?? Received 64 samples... Total buffer size: 12096
?? Received 64 samples... Total buffer size: 12160
?? Received 64 samples... Total buffer size: 12224
?? Received 64 samples... Total buffer size: 12288
?? Received 64 samples... Total buffer size: 12352
?? Received 64 samples... Total buffer size: 12416
?? Received 64 samples... Total buffer size: 12480
```

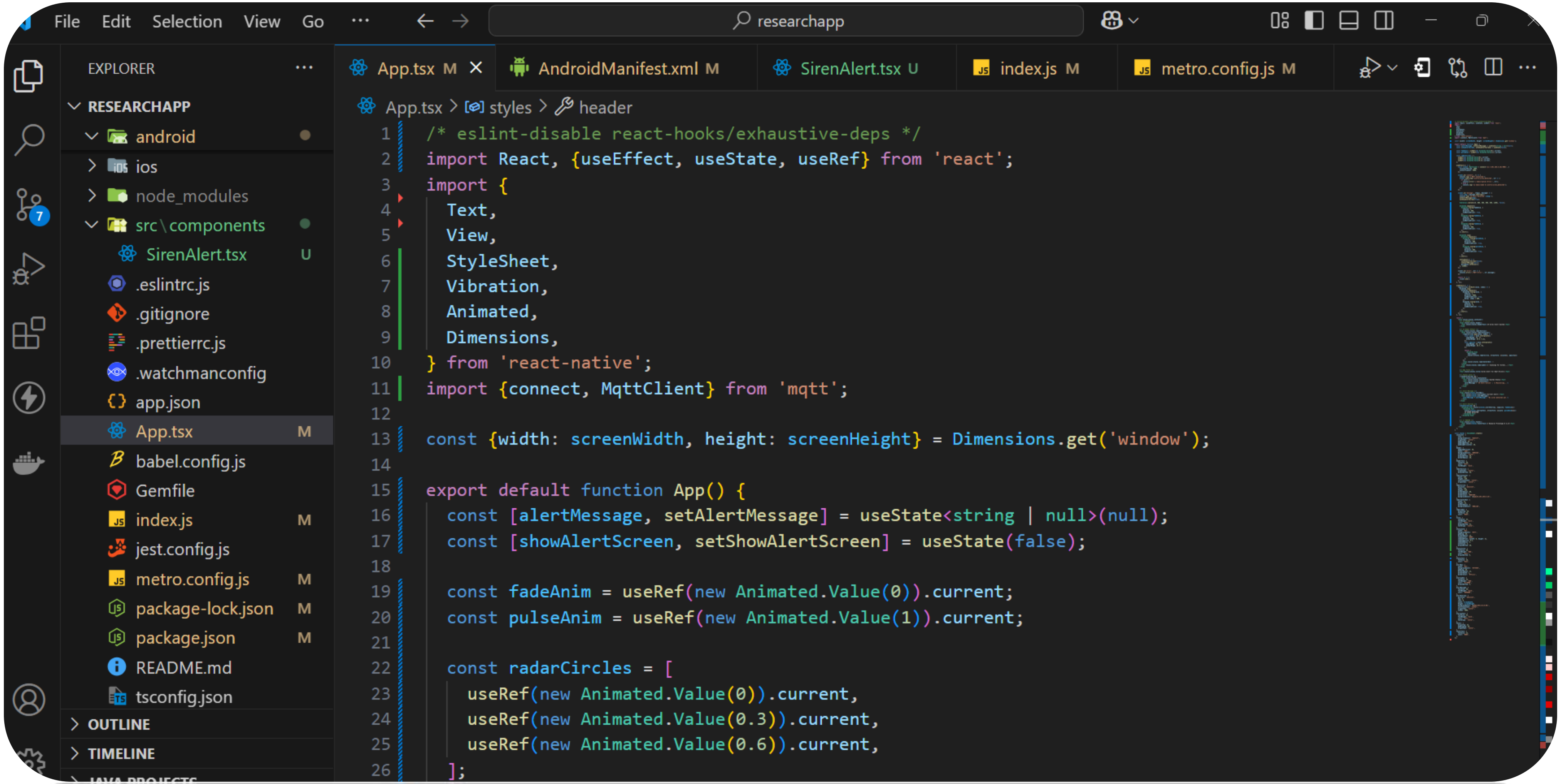


siren_detection.py

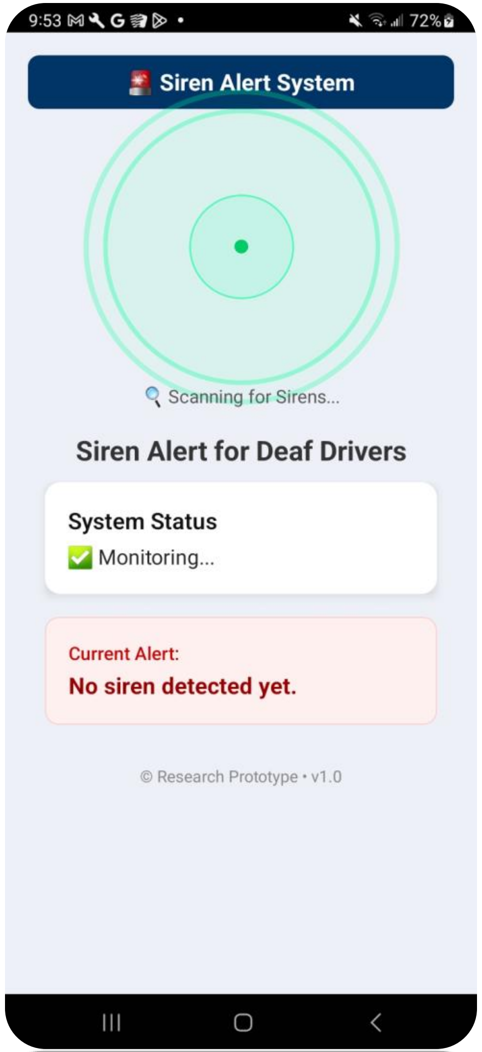
```python
import paho.mqtt.client as mqtt
import numpy as np
import edge_impulse_linux.runner as ei
import time

# MQTT Configuration
MQTT_BROKER = "192.168.8.161"
MQTT_PORT = 1883
AUDIO_TOPIC = "audio/data"
ALERT_TOPIC = "alerts/siren_detected"

# Siren detection threshold
SIREN_THRESHOLD = 0.7  # Adjust if needed
COOLDOWN_PERIOD = 5  # 5 seconds cooldown between detections

# Initialize cooldown timer
last_detection_time = 0

# Initialize Edge Impulse model
try:
    model = ei.ImpulseRunner("/home/pi/siren_model.eim")
    model.init()
    print("? Edge Impulse Model Initialized!")
except Exception as e:
    print(f"? Error initializing model: {e}")
    exit(1)

# MQTT Callbacks
def on_connect(client, userdata, flags, rc):
    if rc == 0:
        print("? Connected to MQTT Broker")
        client.subscribe(AUDIO_TOPIC)
    else:
        print(f"? Failed to connect, return code {rc}")

def on_message(client, userdata, message):
    global last_detection_time

    audio_data = np.frombuffer(message.payload, dtype=np.int16)
    print(f"?? Received {len(audio_data)} samples... Total buffer size: {userdata['buffer_size']}")

    # Accumulate samples until we have 16,000 for inference
    userdata['buffer'] = np.append(userdata['buffer'], audio_data)
```

```
This is Geany 1.38.
Setting Spaces indentation mode for /home/pi/siren_detection.py.
Setting Spaces indentation mode for /home/pi/siren_detection.py.
File /home/pi/siren_detection.py opened (1).
```

React Native App

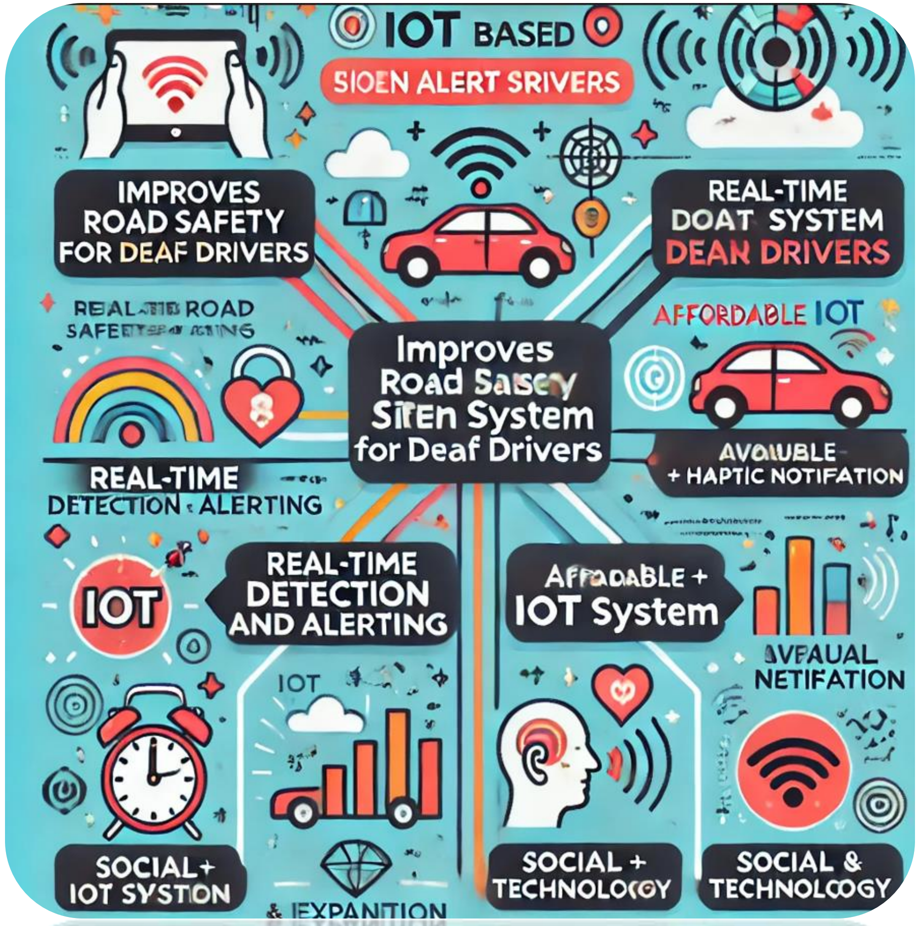


Notification System



Impact & Benefits

- ❑ Improves Road Safety for Deaf Drivers
- ❑ Real-Time Detection and Alerting
- ❑ Affordable IoT-based System
- ❑ Scalable and Expandable
- ❑ Visual + Haptic notification
- ❑ Social and Technological Impact



References

1. T. Harshitha and T. Abhigna, "Emergency Vehicle Sound Detection Systems in Traffic Congestion," Ramachandrapuram, Telangana, 502032, and Jeedimetla, Quthbullapur, Telangana, 500055.
2. J. Smith, R. Brown, and C. White, "Machine Learning for Sound Pattern Recognition," International Journal of Artificial Intelligence, vol. 36, no. 2, pp. 89-101, Feb. 2021.
3. National Safety Council, "Road Safety and Deaf Drivers," National Safety Council Report, 2020. [Online]. Available: https://www.nsc.org
4. Tech Innovators, "Advancements in MEMS Microphone Technology," Tech Innovators Report, 2019. [Online]. Available: https://www.techinnovators.com
5. National Association of the Deaf, "Deaf and Hard of Hearing Drivers," National Association of the Deaf, 2020. [Online]. Available: https://www.nad.org
6. TensorFlow, "TensorFlow: An end-to-end open source machine learning platform," 2023. [Online]. Available: https://www.tensorflow.org

Fernando W.T.R.P

IT21278280

Software Engineering

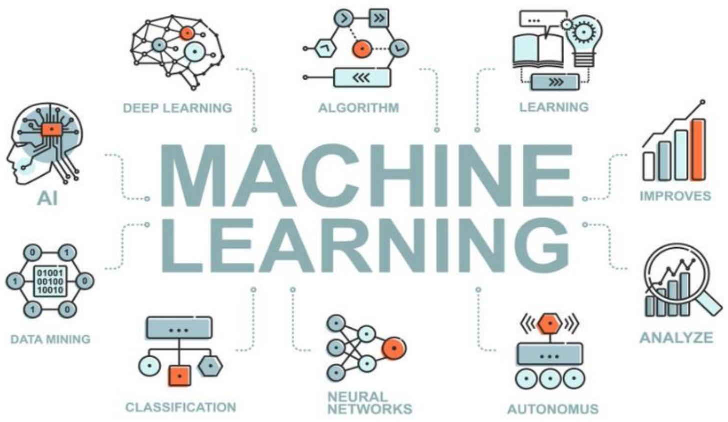
Continued Vehicle Horn Detection and Alert System

DeafConnect

Introduction

- Deaf drivers cannot hear vehicle horns, risking their safety.

- Existing systems don't help deaf drivers effectively.



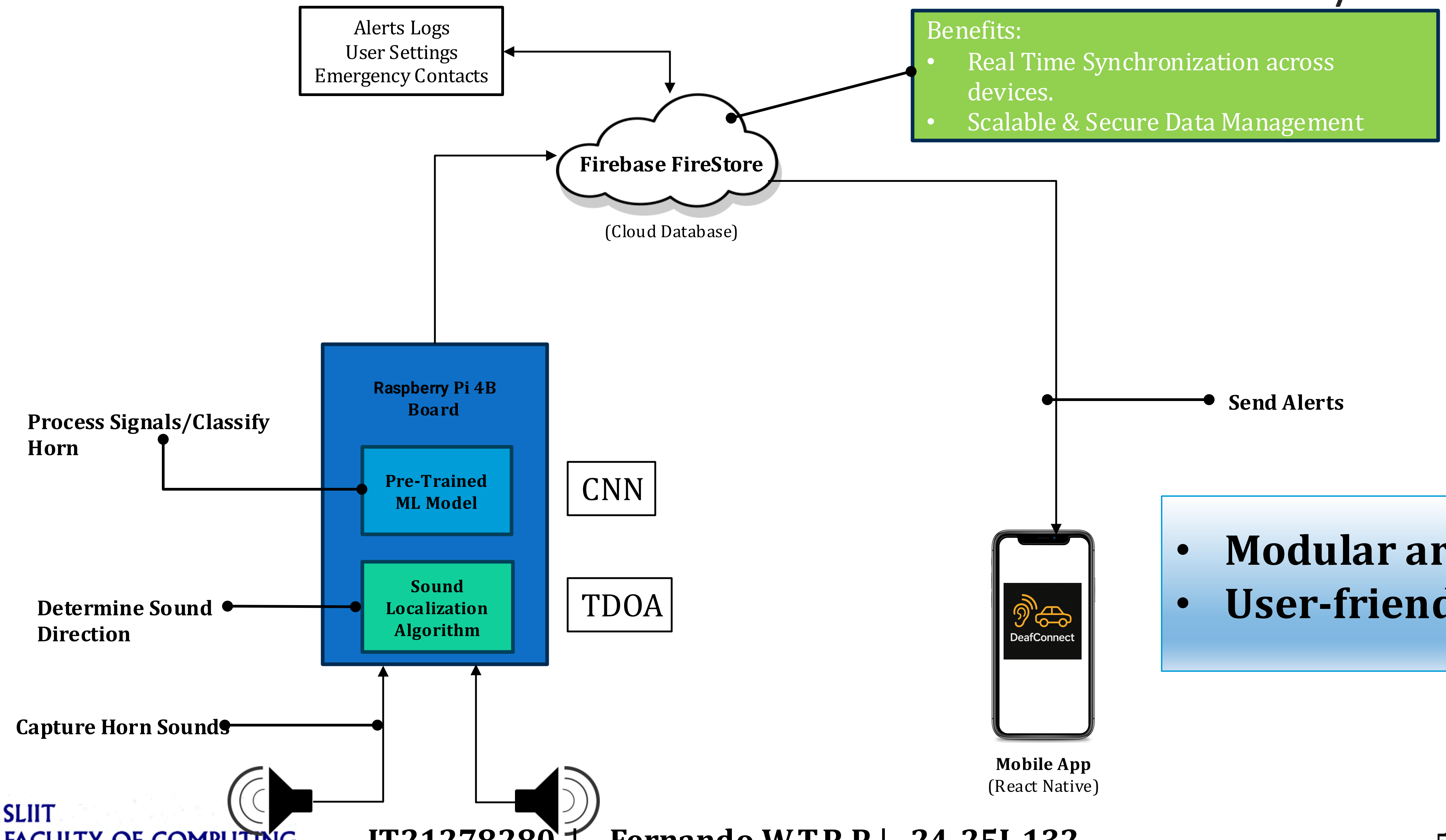
Research Question

- How can we detect and classify horn sounds accurately?
- What machine learning methods can separate horn sounds from background noise?
- How can we determine the direction of the horn sound using microphones?
- What's the best way to alert deaf drivers using visual & haptic feedback?

Research Gap and Novelty

Research	Real-Time Detection	Localization	Noise Robustness	Deaf Driver Focus	Year	Key Limitation
Beritelli	✅	❌	❌	❌	2021	No directional awareness
Sharma	❌	❌	✅	❌	2024	Predictive focus only
Zhao	✅	✅	❌	❌	2023	Lab-Only testing
My System	✅	✅	✅	✅	2025	Comprehensive Solution

Methodology
System Diagram



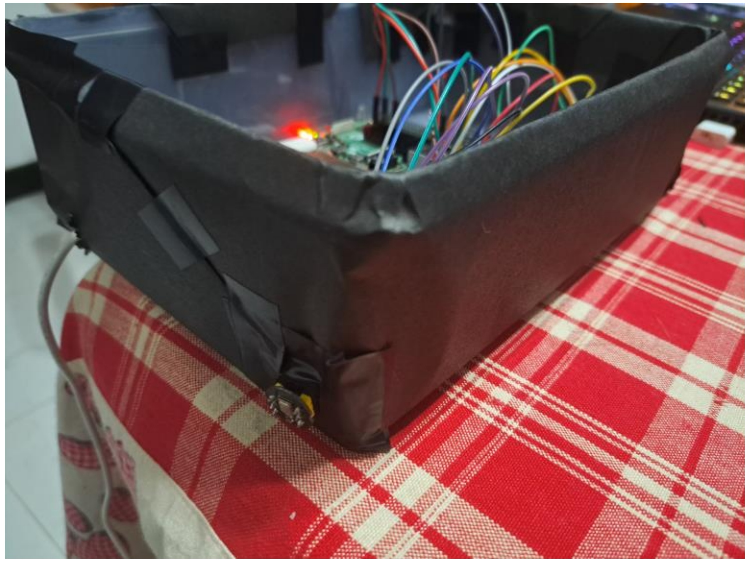
- **Modular architecture**
- **User-friendly App**

Specific Objective

Develop a real-time horn detection system for deaf drivers

Sub Objective

1 **Set up Microphones to Capture Horn Sounds**



100%

Sub Objective

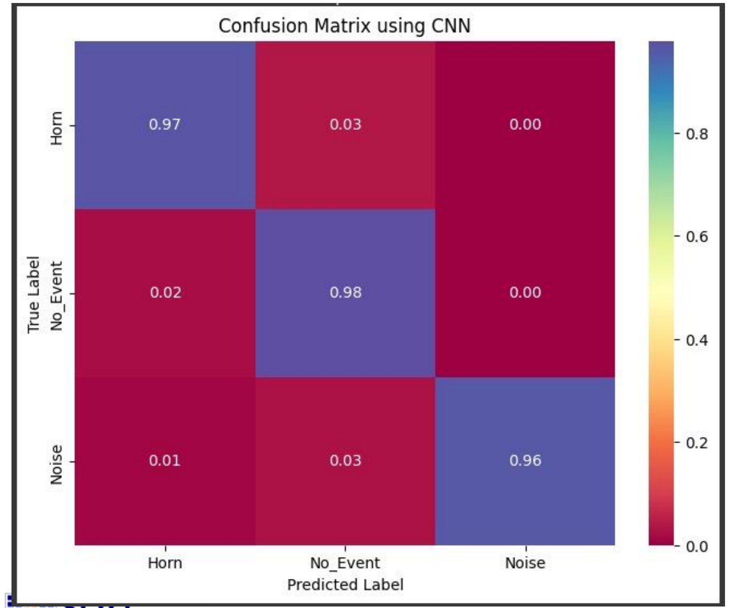
2

Use Machine Learning to Classify Horn Sounds

Horn Classification

- A pre-trained TensorFlow Lite model checks if the sound is a horn.

Mel Spectrogram: Convert audio to a frequency-time map (Mel spectrogram) with 2048-point FFT, 512-sample hop, 512 Mel bands, 86 time steps.



Normalizes the spectrogram :

$$S_{\text{norm}} = \frac{S_{\text{dB}} - 1.39 \times 10^{-14}}{0.9999999999953707}$$

Model Prediction:

- Predicted class (e.g., 'horn').
- Confidence (as %).
- Direction angle (θ_{deg}).
- Is it a horn? (Yes if class = 'horn' and confidence > 0.6).
- Left or right? (Left if $\theta_{\text{deg}} < 0$).

Test Accuracy - 97%

Sub Objective

3

Apply TDOA to find the horn's direction

The system figures out if the horn comes from the left or right using the time difference between the microphones.

Cross-Correlation: Compares left and right channel signals to find the time delay

$$\text{time\_difference} = \frac{\text{max\_lag}}{44100}$$

Angle Calculation:

$$\theta = \arcsin\left(\frac{\text{time\_difference} \cdot 343}{0.1}\right) \rightarrow \theta_{\text{deg}} = \frac{180}{\pi} \cdot \theta$$

Direction:

- If $\theta_{\text{deg}} < 0$: Sound comes from the left.
- If $\theta_{\text{deg}} > 0$: Sound comes from the right.

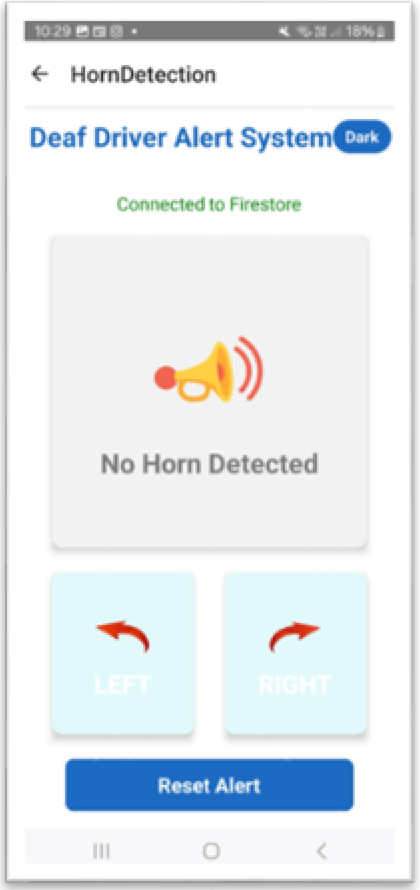
100%

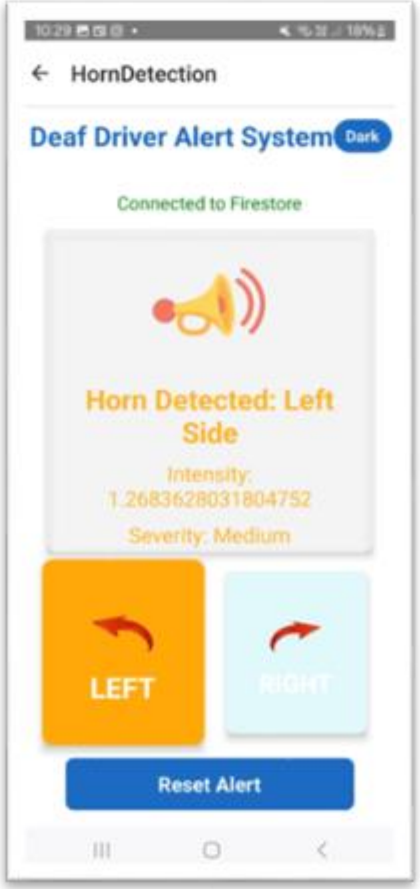
Sub Objective

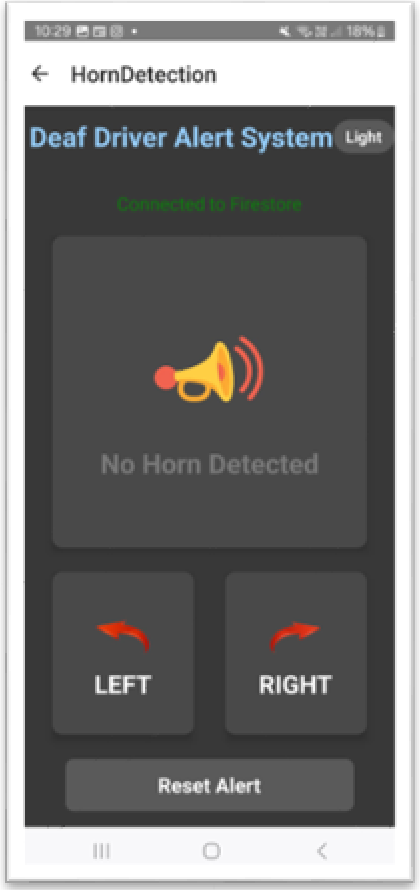
4

Build a Mobile App for Visual & Haptic Alerts

DeafConnect







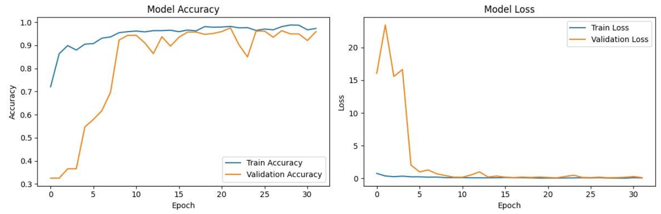
100%

Evidences for Completion

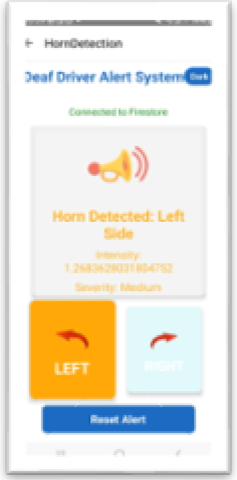
Summary of Sub Objective Achievement

- Data Collection and Pre-Processing
- Training the Model
- Accuracy Measures
- Testing using an App

```python
# Step 1: Label Encoding
label_encoder = LabelEncoder()
encoded_labels = label_encoder.fit_transform(labels)
# Convert to one-hot encoded format for multi-class classification
encoded_labels = to_categorical(encoded_labels, num_classes=len(label_encoder.classes_))

# Step 2: Data Preprocessing

# Normalize the mel-spectrograms (scale to [0, 1])
Mel_features = (Mel_features - np.min(Mel_features)) / (np.max(Mel_features) - np.min(Mel_features))
print("Min = ",np.min(Mel_features))
print("Max = ",np.max(Mel_features))


# Reshape data to include channel dimension for CNN (n_samples, height, width, channels)
Mel_features = Mel_features[..., np.newaxis]  # Shape: (n_samples, 512, max_length_frames, 1)

# Split into training and testing sets (80% train, 20% test)
X_temp, X_test, y_temp, y_test = train_test_split(
    Mel_features, encoded_labels, test_size=0.2, random_state=40, stratify=encoded_labels
)

# Further split training+validation into training and validation (75% train, 25% val of the 80%)
X_train, X_val, y_train, y_val = train_test_split(
    X_temp, y_temp, test_size=0.3, random_state=62, stratify=y_temp
)

# Print shapes to verify
print(f"Training set shape: {X_train.shape}, {y_train.shape}")
print(f"Validation set shape: {X_val.shape}, {y_val.shape}")
print(f"Test set shape: {X_test.shape}, {y_test.shape}")
```

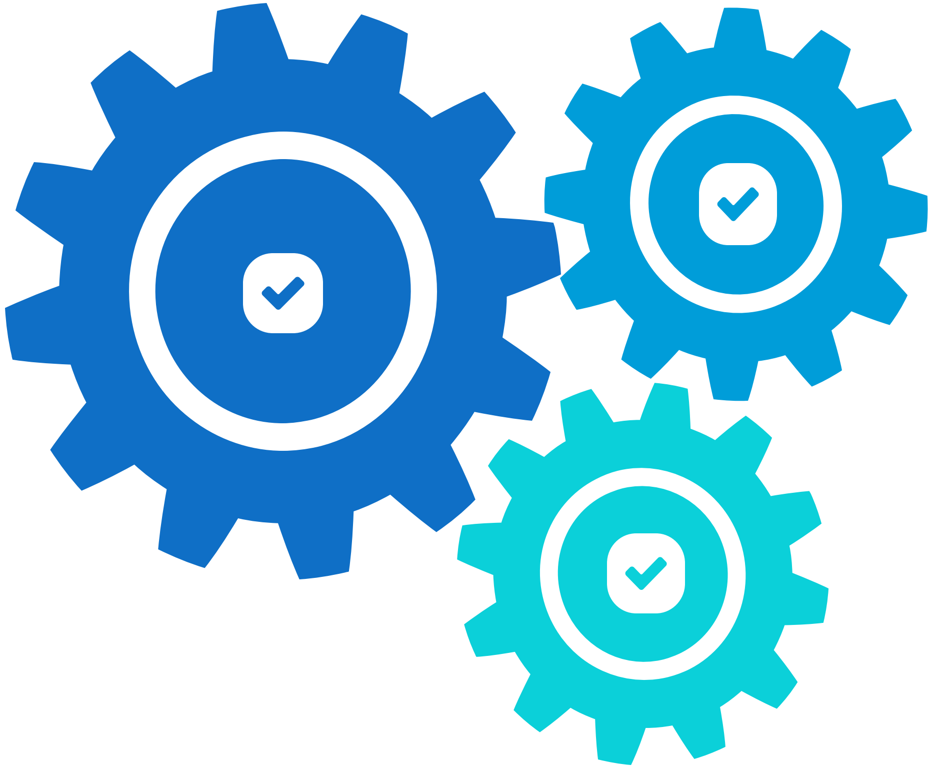


Test Accuracy: 0.9732



100% Completion

Technologies, Techniques & Algorithms



Non-Functional Requirements

- Real-Time Response
- Usability for Deaf drivers
- Reliability and accuracy

Standards/
Best Practices

- Modular Design
- Simple UI for Deaf Users
- Error Handling
- Open-source Frameworks

Ethical, Legal, and Social Issues

- Respects user privacy.
- Does not store personal data.
- Designed for safety and accessibility.
- Supports equal access for deaf drivers.

Proof of Completion

Data Preprocessing

```python
# Step 1: Label Encoding
label_encoder = LabelEncoder()
encoded_labels = label_encoder.fit_transform(labels)
# Convert to one-hot encoded format for multi-class classification
encoded_labels = to_categorical(encoded_labels, num_classes=len(label_encoder.classes_))

# Step 2: Data Preprocessing

# Normalize the mel-spectrograms (scale to [0, 1])
Mel_features = (Mel_features - np.min(Mel_features)) / (np.max(Mel_features) - np.min(Mel_features))
print("Min = ",np.min(Mel_features))
print("Max = ",np.max(Mel_features))


# Reshape data to include channel dimension for CNN (n_samples, height, width, channels)
Mel_features = Mel_features[..., np.newaxis]  # Shape: (n_samples, 512, max_length_frames, 1)

# Split into training and testing sets (80% train, 20% test)
X_temp, X_test, y_temp, y_test = train_test_split(
    Mel_features, encoded_labels, test_size=0.2, random_state=40, stratify=encoded_labels
)

# Further split training+validation into training and validation (75% train, 25% val of the 80%)
X_train, X_val, y_train, y_val = train_test_split(
    X_temp, y_temp, test_size=0.3, random_state=62, stratify=y_temp
)

# Print shapes to verify
print(f"Training set shape: {X_train.shape}, {y_train.shape}")
print(f"Validation set shape: {X_val.shape}, {y_val.shape}")
print(f"Test set shape: {X_test.shape}, {y_test.shape}")
```

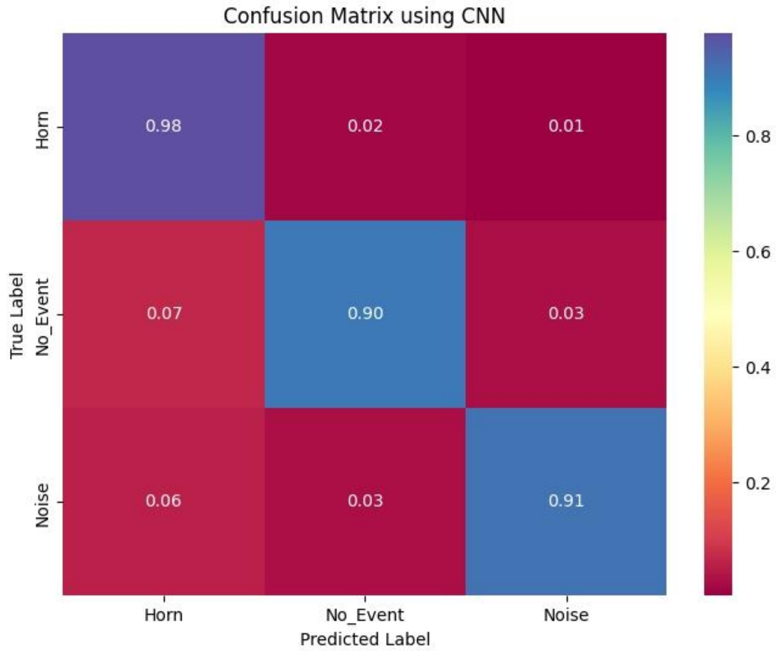
Build the CNN Model

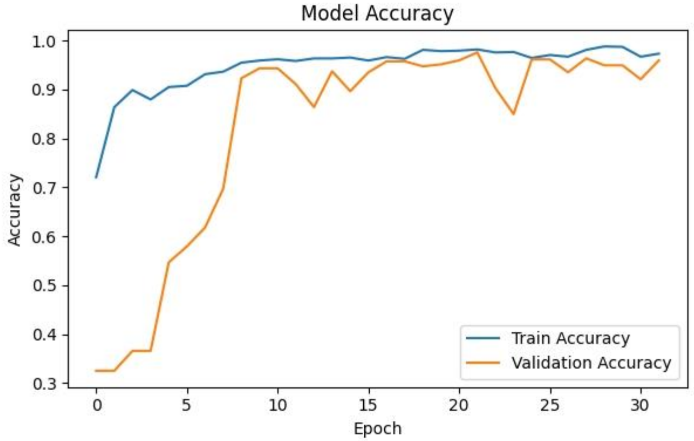
```python
# Step 3: Build the CNN Model
def create_cnn_model(input_shape, num_classes):
    model = models.Sequential([
        # First Convolutional Block
        layers.Conv2D(32, (3, 3), activation='relu', padding='same', input_shape=input_shape),
        layers.BatchNormalization(),
        layers.MaxPooling2D((2, 2)),
        layers.Dropout(0.25),

        # Second Convolutional Block
        layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
        layers.BatchNormalization(),
        layers.MaxPooling2D((2, 2)),
        layers.Dropout(0.25),

        # Third Convolutional Block
        layers.Conv2D(128, (3, 3), activation='relu', padding='same'),
        layers.BatchNormalization(),
        layers.MaxPooling2D((2, 2)),
        layers.Dropout(0.25),

        # Flatten and Dense Layers
        layers.Flatten(),
        layers.Dense(256, activation='relu'),
        layers.BatchNormalization(),
        layers.Dropout(0.5),
        layers.Dense(num_classes, activation='softmax')
    ])
    return model

# Define input shape and number of classes
input_shape = (Mel_features.shape[1], Mel_features.shape[2], 1)  # (512, max_length_frames, 1)
num_classes = len(label_encoder.classes_)  # 3 classes

# Create and compile the model
model = create_cnn_model(input_shape, num_classes)
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Print model summary
model.summary()
```

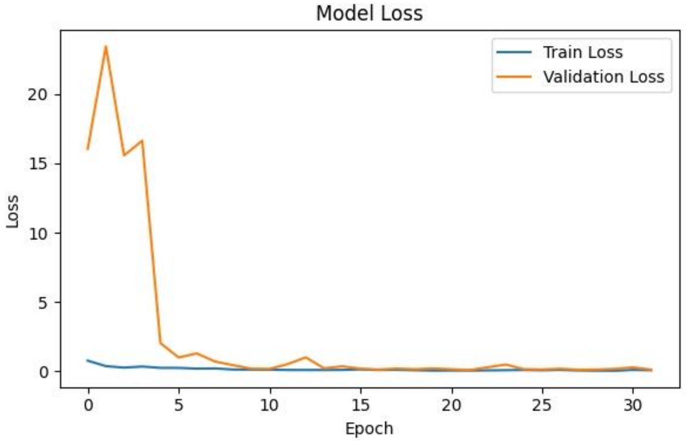
Proof of Completion

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 512, 86, 32)	320
batch_normalization (BatchNormalization)	(None, 512, 86, 32)	128
max_pooling2d (MaxPooling2D)	(None, 256, 43, 32)	0
dropout (Dropout)	(None, 256, 43, 32)	0
conv2d_1 (Conv2D)	(None, 256, 43, 64)	18,496
batch_normalization_1 (BatchNormalization)	(None, 256, 43, 64)	256
max_pooling2d_1 (MaxPooling2D)	(None, 128, 21, 64)	0
dropout_1 (Dropout)	(None, 128, 21, 64)	0
conv2d_2 (Conv2D)	(None, 128, 21, 128)	73,856
batch_normalization_2 (BatchNormalization)	(None, 128, 21, 128)	512
max_pooling2d_2 (MaxPooling2D)	(None, 64, 10, 128)	0
dropout_2 (Dropout)	(None, 64, 10, 128)	0
flatten (Flatten)	(None, 81920)	0
dense (Dense)	(None, 256)	20,971,776
batch_normalization_3 (BatchNormalization)	(None, 256)	1,024
dropout_3 (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 3)	771

Total params: 21,067,139 (80.36 MB)
Trainable params: 21,066,179 (80.36 MB)
Non-trainable params: 960 (3.75 KB)





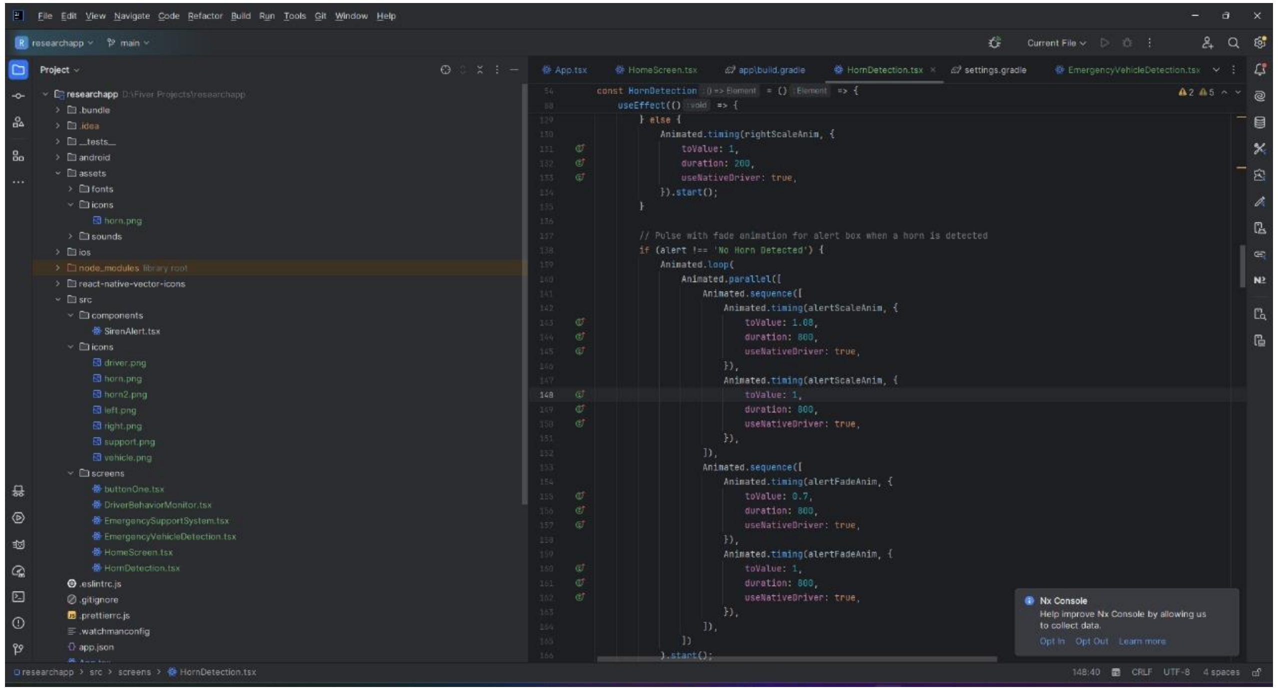


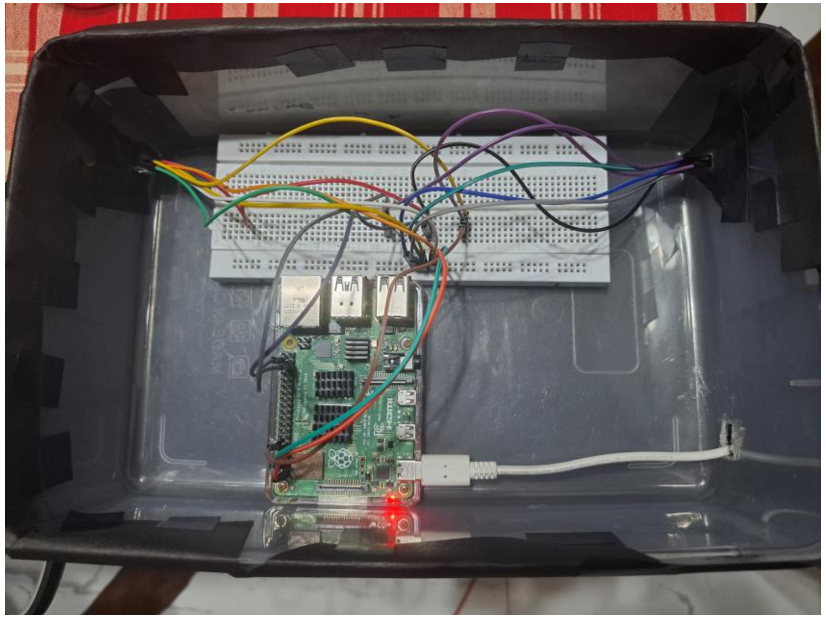
Test Accuracy - 97%

Test Accuracy: 0.9732

Proof of Completion

Evidence of Completion





References

[1] Gomez, R., & Lee, S. (2023). "Real-Time Acoustic Signal Processing for Assistive Technologies." *Journal of Assistive Technology and Accessibility Research*, 22(4), 401–419. Retrieved from https://doi.org/10.1016/j.jatar.2023.04.015.

[2] Chen, H., & Kumar, P. (2023). "Machine Learning Approaches for Sound Recognition in Autonomous Vehicles." *IEEE Transactions on Intelligent Transportation Systems*, 25(5), 2450–2462. Retrieved from https://doi.org/10.1109/TITS.2023.3056125.

[3] Garcia, M., & Rodriguez, F. (2023). "IoT Applications for Enhancing Road Safety: A Focus on Hearing Impairments." *Sensors and Actuators: A Physical*, 350, 113789. Retrieved from https://doi.org/10.1016/j.sna.2023.113789.

[4] Smith, J., & Brown, A. (2023). "Assistive Technologies for Deaf Drivers: A Review." *Journal of Transportation Research*, 56(3), 201–215. Retrieved from https://doi.org/10.1016/j.jtr.2023.03.007.

[5] Wang, L., & Zhang, Y. (2022). "IoT-Based Sound Detection Systems for the Hearing Impaired." *IEEE Transactions on Automation Science and Engineering*, 19(2), 1125–1134. Retrieved from https://doi.org/10.1109/TASE.2022.3142501.

[6] Williams, R. (2021). *Intelligent Systems for Sound Recognition: From Theory to Implementation.* Berlin: Springer. ISBN 978-3-030-56789-0. Retrieved from https://doi.org/10.1007/978-3-030-56790-6.

[7] Anderson, J. (2023). *Assistive Technology in Modern Transportation: Challenges and Innovations.* London: Wiley. ISBN 978-1-118-92783-1.

[8] Davis, L. (2020). *Machine Learning Techniques for Real-Time Sound Classification.* London: Academic Press. ISBN 978-0-12834-5678.

[9] National Institute on Deafness and Other Communication Disorders (NIDCD). (2023). "Driving and Transportation Options for People with Hearing Loss." Accessed August 5, 2024. https://www.nidcd.nih.gov/health/driving-hearing-loss.

[10] IEEE Spectrum. (2023). "How IoT is Revolutionizing Road Safety for Hearing-Impaired Drivers." Accessed August 5, 2024. https://spectrum.ieee.org/iot-road-safety-hearing-impaired.

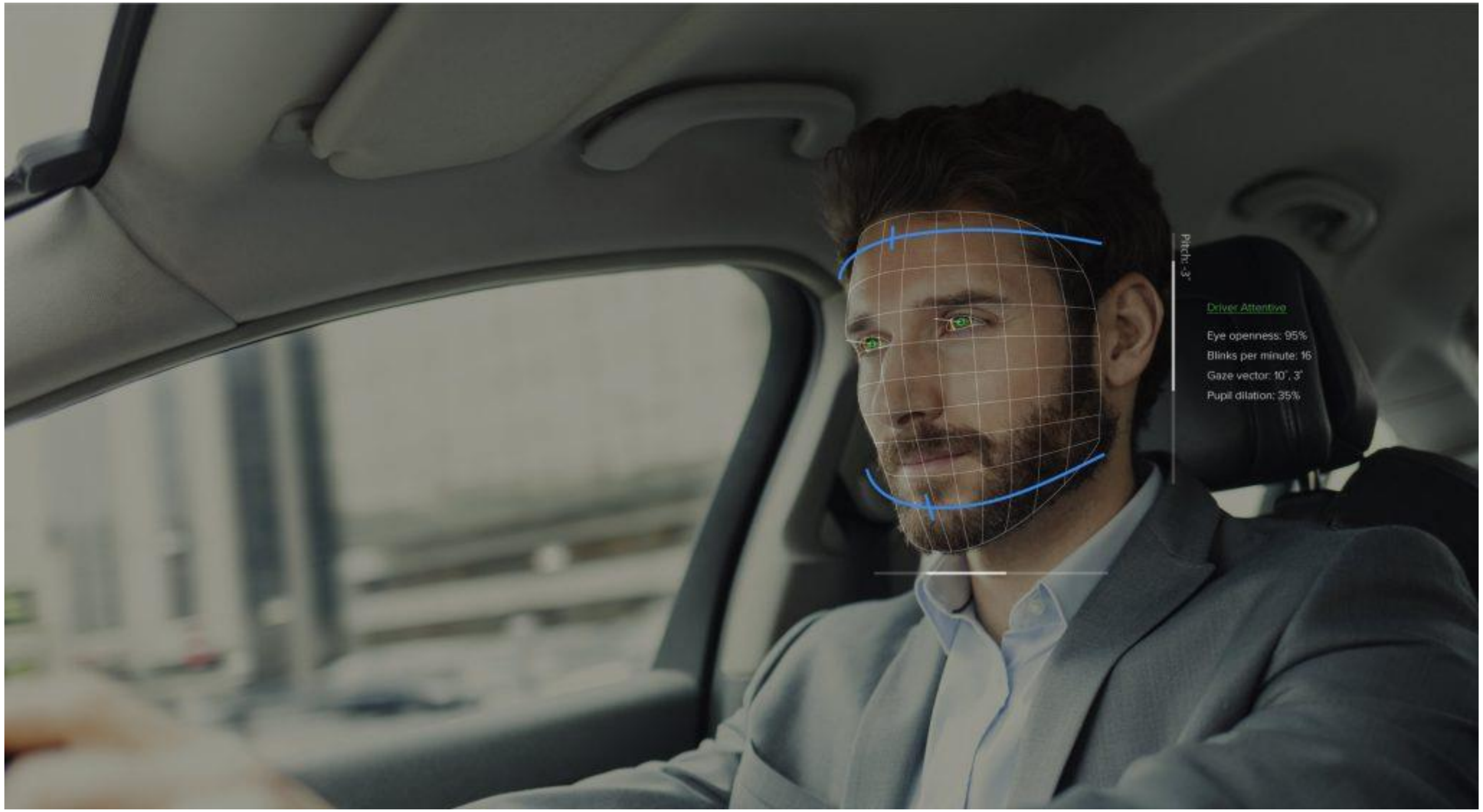
[11] Transportation Research Board (TRB). (2023). "Advancements in Assistive Technologies for Drivers with Disabilities." Accessed August 5, 2024. https://www.trb.org/assistive-technologies-drivers-disabilities.

Thathsara S. M. K.

IT21219566

Software Engineering

Comprehensive Driver Behavior Monitoring System.



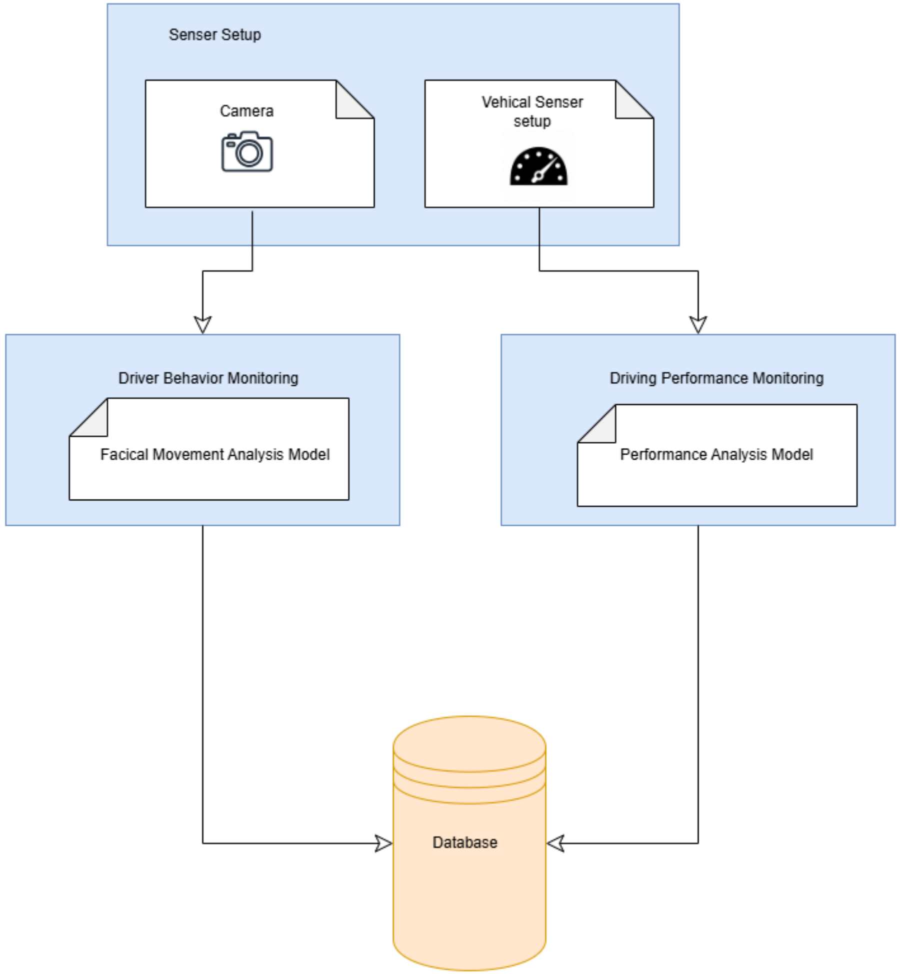
Background

❖ Multimodal Approaches in Driver Behavior Analysis:

1. CNN for Visual Inputs
   - **Purpose:** Process spatial information from video or image data.
   - **Model Architecture:** CNNs for video sequences to capture spatiotemporal features.

2. RNN for Temporal Sensor Data
   - **Purpose:** Handle sequential data such as time-series signals.
   - **Model Architecture:** LSTMs (Long Short-Term Memory) for long-range temporal dependencies.

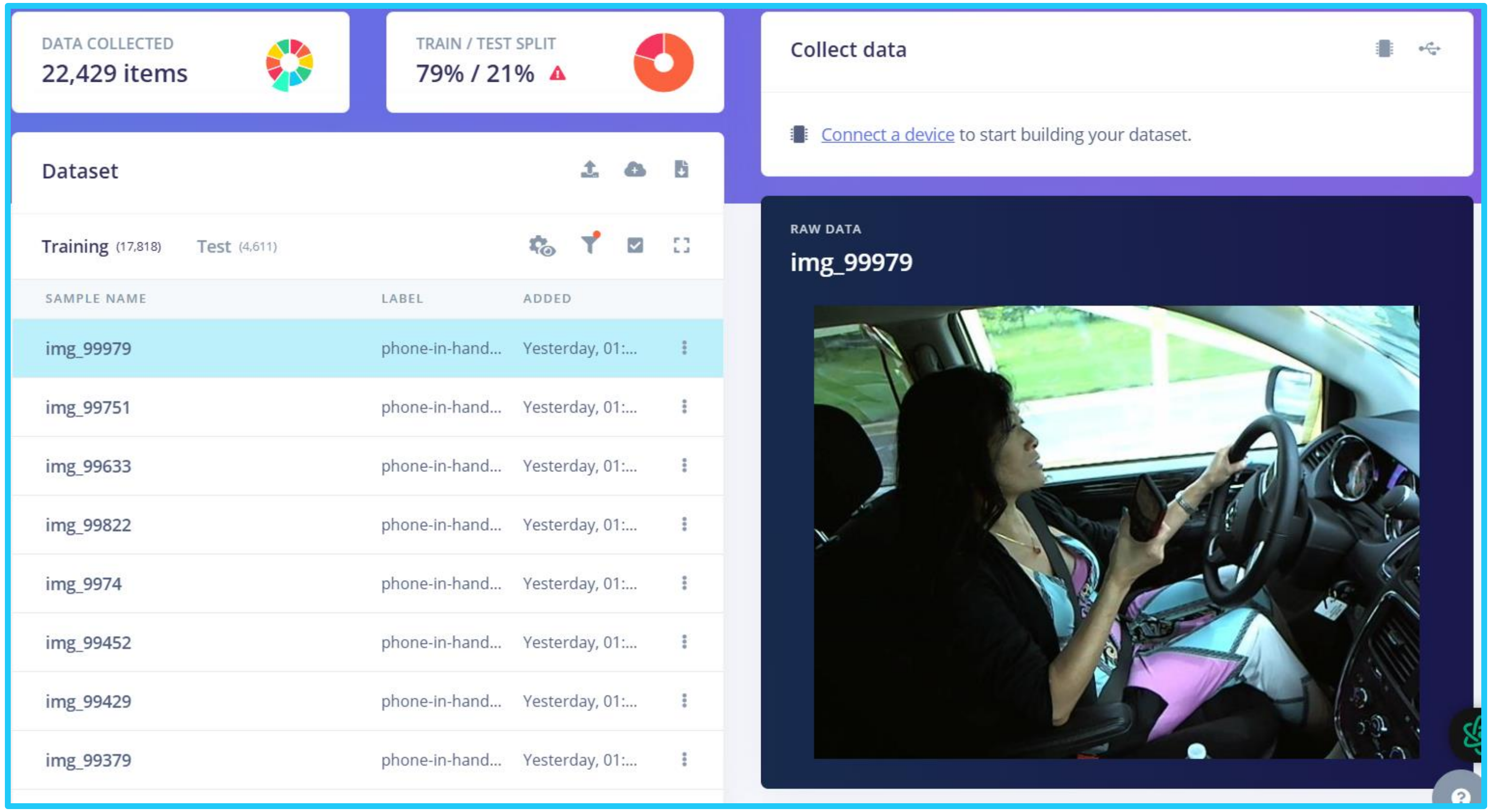
System Diagram



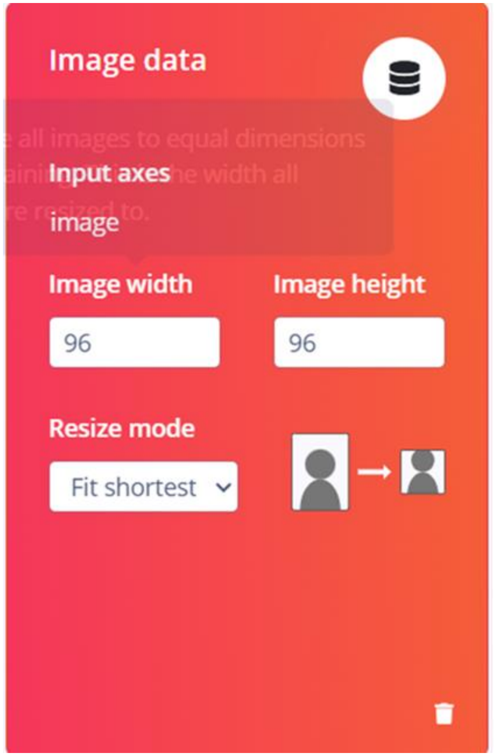
Data Collection

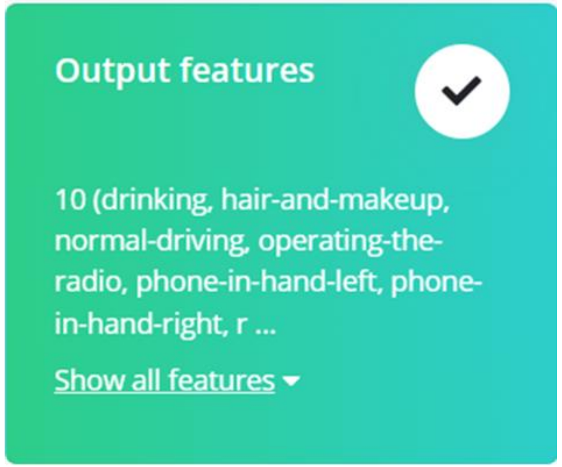
- Driver behavior data was collected from Google, Kaggle.

  https://www.kaggle.com/competitions/state-farm-distracted-driver-detection



Data Preprocessing





- ❑ 10 Data classes

1. Normal driving
2. Texting on right hand
3. Talking on the phone right hand
4. Texting on left hand
5. Talking on the phone left hand
6. Operating the radio
7. Drinking / eating
8. Reaching behind
9. Hair and makeup
10. Talking to the passenger

Model Training

Neural Network settings

Training settings

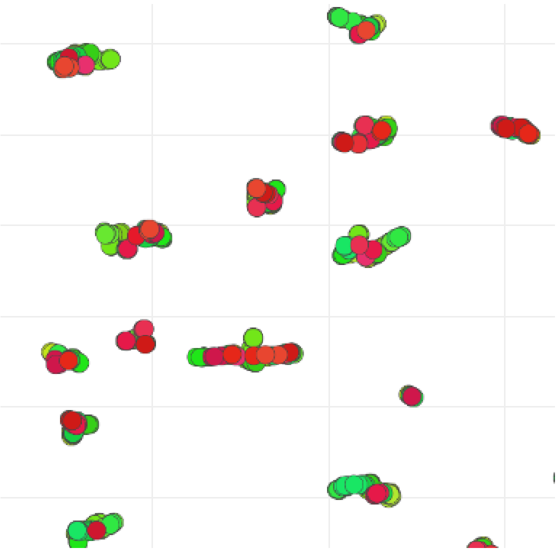
Number of training cycles ⓘ	200
Use learned optimizer ⓘ	☐
Learning rate ⓘ	0.0001
Training processor ⓘ	CPU
Data augmentation ⓘ	☐

Advanced training settings

Validation set size ⓘ	20 %
Split train/validation set on metadata key ⓘ	
Batch size ⓘ	32
Auto-weight classes ⓘ	☐
Profile int8 model ⓘ	☑

Data explorer (full training set) ⓘ

- drinking - correct
- hair-and-makeup - correct
- normal-driving - correct
- operating-the-radio - correct
- phone-in-hand-left - correct
- phone-in-hand-right - correct
- reaching-behind - correct
- talking-on-the-phone-left - correct
- talking-on-the-phone-right - correct
- talking-with-passenger - correct
- drinking - incorrect
- hair-and-makeup - incorrect
- normal-driving - incorrect
- operating-the-radio - incorrect
- phone-in-hand-left - incorrect



On-device performance ⓘ

Engine: ⓘ EON™ Compiler (RAM optimized)

INFERENCING ...	PEAK RAM USA...	FLASH USAGE
3 ms.	159.3K	175.5K

Model Accuracy

ACCURACY
91.93%

Metrics for Transfer learning

METRIC	VALUE
Area under ROC Curve	1.00
Weighted average Precision	0.95
Weighted average Recall	0.95
Weighted average F1 score	0.95

Confusion matrix

	DRINK	HAIR-/	NORM	OPERA	PHON	PHON	REACH	TALKI	TALKI	TALKI	UNCEI
DRINKIN	93.2%	0.2%	0%	0.2%	0%	0.6%	0%	0.6%	0.6%	0.2%	4.3%
HAIR-ANI	1.8%	84.3%	0%	0.3%	0.3%	0.3%	1.3%	0.3%	0.8%	0.8%	10.0%
NORMAL-	0%	0.4%	89.5%	0.2%	1.5%	0.4%	0.2%	0.2%	0%	0.4%	7.2%
OPERATII	0.2%	0%	0%	96.2%	0.4%	0%	0%	0.6%	0%	0.2%	2.3%
PHONE-II	0%	0.2%	0.8%	0%	92.7%	0%	0%	0%	0%	0%	6.3%
PHONE-II	0.4%	0.2%	0%	0%	0.2%	95.2%	0%	0%	0.2%	0%	3.8%
REACHIN	0%	0.3%	0%	0%	0%	0.3%	95.4%	0.3%	0%	0.8%	3.0%
TALKING-	0.2%	0.8%	0.6%	0.2%	0.8%	0%	0.4%	89.4%	0.2%	0.8%	6.4%
TALKING-	0.6%	0.4%	0%	0%	0%	0.2%	0.2%	0%	94.6%	0%	3.9%
TALKING-	0%	1.5%	1.2%	0.7%	0.5%	0.2%	0.5%	0.2%	0%	87.6%	7.5%
F1 SCORE	0.95	0.89	0.93	0.97	0.94	0.97	0.96	0.93	0.96	0.92	

Model Deployment

```python
# Capture webcam frames
cap = cv2.VideoCapture(0)

while True:
    ret, frame = cap.read()
    if not ret:
        break

    # Resize frame to match model input size (e.g., 96x96)
    resized_frame = cv2.resize(frame, (96, 96))

    # Classify the frame
    result = classify_image(resized_frame)
    print(result)

    # Store the result and timestamp in Firebase
    store_result(result)

    # Display the frame
    cv2.imshow('Webcam', frame)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

cap.release()
cv2.destroyAllWindows()
```

DBM-Model-Run.py

DBM-Model-Run.py > store_result

```python
import cv2
import requests
import firebase_admin
from firebase_admin import credentials, firestore
from datetime import datetime

# Initialize Firebase
cred = credentials.Certificate("D:/Driver-Behavior-Monitoring-Model-Run/serviceAccountKey.json")
firebase_admin.initialize_app(cred)
db = firestore.client()

# Function to send image to inference server
def classify_image(image):
    url = "http://localhost:1337/api/image"
    _, img_encoded = cv2.imencode('.jpg', image)
    files = {'file': ('image.jpg', img_encoded.tobytes(), 'image/jpeg')}
    response = requests.post(url, files=files)
    return response.json()

# Function to store result and timestamp in Firebase
def store_result(result):
    # Get the predicted class (label with the highest confidence)
    predicted_class = max(result['result']['classification'], key=result['result']['classification'].get)

    # Store result and timestamp in Firebase
    doc_ref = db.collection('detections').document()  # Use .document() instead of .doc()
    doc_ref.set({
        'result': predicted_class,
        'timestamp': datetime.now().isoformat()
    })

# Capture webcam frames
cap = cv2.VideoCapture(0)
```

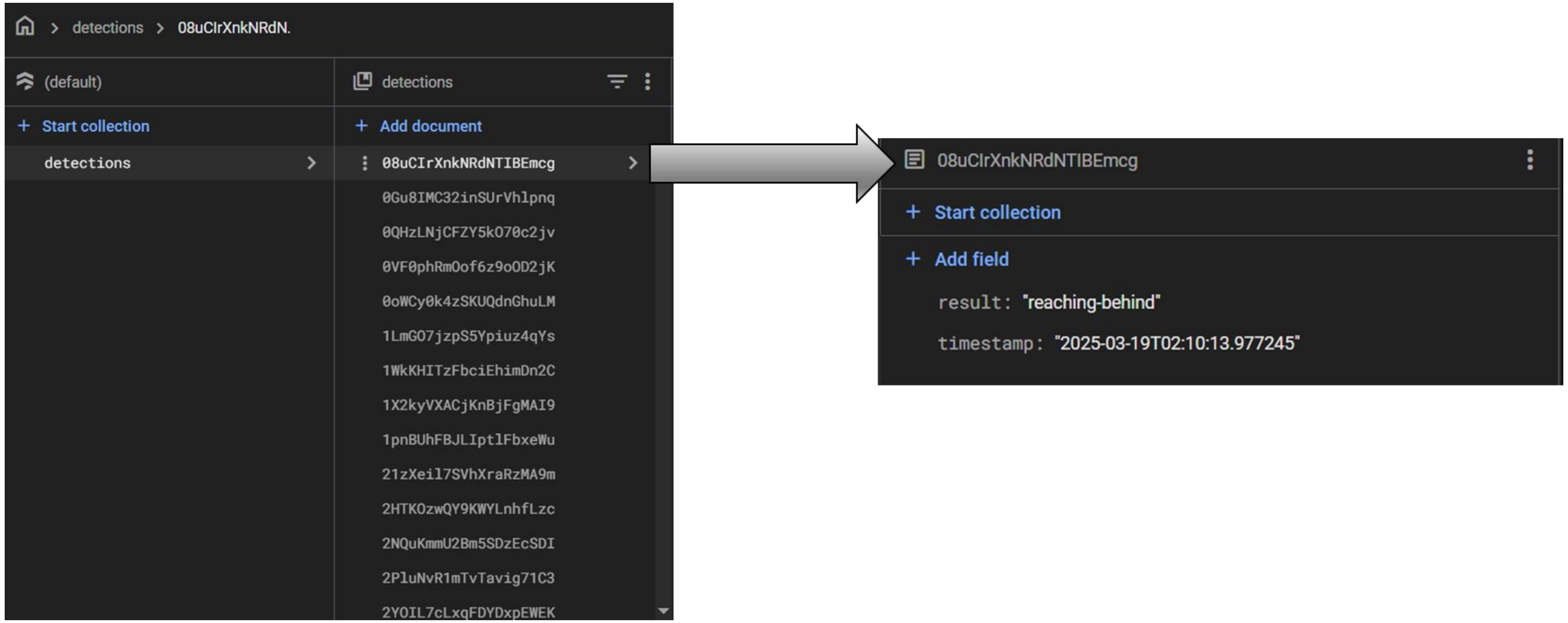
Model Deployment

Docker container that includes an HTTP interface to run the model.

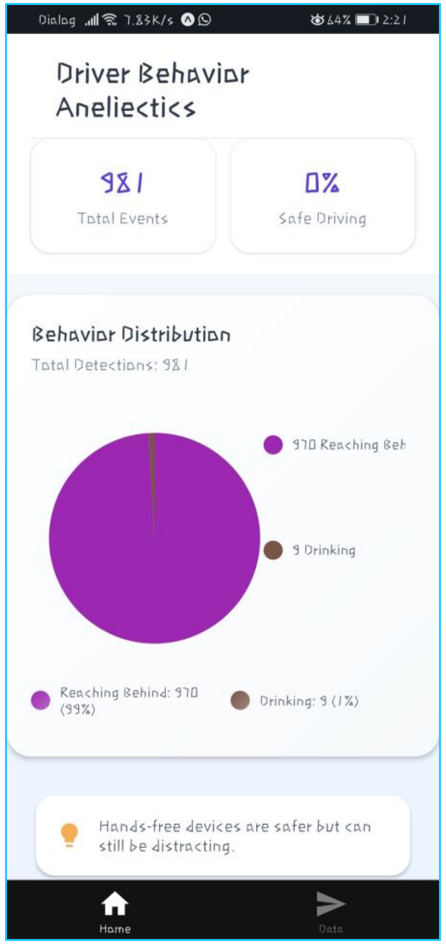
```
PS D:\Driver-Behavior-Monitoring-Model-Run> & C:/Python311/python.exe d:/Driver-Behavior-Monitoring-Model-Run/DBM-Model-Run.py
{'result': {'classification': {'drinking': 0.00018309261940885335, 'hair-and-makeup': 1.2963785138708772e-06, 'normal-driving': 3.904605705429276e-07, 'operati
ng-the-radio': 4.260187438376306e-07, 'phone-in-hand-left': 1.039543462866277e-06, 'phone-in-hand-right': 0.00031643625698052347, 'reaching-behind': 0.99392151
83258057, 'talking-on-the-phone-left': 6.087520765252208e-11, 'talking-on-the-phone-right': 0.0055737304501235485, 'talking-with-passenger': 2.089898316626204e
-06}}, 'timing': {'anomaly': 0, 'classification': 32, 'dsp': 0, 'json': 19, 'stdin': 1}}
{'result': {'classification': {'drinking': 8.736484596738592e-05, 'hair-and-makeup': 7.89706646742161e-08, 'normal-driving': 4.275167719836048e-11, 'operating-
the-radio': 8.759949565728675e-08, 'phone-in-hand-left': 1.4678050441752077e-10, 'phone-in-hand-right': 0.00012645396054722237, 'reaching-behind': 0.99976295232
77283, 'talking-on-the-phone-left': 3.215795947876499e-15, 'talking-on-the-phone-right': 2.3131262423703447e-05, 'talking-with-passenger': 5.321054263873748e-0
9}}, 'timing': {'anomaly': 0, 'classification': 0, 'dsp': 0, 'json': 0, 'stdin': 0}}
```

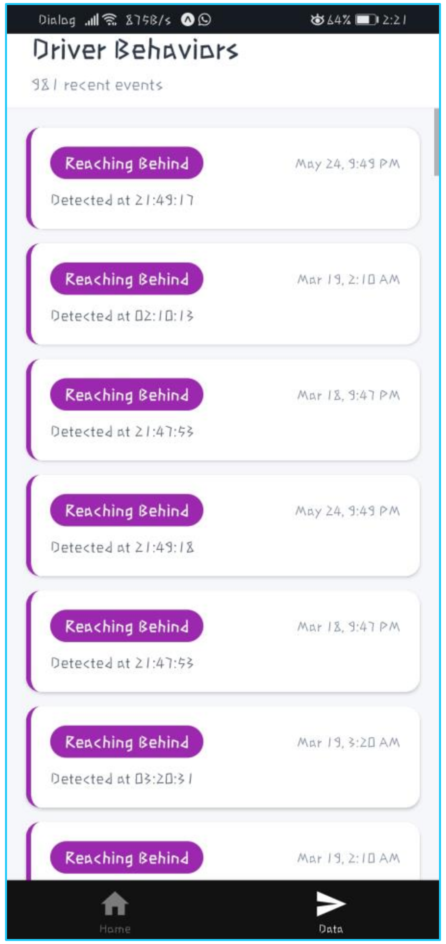
Database Configuration

```
# Initialize Firebase
cred = credentials.Certificate("D:/Driver-Behavior-Monitoring-Model-Run/serviceAccountKey.json")
firebase_admin.initialize_app(cred)
db = firestore.client()
```



Mobile Application





References

1. A. Alamri, A. Gumaei, M. Al-Rakhami, M. M. Hassan, M. Alhussein and G. Fortino, "An Effective Bio-Signal-Based Driver Behavior Monitoring System Using a Generalized Deep Learning Approach," in IEEE Access, vol. 8, pp. 135037-135049, 2020, doi: 10.1109/ACCESS.2020.3011003.

2. Qu, Fangming & Dang, Nolan & Furht, Borko & Nojoumian, Mehrdad. (2024). Comprehensive study of driver behavior monitoring systems using computer vision and machine learning techniques. Journal of Big Data. 11. 10.1186/s40537-024-00890-0.

3. H. -B. Kang, "Various Approaches for Driver and Driving Behavior Monitoring: A Review," 2013 IEEE International Conference on Computer Vision Workshops, Sydney, NSW, Australia, 2013, pp. 616-623, doi: 10.1109/ICCVW.2013.85. keywords: {Vehicles;Feature extraction;Monitoring;Accidents;Visualization;Head;Wheels;Driver drowsiness monitoring;Driving behavior monitoring;Unsafe driving behavior prediction},

4. V. Sanjay Kumar, S. Nair Ashish, I.V. Gowtham, S.P. Ashwin Balaji, E. Prabhu, Smart driver assistance system using raspberry pi and sensor networks, Microprocessors and Microsystems, Volume 79, 2020, 103275, ISSN 0141-9331,

5. Sameer Rafee, Xu Yun, Zhang Jian Xin and Zaid Yemeni, “Eye-movement Analysis and Prediction using Deep Learning Techniques and Kalman Filter” International Journal of Advanced Computer Science and Applications(IJACSA), 13(4), 2022. http://dx.doi.org/10.14569/IJACSA.2022.01304107

Iroshan G.H.M

IT21229084

Software Engineering

Communication Support System for Deaf Drivers

INTRODUCTION

- LipAssist is a **lip reading model** designed to assist the deaf and hard-of-hearing community

- It uses **deep learning** to interpret lip movements and convert them into text.

RESEARCH QUESTION

How can deaf drivers share emergency details effectively without spoken communication ?

What delays do deaf drivers face in getting help during emergencies ?

OBJECTIVES

Specific Objective

To create a **real-time lip-reading tool** that helps deaf drivers **understand spoken instructions** during emergencies, ensuring **safety** and **clear communication.**

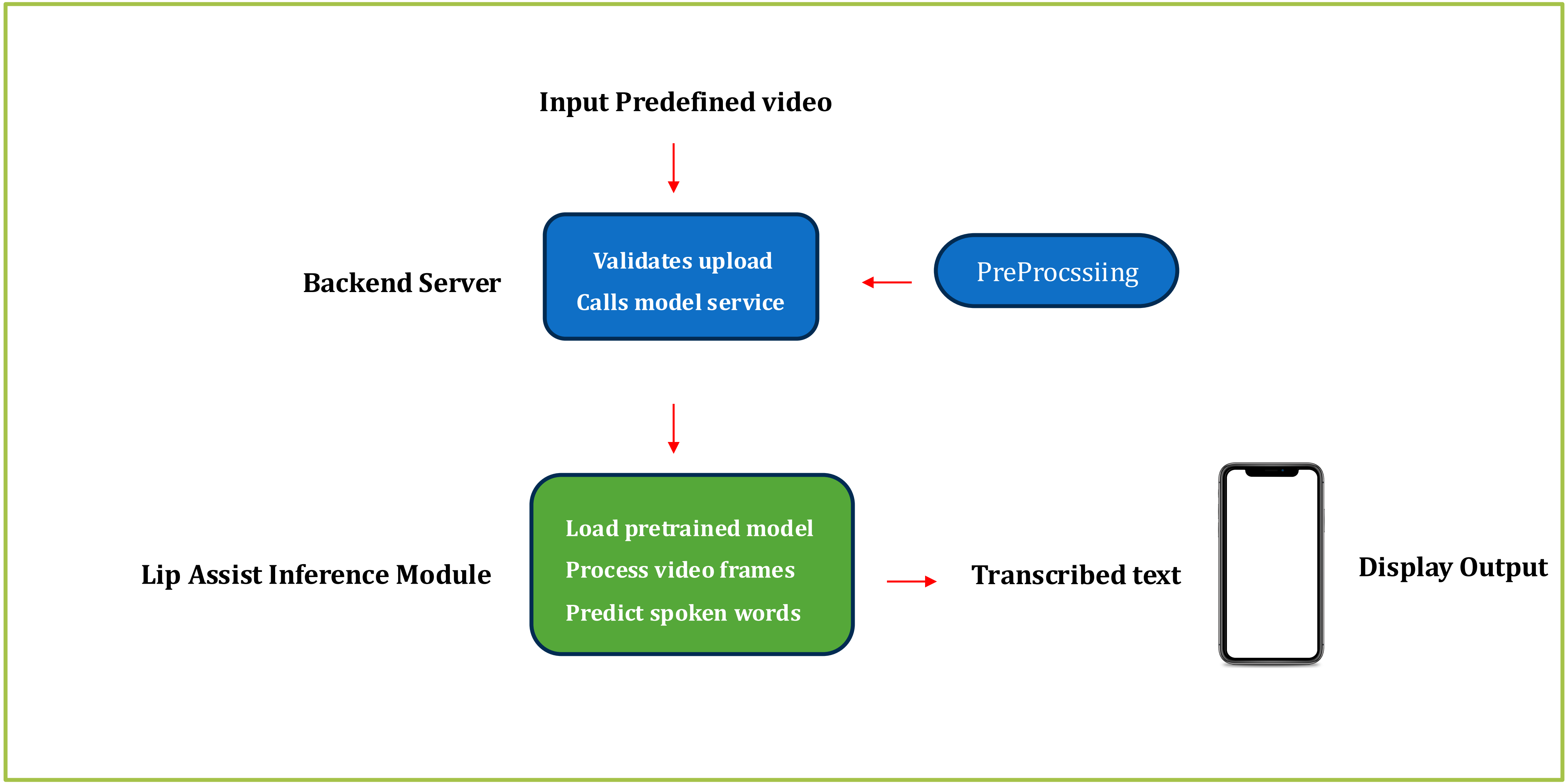
Sub Objective

Communication Facilitation

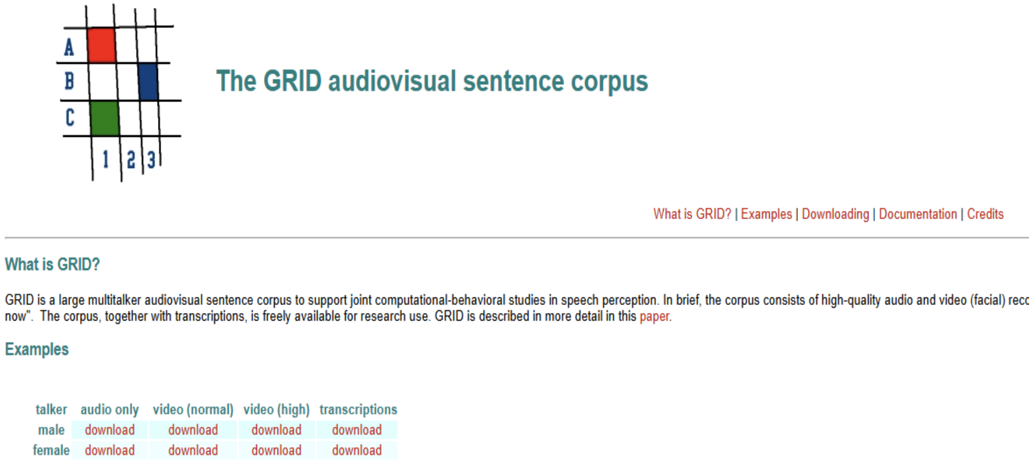
lip reading model

Emergency scenarios

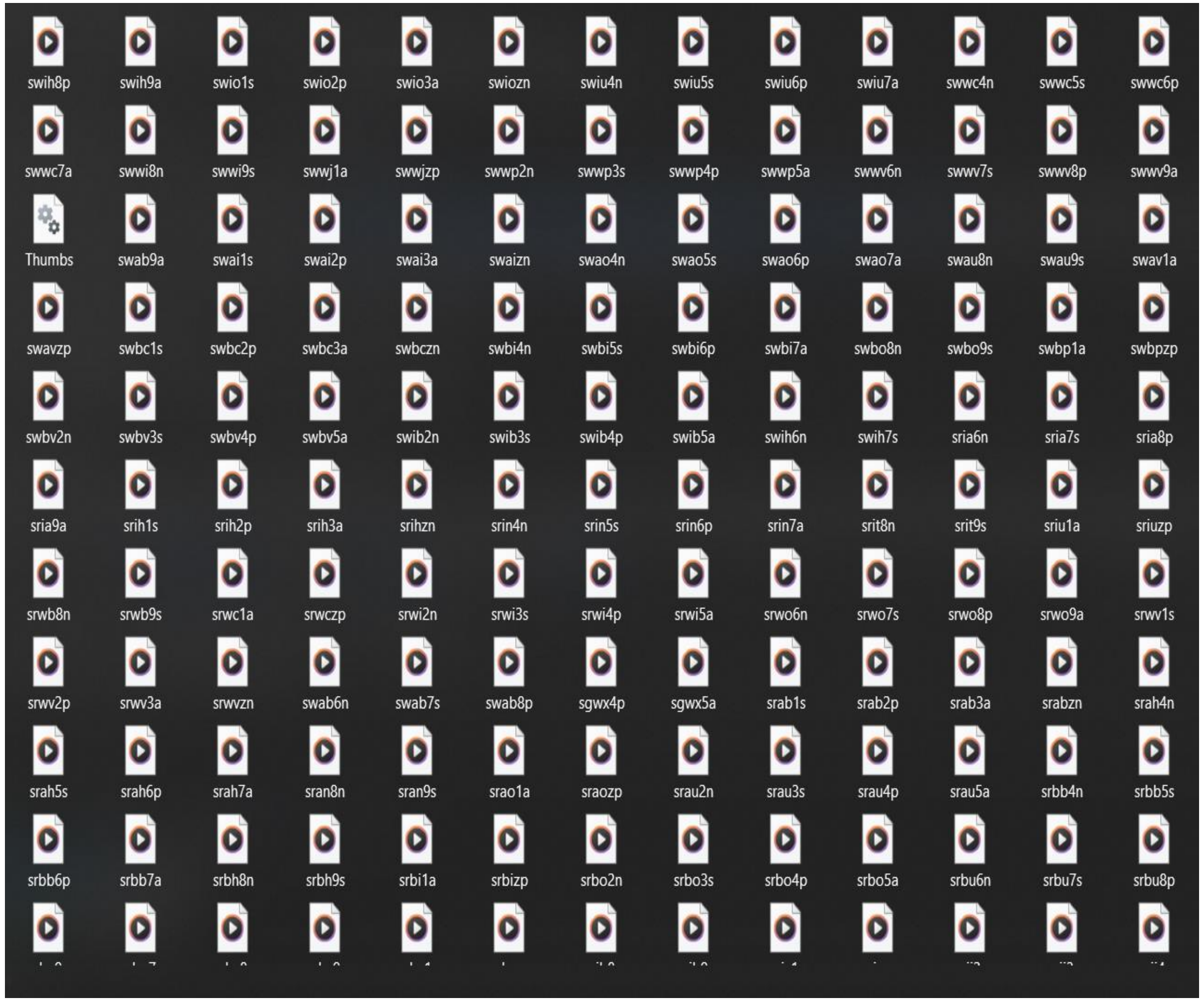
Offline Capability


Input Predefined video
Backend Server
Validates upload
Calls model service
PreProcssiing
Lip Assist Inference Module
Load pretrained model
Process video frames
Predict spoken words
Transcribed text
Display Output

Data collection



Alignments



Dataset

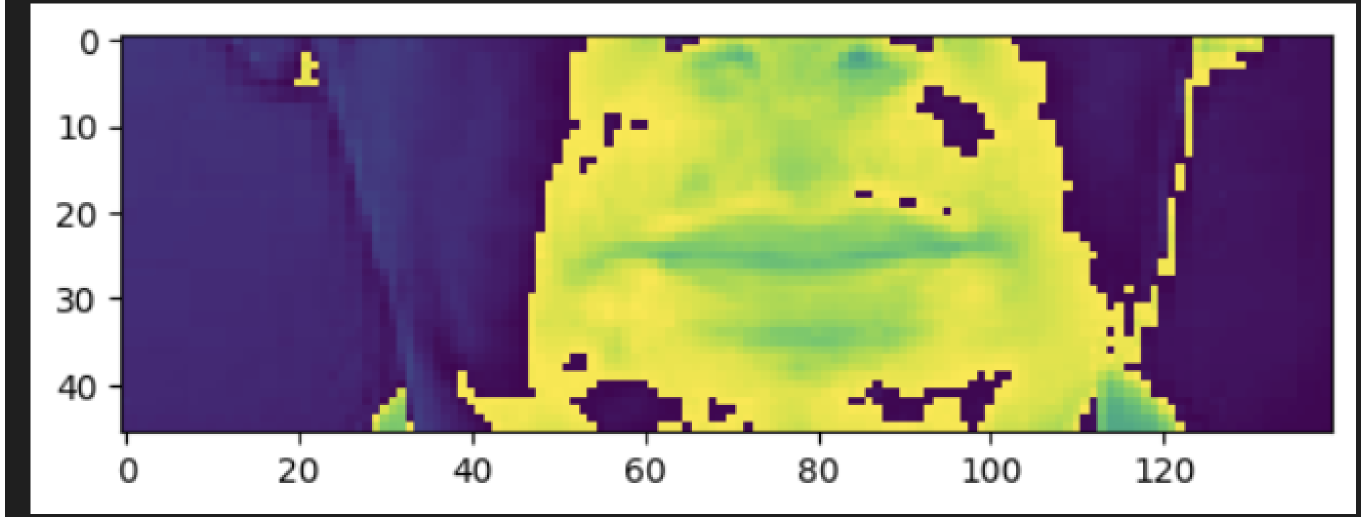
Word Structure

Word Type	Examples
Command	bin, lay, place, set
Color	blue, green, red, white
Preposition	at, by, in, with
Letter	A to Z (excluding W)
Digit	0 to 9

Alignment Data Visualization

```
print(f'This is the alingments of the one video: {alignments}')
print(' ')
tf.strings.reduce_join([bytes.decode(i) for i in num_to_char(alignments.numpy()).numpy()])
plt.imshow(frames[74])
```

```
This is the alingments of the one video: [ 2  9 14 39  2 12 21  5 39  1 20 39  6 39 20 23 15 39 14 15 23]


<matplotlib.image.AxesImage at 0x7eac63db5570>
```



Video Frame Data Representation & Predictions

```
print(f'whole the frame: {frames.shape}')
frames[0]
```

```
whole the frame: (75, 46, 140, 1)

<tf.Tensor: shape=(46, 140, 1), dtype=float32, numpy=
array([[[1.4991663 ],
        [1.4991663 ],
        [1.4616871 ],
        ...,
        [0.41227072],
        [0.41227072],
        [0.41227072]],

       [[1.4991663 ],
        [1.4991663 ],
        [1.4616871 ],
        ...,
        [0.41227072],
        [0.41227072],
        [0.41227072]],

       [[1.4616871 ],
        [1.4616871 ],
        [1.4991663 ],
        ...,
        [0.3373124 ],
        [0.3373124 ],
        [0.3373124 ]],
...
        [1.0119373 ],
        ...,
        [0.07495831],
        [0.07495831],
        [0.03747915]]], dtype=float32)>
```

```
Epoch 123: val_loss did not improve from 1.43750
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 122ms/step
Original:  set white at u nine soon
Predicted:  set white at u nine son
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Original:  set white by i four now
Predicted:  set white by i four now
```

```
Epoch 129: val_loss improved from 1.38357 to 1.31727, saving model to best_weights.weights.h5
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
Original:  set red by h eight now
Predicted:  set red by h eight now
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Original:  set blue at n two now
Predicted:  set blue at t two now
```

Model Architecture Summary

```
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv3d (Conv3D)	(None, 75, 46, 140, 128)	3,584
batch_normalization (BatchNormalization)	(None, 75, 46, 140, 128)	512
activation (Activation)	(None, 75, 46, 140, 128)	0
max_pooling3d (MaxPooling3D)	(None, 75, 23, 70, 128)	0
conv3d_1 (Conv3D)	(None, 75, 23, 70, 256)	884,992
batch_normalization_1 (BatchNormalization)	(None, 75, 23, 70, 256)	1,024
activation_1 (Activation)	(None, 75, 23, 70, 256)	0
max_pooling3d_1 (MaxPooling3D)	(None, 75, 11, 35, 256)	0
conv3d_2 (Conv3D)	(None, 75, 11, 35, 75)	518,475
batch_normalization_2 (BatchNormalization)	(None, 75, 11, 35, 75)	300
activation_2 (Activation)	(None, 75, 11, 35, 75)	0
max_pooling3d_2 (MaxPooling3D)	(None, 75, 5, 17, 75)	0
time_distributed (TimeDistributed)	(None, 75, 6375)	0
bidirectional (Bidirectional)	(None, 75, 256)	6,660,096
dropout (Dropout)	(None, 75, 256)	0
bidirectional_1 (Bidirectional)	(None, 75, 256)	394,240
dropout_1 (Dropout)	(None, 75, 256)	0
dense (Dense)	(None, 75, 41)	10,537

```
Total params: 8,473,760 (32.32 MB)

Trainable params: 8,472,842 (32.32 MB)

Non-trainable params: 918 (3.59 KB)
```

Backend Implementation

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

    # First Conv3D block
    model.add(Conv3D(128, 3, input_shape=(75, 46, 140, 1), padding='same'))
    model.add(BatchNormalization())  # Added BatchNormalization
    model.add(Activation('relu'))
    model.add(MaxPool3D(pool_size=(1, 2, 2)))

    # Second Conv3D block
    model.add(Conv3D(256, 3, padding='same'))
    model.add(BatchNormalization())  # Added BatchNormalization
    model.add(Activation('relu'))
    model.add(MaxPool3D(pool_size=(1, 2, 2)))

    # Third Conv3D block
    model.add(Conv3D(75, 3, padding='same'))
    model.add(BatchNormalization())  # Added BatchNormalization
    model.add(Activation('relu'))
    model.add(MaxPool3D(pool_size=(1, 2, 2)))

    # Flatten the spatial dimensions while preserving the time dimension
    model.add(TimeDistributed(Reshape((-1,))))

    # First LSTM block
    model.add(Bidirectional(LSTM(128, kernel_initializer='orthogonal', return_sequences=True)))
    model.add(Dropout(0.5))

    # Second LSTM block
    model.add(Bidirectional(LSTM(128, kernel_initializer='orthogonal', return_sequences=True)))
    model.add(Dropout(0.5))

    # Fully connected output layer
    model.add(Dense(char_to_num.vocabulary_size() + 1, kernel_initializer='he_normal', activation='softmax'))

    # model.load_weights('../models/best_weights.weights.h5')
    # Load the weights
    model.load_weights(os.path.join('models', 'new_best_weights2.weights.h5'))
    return model
```

```python
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
```

```python
def load_data(path: str):
    path = bytes.decode(path.numpy())
    print(f'This is the path: {path}')
    file_name = path.split('\\')[-1].split('.')[0]
    # Extract the file name without extension
    file_name = os.path.splitext(os.path.basename(path))[0]

    # Construct video and alignment paths
    video_path = os.path.join('data', 's1', f'{file_name}.mpg')
    alignment_path = os.path.join('data', 'alignments', 's1', f'{file_name}.align')

    # Ensure files exist
    if not os.path.exists(video_path):
        raise FileNotFoundError(f"Video file not found: {video_path}")
    if not os.path.exists(alignment_path):
        raise FileNotFoundError(f"Alignment file not found: {alignment_path}")

    # Load video frames and alignments
    frames = load_video(video_path)
    alignments = load_alignments(alignment_path)

    return frames, alignments
```

```python
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:]
```

Testing Using A WebApp