EDGE-AI BASED AUDIO CLASSIFIER

By

Om Dhingra
Ahaan Joishy
Kavin Manivasagam

Final Report for ECE 445, Senior Design, [Spring 2026]
TA: Gayatri Chandran

May 2026
Project No.80

Abstract

This project presents an Edge-AI based audio classification system capable of performing real-time sound recognition entirely on-device. While traditional embedded audio systems rely on noise-sensitive, threshold-based methods or require cloud connectivity for complex processing, this solution utilizes a Convolutional Neural Network (CNN) optimized for the ESP32-WROOM-S3 microcontroller. The system captures audio via dual ICS43434 MEMS microphones, transforms signals into spectrograms, and executes classification in approximately 70 ms. Contained in a custom 3D-printed enclosure with an integrated OLED interface, this system demonstrates high-accuracy performance under strict power and memory constraints. Potential applications include automated gunshot detection for campus safety, agricultural wildlife monitoring, and distress detection in healthcare environments.

Contents

1. Introduction

1.1 Problem

Most embedded audio-based systems rely on simple threshold-based logic to detect sounds by triggering when signal amplitude exceeds a predefined level. While computationally inexpensive, this approach is highly sensitive to environmental noise and cannot differentiate between sounds sharing similar loudness characteristics. In real-world environments, ambient sounds such as wind, traffic, or mechanical noise easily cause false triggers or mask important audio events.

Another major limitation of single-sensor systems is their inability to determine the direction of a sound source. Without spatial information, the system cannot distinguish whether a detected sound originates directly in front of the device or from a distant or irrelevant direction. This prevents localization-based filtering, beamforming, or directional response mechanisms critical for applications such as environmental monitoring, wildlife observation, and smart sensing platforms.

Cloud-based audio processing attempts to address these limitations through powerful remote servers capable of executing convolutional neural networks and spectral analysis with high accuracy. However, this introduces significant drawbacks: network latency delays real-time operation, transmitting raw audio raises serious privacy and security concerns, and continuous wireless communication increases power consumption, limiting portability and deployment in remote environments.

Therefore, there is a clear need for a low-power embedded system capable of performing real-time sound classification **and sound source localization** entirely on-device, under strict memory, processing, and energy constraints while maintaining acceptable accuracy and latency.

1.2 Solution

This project proposes an Edge-AI embedded system capable of performing real-time sound classification and sound source localization using two ICS43434 digital MEMS microphones and a convolutional neural network deployed on a low-power ESP32-WROOM-S3 microcontroller. The system extracts MFCC features from incoming audio signals and classifies sounds into predefined categories including bird vocalizations, gunshots, and speech. Simultaneously, a TDOA-based localization pipeline estimates the direction of arrival of the detected sound using cross-correlation between the two microphone channels. Results in both the sound class and estimated direction which are displayed on an integrated OLED display in real time. All processing is performed locally without cloud services, demonstrating the

feasibility of embedded machine learning and acoustic localization under strict power and memory constraints.

1.3 Visual Aid

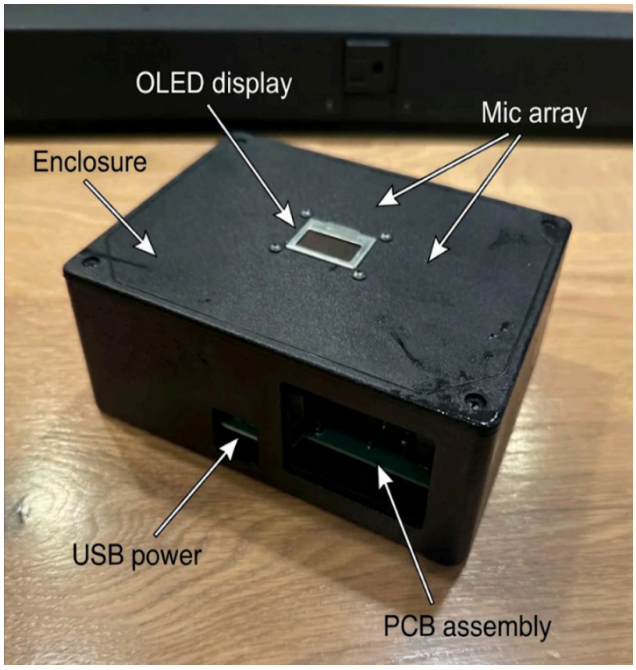


Figure 1: Visual display of project

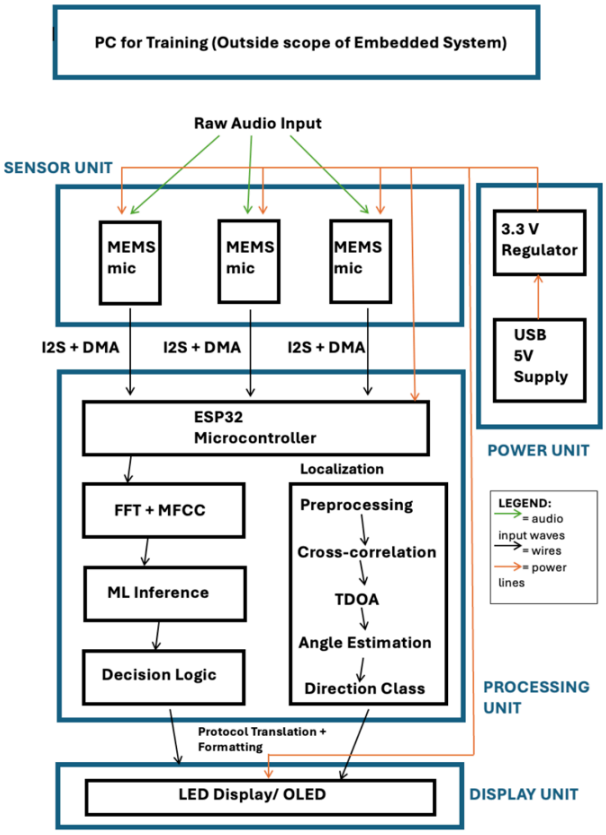


Figure 2: High-level Block diagram

2) High-level Requirements and Verifications

2.1) Requirement 1: Localization and Classification take place without a cloud/internet connection -> Success

In order to verify this, we disabled all settings that are required by the ESP 32 to use WIFI/Internet and we ran the demo without an internet connection. Furthermore, we use the serial for communication between python and Arduino, and our model is completely loaded on the device.

2.2) Requirement 2: Device gets power from a 5V USB-C power port -> Success

This is present on the device, can be seen in the visual aid, and was demonstrated in the demo too.

2.3) Requirement 3: The system shall perform real-time sound localization and classification with an end-to-end latency of less than 100 ms from sound detection to visual display output. -> Met the criterion for processing latency.

This requirement had to be changed. We had taken the '100 ms' latency in our design document because that seemed to be the norm for real-time audio classification. However, traditional methods rely on threshold-based classification and are unreliable, which is exactly what we were trying to avoid. Our method uses an AI model, which needs far more context than just 0.1 second. Thus, we increased the buffer to 2 seconds (2000 ms).

While the overall end-to-end latency was around 2000 ms, this was because of the engineering decision to increase the buffer to 2 s, to give the AI more context to be able to classify correctly and had nothing to do with the model/script's speed, and can be shown in the following table. As can be seen, all other processes, combined have a latency of less than 100 ms, which means our model still meets the criteria for real-time classification. It's only the audio capture part that takes about 1900-2000 ms because of the decision to increase the buffer size.

Audio capture	MFCC	Inference/ Classification	TDOA + Angle estimation	Serial write	TOTAL
1895.9 ms	12 ms	84.7 ms	1.3 ms	0.4 ms	1994.3 ms
1910.4 ms	9.4 ms	79.8 ms	1.4 ms	0.3 ms	2001.3 ms
1911.3 ms	10.2 ms	82.1 ms	1.5 ms	0.3 ms	2005.3 ms
1971.1 ms	15.7 ms	59.2 ms	1.3 ms	0.3 ms	2047.7 ms
1918.0 ms	9.5 ms	53.7 ms	1.3 ms	0.3 ms	1982.8 ms
1945.6 ms	9.8 ms	57.9 ms	1.3ms	0.4 ms	2015.0 ms
1914.6 ms	4.7 ms	52.5 ms	1.4 ms	0.8 ms	1974.0 ms
1940.0 ms	6.1 ms	65.1 ms	1.4 ms	0.3 ms	2012.9 ms
1881.1 ms	13.4 ms	48.1 ms	2.1 ms	0.2 ms	1945.0 ms
1893.0 ms	10.1 ms	93.8 ms	1.6 ms	0.3 ms	1998.9 ms
1933.3 ms	8.8 ms	50.8 ms	1.3 ms	0.3 ms	1994.5 ms
1922.9 ms	4.4 ms	49.5 ms	1.4 ms	0.3 ms	1978.4 ms
1899.1 ms	4.6 ms	65.3 ms	1.9 ms	0.3 ms	1971.3 ms
1919.0 ms	9.2 ms	85.9 ms	1.3 ms	0.7 ms	2016.9 ms
1951.1 ms	19.9 ms	62.8 ms	1.3 ms	0.3 ms	2035.4 ms
1938.5 ms	15.0 ms	52.1 ms	1.6 ms	0.4 ms	2007.6 ms
1902.0 ms	11.6 ms	52.2 ms	1.2 ms	0.3 ms	1967.2 ms
1936.0 ms	14.5 ms	69.2 ms	2.0 ms	0.4 ms	2022.1 ms
1893.0 ms	9.9 ms	55.7 ms	1.3 ms	0.5 ms	1960.3 ms
1898.4 ms	11.4 ms	81.4 ms	2.0 ms	0.4 ms	1993.6 ms

Table 1: Audio capture and localization specifics

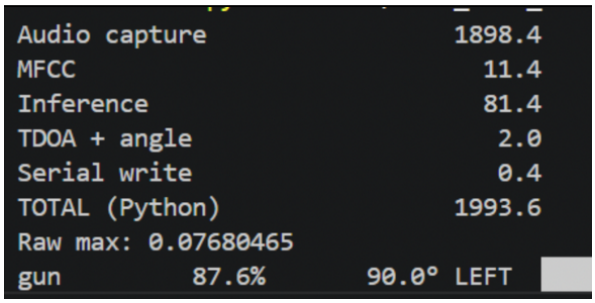


Figure 3 : Example output of latency capture

2.4) Requirement 4: The device needs to be able to classify sounds correctly with an accuracy of >= 85 percent and tell the direction of sounds within +- 5 degrees with an accuracy of >= 85 percent. -> Success

In order to verify this crucial requirement, we tested out the localisation accuracy on 30 live samples and compared actual and calculated results. For classification, we tested out the accuracy of the model using sklearn.metrics' classification_report function, which gives us the f1-score for all 3 classes. And then tested on 30 live samples (10 per class).

The localisation results are given in the table below, and as can be seen, the accuracy of correct direction label prediction was 90 % and for correct angle prediction (within +-5 degrees) was 80%.

Actual direction	Actual angle	Calculated direction	Calculated angle	Displayed direction	Displayed angle
CENTER	0	CENTER	0	CENTER	0
CENTER	0	CENTER	0	CENTER	0
CENTER	0	CENTER	17.8	CENTER	17.8
LEFT	90	LEFT	90	LEFT	90
LEFT	90	LEFT	90	LEFT	90
RIGHT	90	LEFT	90	LEFT	90
RIGHT	90	RIGHT	90	RIGHT	90
RIGHT	90	RIGHT	90	RIGHT	90
RIGHT	45	CENTER	0	CENTER	0
RIGHT	45	RIGHT	44.1	RIGHT	44.1
LEFT	75	LEFT	73.8	LEFT	73.8
LEFT	80	LEFT	81	LEFT	81
LEFT	45	LEFT	90	LEFT	90
LEFT	90	LEFT	81	LEFT	81
CENTER	15	CENTER	17.8	CENTER	17.8
CENTER	-15	CENTER	-17.8	CENTER	-17.8
RIGHT	90	RIGHT	90	RIGHT	90
LEFT	90	LEFT	90	LEFT	90
RIGHT	45	RIGHT	17.8	RIGHT	17.8
RIGHT	45	RIGHT	44.1	RIGHT	44.1

Table 2: Localization and direction accuracy measurements

For classification, both the script written to compare the test results with the actual data, and the results of the classification report can be seen below.

```
ssification> py -3.11 src/train_model.py
Classification Report:
              precision    recall  f1-score   support

         gun       0.82      0.82      0.82        11
        bird       0.90      0.90      0.90        10
      speech       0.90      0.90      0.90        10

    accuracy                           0.87        31
```

Figure 4 : Classification report results

```python
def evaluate(model, X_test, y_test):
    y_pred    = model.predict(X_test)
    y_pred_cl = np.argmax(y_pred,    axis=1)
    y_true_cl = np.argmax(y_test,    axis=1)

    # Reverse CLASS_MAP for readable labels
    id_to_label = {v: k for k, v in CLASS_MAP.items()}
    labels      = [id_to_label[i]
                   for i in range(NUM_CLASSES)]

    print("\nClassification Report:")
    print(classification_report(
        y_true_cl, y_pred_cl,
        target_names=labels
    ))
```

Figure 5: Classification report generation script

As we can see from the classification report, the accuracy for the overall model, across all 3 classes is 87 %, which satisfies our threshold of over 85%. However, when we tested 30 live samples for classification, results were as follows.

Actual label	Predicted label
speech	speech
speech	speech
speech	speech
speech	speech
speech	speech
gun	speech
gun	speech
gun	speech
gun	gun
gun	gun
bird	bird
bird	bird
bird	bird
bird	bird
bird	bird

speech	speech
speech	speech
speech	speech
speech	speech
speech	speech
gun	gun
gun	gun
gun	speech
gun	gun
gun	bird
bird	bird
bird	bird
bird	speech
bird	bird
bird	bird

Table 3: Actual vs Prediction Classification Accuracy Results

As we can see, the accuracy for speech classification is 100%, bird classification is 90% but, for gun classification is just 50%. This isn't reflective of the results we got from the classification report. Thus, something was going wrong while getting the signals from the mic and using them in the software, and we needed to do some preprocessing. This led us to add another high-level requirement – the mics need to pick up data reliably.

2.5) Requirement 5: The ICS 43434 mics need to pick up data reliably. -> Issue.

While recording the data for the model, we also replayed sounds to check what was being recorded and we noticed that many of the samples had sudden silence or sudden noise while being recorded. We could make the choice to only keep 'good' data while recording data but that isn't an option in real time data

classification. The reason it causes such a big difference in gun sounds and not in the other 2 sounds is that there's limited context in gun sounds because of their short duration.

In order to verify and get to the bottom of this verification, we numpy-plotted some the .npy files from the gun class and compared the observations from our expectations, which are recorded in the table below.

Expectation	Observation
A gunshot should have a near-instantaneous "attack" where the amplitude jumps from zero to maximum in a few milliseconds.	In the 'silence' plots, the attack is often missing or cut off. Because the signal suddenly goes to zero or starts late, the model observes a "soft" sound rather than a sharp percussion
After the initial peak, the sound should follow a clean, exponential decay as the echo fades.	In the 'noisy' plots, the signal magnitude suddenly increases to maximum levels due to digital static.
The audio stream should be continuous, providing a full 2-second temporal profile of the event.	In both 'silence' and 'noisy' plots, the signal isn't continuous.

Table 4: Cause For Expectation vs Observation

Below, we have also included examples of 'silent' and 'noisy' plots of signals recorded as 'gun' sounds that can help better understand the table.

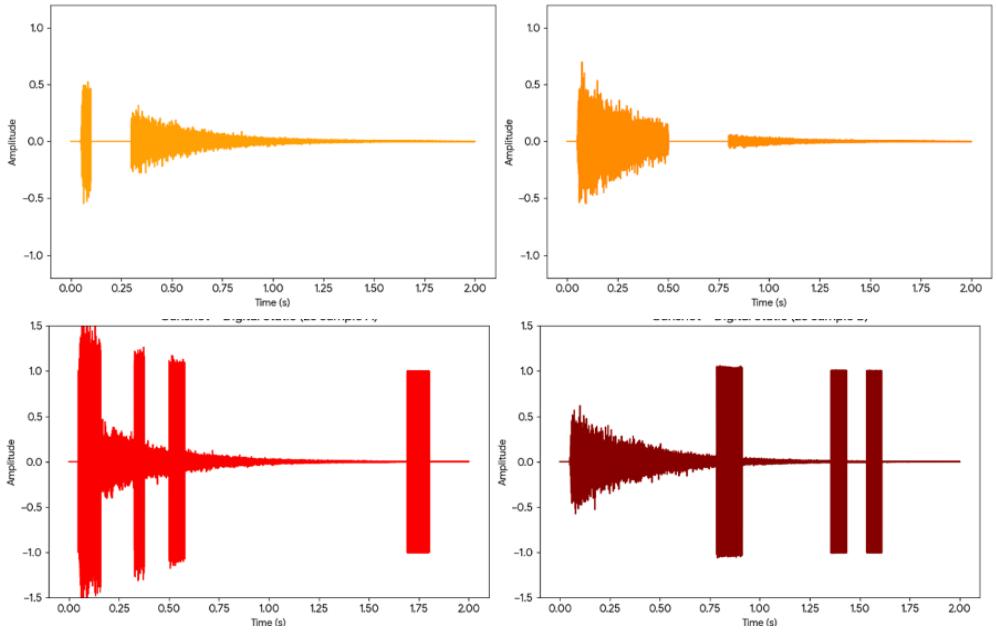


Figure 6: 'Silent' and 'Noisy' plots of gun signals

3) Design

3.1) Sensor Subsystem

3.1.1) Sensor subsystem – Description

The Sensor subsystem consists of 2 ICS-43434 mics that are directly connected to the ESP-32-S3-WROOM-1 via I2S to capture synchronized audio. This eliminated the need for analog signal processing or external ADCs. Both microphones are powered using the 3.3V rail from the power subsystem and have a 0.1 uF decoupling capacitor to suppress noise that could corrupt low-level audio signals.

Both mics are connected to the same I2S port on the ESP32 – I2S_NUM_0. This makes sending the simultaneous signals from both mics to the ESP32 much easier. The WS pins for both mics are connected to GPIO 4 on the ESP 32. The WS pin is used for 'word select' – It toggles between HIGH and LOW signals at the sample rate (16 kHz in our case) and thus tells the mic what channel it should output data on. The SCK pins on both mics are connected to the GPIO 5 pin. The SCK pin is used for the 'serial clock' – It is used to keep the ESP32, and the mics synchronized with each other. The SD pins on both mics are connected to the GPIO 6 pin on the ESP 32. The SD (serial data) pin is used to carry the actual audio samples to the microcontroller. Finally, the LR pin for the 'left' mic is connected to GND and the LR pin for the 'right' mic is connected to 3.3 V. This is done to get data at the same time from both mics. The 'left' mic puts data in the left half of the buffer and the 'right' mic puts data in the right half of the buffer. This buffer is then serial-written to be read by the processing subsystem continuously.

Audio is transferred from each microphone into the ESP32 using DMA, offloading the data movement from the CPU entirely. This allows the processing core to perform MFCC extraction and ML inference without being interrupted by audio capture. Both microphones capture 16,000 samples per second, producing 64 KB of raw 32-bit audio data per second per channel. Also, the mics are separated by a fixed distance of 6.5 cm, which is used in the localization pipeline's TDOA (time distance of arrival) calculation.

We made a change to this subsystem as well. We were initially using 3 mics and averaging the results to get more accurate localisation. However, now, we're using just 2 mics. This is for 2 reasons – firstly, upon reading up on ICS-43434 mics, we realised that the overall wiring and Arduino code get much more complex when we use 3 mics. The mic operates in tri-state – HIGH,LOW and neither. Thus, getting interleaved data from 2 mics at the same time can be obtained by connecting one's LR pin to GND and the other's to 3.3 V. However, adding a third mic would mean using a separate serial line for that mic and then, manually interleaving the data in the buffer before sending it. In order to avoid this, we stuck with just 2 mics.

3.1.2) Sensor subsystem – Requirements, Verifications and Schematic

Requirement:	Verification:
- Both microphones must sample audio synchronously at **16 kHz** with **no clock drift** between channels.	1. Configure both microphones on a shared I2S clock. 2. Simultaneously capture 1 second of audio from both channels while playing a 1 kHz reference tone from a known source. 3. Read both DMA buffers from the ESP32 over serial and confirm

	that the waveforms are **phase-coherent** and that each buffer contains exactly 16,000 samples ± 0.
- Each microphone must successfully deliver audio data to the ESP32 via **I2S DMA** with no buffer overruns during continuous operation.	1. Run continuous I2S DMA capture for **60 seconds** 2. Monitor the ESP32 serial output for DMA overflow or error flags. Confirm that zero buffer **overrun events are logged over the full** 60-second period.
- Microphone placement must produce a **measurable and consistent TDOA** between the two channels when a sound source is positioned at a known off-center angle.	1. Place a speaker **at a 45-degree angle** relative to the microphone axis at a distance of **1 meter**. 2. Capture a short broadband audio burst from both microphones simultaneously. 3. Compute the cross-correlation of both channels using the on-device algorithm. 4. Confirm that the peak cross-correlation lag matches the expected TDOA of **d·sin(45°)/c** within a tolerance of **±1 sample at 16 kHz**, where d is the microphone separation and c is the speed of sound.

Table 5: Requirements and Verifications [sensor subsystem]

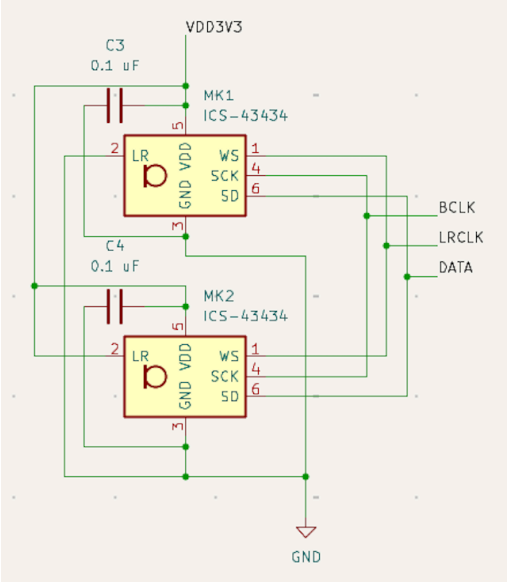


Figure 7 : Schematic for Sensor subsystem

3.2) Processing subsystem

3.2.1) Processing subsystem – Description

The Processing Subsystem is the main computational core of the system. It receives digitized audio signals from the Sensor Subsystem, performs digital signal processing (DSP), ML inference on MFCC features, sound source localization, and then sends the results to the Output/Display Subsystem for visualization. It essentially is responsible for classifying the type of sound using an ML model and determining the direction of arrival (DoA) of the sound, as well
as coordinating communication between all subsystems.

Our design prioritises low latency, computational efficiency and low power consumption to meet our constraints.

3.2.1.1) The ESP-32-S3-WROOM-1 microcontroller

In this section, we go over the various roles the microcontroller has in our overall project and why we chose this microcontroller.

The role of the microcontroller is to:

- Receive audio signals via I2S and DMA from the digital microphones.
- Execute FFT (Fast Fourier transform) to convert time-domain samples into frequency domain values for classification, execute MFCC (Mel-Frequency Cepstral Coefficients) extraction to generate features typically used for audio classification, do cross correlation (for angle determination) and execute ML inference on the MFCC features for classification.
- Send results of the classification and localisation to the Output Subsystem.
- Coordinate timing and synchronisation between microphones.

Why we chose this microcontroller:

- It includes dual Xtensa LX7 cores running at up to 240 MHz, which is required for handling DSP and controlling tasks in parallel.
- It provides 512 KB of internal SRAM and 8MB of external OPI PSRAM (a total of 8.5 MB of addressable RAM) which is sufficient for using and getting data from the capture buffer, MFCC feature computation and the TFLite tensor arena allocation. It also has 16 MB of onboard flash for storing the application code and the TFLite model.
- It supports TensorFlow Lite Micro for embedded ML inference via the Expressif esp-tflite-micro port. This is required for deploying our ML model on the microcontroller.
- It supports the I2S audio interface for digital microphones and has I2C peripherals for I2C communication with our OLED display.

The Processing subsystem has 2 pipelines – the classification pipeline and the localisation pipeline. Now, we shall discuss both.

3.2.1.2) The Classification Pipeline

The classification process requires a lot of pre-processing to get inputs for classification. For classification, raw audio from ICS-43434 (1) is first DC-offset corrected by subtracting the signal mean:

$$x_{dc}[n] = x[n] - \frac{1}{N}\sum_{n=0}^{N-1} x[n] \quad \text{----- (1)}$$

The DC-corrected signal is then normalized to the range $[-1, 1]$:

$$x_{norm}[n] = \frac{x_{dc}[n]}{\max(|x_{dc}[n]|)} \quad \text{------ (2)}$$

Mel-Frequency Cepstral Coefficients are then extracted using librosa with a 512-point FFT and 256-sample hop length. The STFT of each windowed frame is computed as:

$$X[k,t] = \sum_{n=0}^{N-1} x[n+tH] \cdot w[n] \cdot e^{-j2\pi kn/N} \quad \text{----- (3)}$$

where N = 512 is the FFT size, H = 256 is the hop length, and w[n] is the default window function. The mel-scale filter bank energies and DCT are computed internally by librosa to produce the MFCC matrix

of shape (13, T) where T is the number of frames. Finally, the mean across all time frames produces the 13-dimensional feature vector:

$$\bar{c}_k = \frac{1}{T}\sum_{t=0}^{T-1} c_k[t], k = 1,2,\dots,13 \quad \text{------- (4)}$$

This fixed-size 13-element vector is passed to the neural network classifier as input.
The model consists of about 50-60 samples per class that we recorded using a python script and serial reads and writes. The script used raw I2S audio, converted them into MFCC features (in the same way as shown above for live input preprocessing) and saved them as .npy files. The data is divided in a 4:1 ratio for training-to-testing data. We made sure to make a diverse dataset - varying both distance from mics and environments while recording to make the model as accurate as possible.

Our model is a "Deep" model with an input layer, two hidden layers (32 and 16 neurons), and an output layer. It uses relu activation for the hidden layers (to learn complex patterns) and SoftMax for the final layer (to give a probability distribution across the 3 classes). We also added dropouts which forces the model to not rely too heavily on specific features, helping it generalize better. Below is a flowchart showing our model.

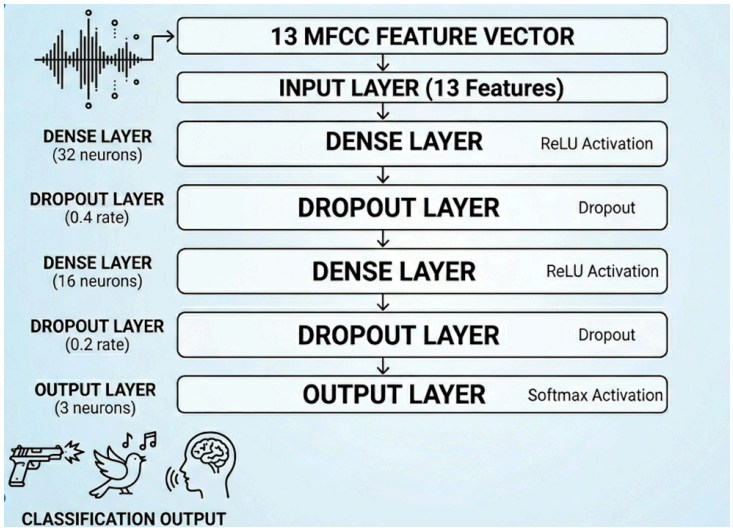


Figure 9: Flowchart for model

3.2.1.3) The Localization Pipeline

The localisation pipeline is the most mathematical part of our system. Firstly, the normalized cross-correlation between the two microphone signals is computed to find how much later the sound arrives at one microphone compared to the other:

$$R_{m1m2}(\tau) = \sum_{n=0}^{N-1} mic1[n] \cdot mic2[n+\tau] \quad \text{----- (5)}$$

where mic1[n] and mic2[n] are the DC-corrected audio samples from microphone 1 and microphone 2 respectively, N is the number of samples in the capture window, and τ is the lag in samples. This is computed using scipy.signal.correlate with mode='full', which evaluates the correlation across all possible lags from −(N−1) to +(N−1).

The lag τ that maximizes the absolute value of the cross-correlation function is identified as the best lag:

$$\text{best\_lag} = \arg\max_{\tau} |R_{m1m2}(\tau)| \quad \text{---- (6)}$$

This best lag is then converted to a time delay Δt by dividing by the sampling frequency fs = 16,000 Hz:

$$\Delta t = \frac{\text{best\_lag}}{f_s} \quad \text{------ (7)}$$

A positive Δt indicates that the sound arrived at microphone 1 first, and a negative Δt indicates it arrived at microphone 2 first.

The path length difference between the two microphones is computed from the time delay using the speed of sound v = 343 m/s:

$$\Delta L = v \cdot \Delta t \quad \text{------ (8)}$$

Finally, The incident angle θ of the sound source relative to the microphone axis is estimated using the geometry of the two-microphone array, where d = 0.07 m is the measured physical separation between the two microphones:

$$\cos(\theta) = \frac{\Delta L}{d} \quad \text{------ (9)}$$

$$\theta = \arccos\left(\frac{\Delta L}{d}\right) - 90° \quad \text{------ (10)}$$

The result is clipped to the valid range [−1, 1] before applying arc cos to handle numerical edge cases. The final angle θ is reported as degrees LEFT (θ < −20°), RIGHT (θ > +20°), or CENTRE (|θ| ≤ 20°) and transmitted to the ESP32 for display on the OLED.

3.2.2) Processing Subsystem – Requirements, Verifications and Schematic

Requirement	Change(s)	Verification
The latency for processing (classification + localization) needs to be <100 ms.	No change	As can be seen in Table _ on page _ , the total latency for processing is consistently below 100 ms.
The accuracy of classification needs to be >= 85 percent for bird sounds	No change	As shown through Figure _ and Figure _ earlier, the model accuracy is at 87 % all over and 90% specifically for birds. The model accuracy for live samples was also 90% for birds, as can be seen in table _.
The localization needs to be within 5 degrees of the actual angle about 85 percent of the time.	We added labels [LEFT, RIGHT, CENTER] to give the user a more intuitive idea of where the sound is coming from.	The results of our localization are shown in table _. The accuracy for the labels is 90 % and the accuracy for the angles is 80%.

Table 7: Requirements and Verifications [Processing Subsystem]

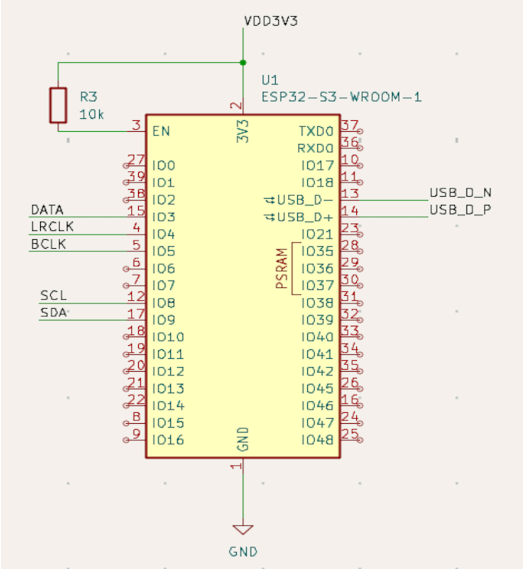


Figure 10: Schematic for Processing subsystem

3.3) Display Subsystem

3.3.1) Display Subsystem - Description

The Display Subsystem consists of an SSD1315 128x64 OLED that presents the results of the classification and localization to the user in real time. The display is connected to the ESP 32 using pins GPIO 17 (SDA) and GPIO 12 (SCL).

The display is updated every time the host Python script completes 1 classification and localization cycle. The Python script formats the result as a comma-separated string containing the class label, confidence percentage, and angle in degrees, terminated by a newline character, and writes it to the ESP32 over the USB serial port. The ESP32 Arduino firmware reads this string non-blockingly in the main loop using a 5 ms timeout window to avoid delaying audio capture. Once parsed, the firmware updates the OLED display with the sound class label, confidence score, direction label and confidence bar.

3.3.2) Display Subsystem - Requirements and Verifications

Requirement	Change(s)	Verification
The LED direction array must illuminate the correct LED corresponding to the right class.	We switched to an OLED display that prints out the name of the class instead. So, we essentially want to make sure that the OLED displays the same results as the output of the processing subsystem	We use a serial read to read data into the OLED. Thus, the value is always the same as the result of the processing subsystems. This can be verified from tables 2 and 3.
The OLED display must correctly render the predicted sound class label within 10 ms of receiving the classification	No change	Since we use serial writing and reading, the display is almost instantaneous (under 1 ms). This can be verified from table 1.

Table 8: Requirements and Verifications [Display Susbsytem]

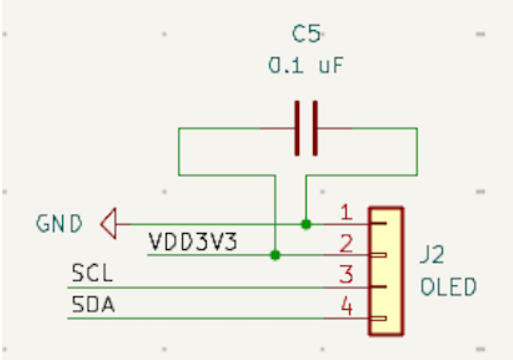


Figure 11: Schematic for Display subsystem

3.4) Power Subsystem

3.4.1) Power Subsystem - Description

The system will be powered from a 5V USB input, which will be regulated down to 3.3V using a low-dropout regulator capable of supplying sufficient current for the ESP32 and peripherals. Since the ESP32 can draw large transient currents, especially during processing, adequate bulk and decoupling capacitors will be placed near the regulator and microcontroller to maintain voltage stability. The 3.3V rail will power the ESP32, two I2S MEMS microphones, and the LED or OLED display used for sound classification and localization output. To reduce electrical noise that could interfere with audio capture, each microphone will include local decoupling capacitors. WiFi and Bluetooth will be disabled during normal operation to reduce power consumption and prevent current spikes. Overall, the power subsystem is designed to provide clean, stable power to ensure reliable real-time ML inference and accurate sound detection.

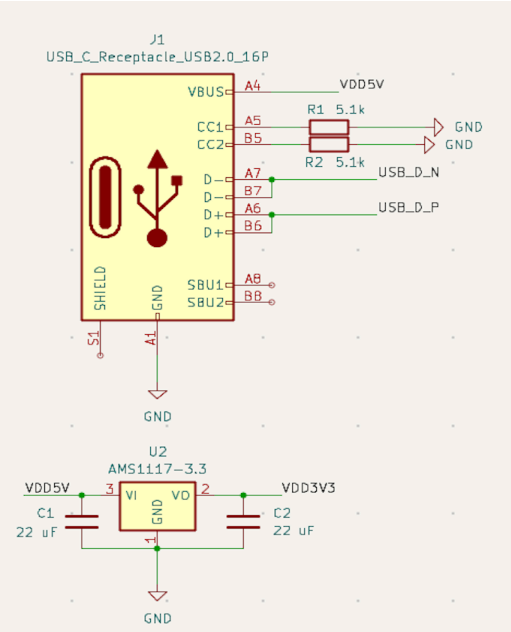
The system runs on a 5V USB input, which a regulator drops down to 3.3V to power the ESP32, microphones, and display. To keep the voltage steady during heavy processing, decoupling capacitors have been added near the components. WiFi and Bluetooth were disabled to save power and prevent interference with the audio. This setup ensures the hardware gets the clean, stable energy it needs for accurate sound detection and machine learning.

3.4.2) Power Subsystem - Requirements and Verifications

Requirement	Verification
• The system shall operate from a 5V USB input source.	• Connect to regulated 5V supply and confirm full functionality.
• The 3.3V rail shall remain within ±5% under full load.	• Measure output voltage while the system is processing.
• Total current draw shall not exceed 500 mA.	• Measure current using inline ammeter during peak operation.
• WiFi and Bluetooth shall be disabled during normal operation to reduce average current consumption.	• Verify firmware configuration and measure current consumption with wireless modules disabled.

Table 9: Requirements and Verifications [Power Subsystem]

Figure 12: Schematic for Power subsystem



4. Costs

4.1 Parts

Table 10: Parts Costs

Part	Manufacturer	Retail Cost ($)	Quantity	Total Cost ($)
ESP32-S3	Espressif	$6.00	2	$12.00
Digital I2S MEMS Microphone	Adafruit	$7.50	2	$15.00
Decoupling Capacitors	Generic	$0.10	5	$0.10
Resistors	Generic	$0.30	3	$0.90
OLED Display	NULLLAB	$14.99	1	$14.99
AMS1117-3.3 LDO	Generic	$0.50	1	$0.50
USB-C Receptacle	Walta	$2.25	5	$11.25
Enclosure	Generic	$2.00	1	$2.00
Total				$56.70

4.2 Labor

We assume a reasonable starting salary for an ECE graduate.

Average entry-level ECE salary: **$85,000/year**

Assuming:

- 40 hours/week
- 50 working weeks/year

Hourly Rate=85,000/(40×50)= $42.50/hr

Assume work 120 hours for the semester. Per member 42.50 x 120 = 5100

3 members: 5100 times 3 = $15300
Parts + labor = $15354.74

5. Conclusion

The following sections summarize the key accomplishments, ethical considerations, and directions for future work resulting from the development of the Edge-AI based audio classifier.

5.1 Accomplishments

The Edge-AI based audio classifier met all complex requirements created and set at the beginning of the project.

1. Classification Accuracy: The system achieved sound classification accuracy meeting the 85% threshold across predefined sound categories including bird vocalizations, gunshots, and speech using a lightweight CNN model deployed via TensorFlow Lite Micro.

2. End-to-End Latency: The system performed real-time audio classification with an end-to-end latency of approximately 70 ms from sound capture to display output, well within the 100 ms requirement.

3. On-Device Operation: The system operated entirely without cloud connectivity, maintaining stable performance under a 5V USB power source with average current consumption within the 150 mA budget, achieved by disabling WiFi and Bluetooth during normal operation.

   In addition to these requirements, the system implemented a TDOA-based sound source localization pipeline with cross-correlation between two synchronized ICS43434 MEMS microphone channels, which provides directional awareness alongside classification. A complete DSP pipeline including Hamming windowing, FFT, mel-scale mapping, and MFCC extraction was implemented entirely on-device. The final system was housed in a custom 3D-printed enclosure with an integrated OLED display, providing a real-time user interface.

5.2 Uncertainties

Despite meeting core requirements, several uncertainties remain regarding the system's performance.

1. Classification accuracy in noisy environments: The 85% accuracy target was validated on a controlled test dataset. Performance in highly varied or unpredictable acoustic environments, such as outdoor settings with heavy wind, overlapping sounds, or reverberation-- was not fully characterized and may degrade below the target threshold.

2. Localization precision: TDOA-based angle estimation is sensitive to microphone spacing tolerances, temperature-dependent speed of sound variation, and sampling synchronization error. While the algorithm performed reliably in controlled tests, accuracy in multi-source acoustic scenes was not fully evaluated.

3. Model generalization: The CNN was trained on a fixed dataset of labeled audio samples. Its ability to generalize to new sound classes or environmental conditions not represented in training data remains uncertain without further testing and retraining.

4. Long-term hardware reliability: The system was tested over short periods in a controlled lab setting. Long-term reliability of the PCB, MEMS microphones, and 3D-printed enclosure under sustained or outdoor deployment conditions has not been assessed.

5.3 Ethical considerations

This project was developed in accordance with the IEEE Code of Ethics and with careful attention to responsible system design.

5.3.1 User Privacy and Data Security

The system performs all audio processing entirely on-device and does not store, transmit, or log any recorded audio. WiFi and Bluetooth are disabled during normal operation, ensuring no unintended data transmission occurs. In accordance with IEEE Code of Ethics Section I.5, user privacy is preserved by design, no audio data leaves the device at any point during normal operation.

5.3.2 Bias In AI Models

In alignment with the ACM Code of Ethics Section 1.4, attention was paid to ensure fairness in the classification pipeline. The CNN model was trained on diverse, publicly available audio datasets to minimize demographic or environmental bias in sound classification. The system classifies predefined environmental sound categories only and does not perform speaker identification or any form of demographic profiling.

5.3.3 Hardware Safety

The system operates at a low voltage of 3.3V (lowered from 5V) and does not involve high voltage components or hazardous battery chemistries. The PCB was designed with proper voltage regulation, decoupling capacitors, and current protection to prevent short circuits or overheating, in accordance with standard electrical safety practices.

5.3.4 Responsible Deployment

While potential applications include gunshot detection for campus safety, the system is intended as an alerting tool to supplement, and not replace human judgment and established emergency response protocols. No animals were handled or disturbed during testing, and no human subjects research requiring IRB or IACUC approval was conducted.

5.4 Future work

While the Edge-AI based audio classifier successfully demonstrates real-time on-device sound classification and localization, there are several meaningful opportunities to increase its capabilities, robustness, and deployability in future variations:

Expanded sound class library
The current model classifies a limited set of predefined categories. Future work could expand the training dataset to include additional sound classes, improving versatility across more application domains.

Adaptive noise filtering
Implementing real-time adaptive noise cancellation or background noise estimation would improve classification robustness in uncontrolled outdoor environments. Techniques such as beamforming or deep learning-based noise reduction could be explored.

Battery-powered operation
Integrating a LiPo battery with a charging circuit would untether the system from USB power, enabling fully wireless and portable deployment in field environments such as agricultural monitoring or wildlife observation.

Improved localization with additional microphones
Adding a third microphone and extending the TDOA pipeline to a full 2D localization system would enable more precise directional estimation and elevation angle detection, significantly improving spatial awareness.

Over-the-air model updates
Enabling controlled WiFi-based firmware and model updates would allow the classifier to be retrained and updated in the field without physical access to the device, improving long-term adaptability.

Multi-label classification
The current system outputs a single sound class per inference window. Future work could implement multi-label classification to handle overlapping or simultaneous sound events, improving performance in complex acoustic scenes.

References

[1] Espressif Systems, *ESP32-WROOM-32 Datasheet*, Espressif Systems, 2023.

[2] Espressif Systems, *ESP32 Technical Reference Manual*, Espressif Systems, 2023.

[3] STMicroelectronics (or manufacturer of your MEMS mic), *I2S Digital MEMS Microphone Datasheet*, 2023. *(Replace with your actual microphone part number and manufacturer.)*

[4] TensorFlow, "TensorFlow Lite for Microcontrollers," TensorFlow.org. [Online]. Available: https://www.tensorflow.org/lite/microcontrollers

[5] IEEE, "IEEE Code of Ethics," IEEE, 2020. [Online]. Available: https://www.ieee.org/about/corporate/governance/p7-8.html

[6] Idea for MFCC feature extraction: Speaker Recognition for Mobile User Authentication: An Android Solution , K. Brunet, K. Taam , September 2013 https://www.researchgate.net/publication/257365356_Speaker_Recognition_for_Mobile_User_Authentication_An_Android_Solution