SYSTEM DESIGN LAB

2024-2025

PROJECT REPORT

Sound Classification using RaspberryPi Zero 2W

Done by,

Pallavi Alanka - 24CS4508

Nataraj Gowd Velagana - 24CS4511

Madabattula Venkatesh - 24CS4513

Pfokreni Adahrii - 24CS4501

1. Introduction:

In the modern era of smart systems and automation, the ability of machines to perceive and understand sound has become a fundamental component of intelligent environments. From voice assistants and surveillance systems to smart homes and healthcare monitoring, **sound classification** has opened new avenues for interaction between humans and machines. This project focuses on developing a **real-time environmental sound classification system** using the **Raspberry Pi Zero 2 W**, a compact and cost-effective edge computing device.

Environmental sounds carry rich contextual information that can be harnessed to identify specific activities, detect anomalies, and trigger automated actions. For example, detecting a **baby crying** could prompt parentally alerts, while recognizing a **dog bark** or **human conversation** could be used in security or caregiving scenarios. The challenge, however, lies in capturing and classifying these sounds accurately in real-world, noisy environments using resource-constrained hardware.

2. Problem Statement:

In a world that is increasingly moving toward automation and smart environments, the ability for machines to understand and respond to audio cues is becoming crucial. Sound is a rich source of contextual information - unlike visual data, it does not require a direct line of sight and can capture activities happening in different spatial directions. Despite its advantages, sound classification, particularly on low-cost and low-power embedded systems, remains a significant technical challenge.

The core problem addressed in this project is:

How can a low-power embedded device like the Raspberry Pi Zero 2W be used to accurately detect and classify real-time environmental sounds, such as a baby crying or a dog barking, in a resource-efficient and robust manner?

2.1. Challenges Involved :

2.1.1. Hardware Limitations:

The Raspberry Pi Zero 2 W is a compact microcomputer with limited CPU, memory, and GPU resources. It cannot afford to run large, resource-intensive models commonly used in commercial sound classification systems.

2.1.2. Environmental Noise:

Audio captured in real-world environments is often contaminated by background noise (fans, wind, human chatter), making it difficult to isolate and classify the actual sound events.

2.1.3. Variability in Sounds:

Sounds like a baby cry or a dog bark can vary greatly in pitch, duration, and intensity. Capturing this variability requires a diverse and augmented dataset, as well as models that generalize well.

2.1.4. Real-Time Constraints:

The system must process incoming audio, extract features, classify the sound, and deliver predictions within a short time window (~1–2 seconds).Balancing speed and accuracy is essential for real-time deployment.

2.1.5. Deployment and Accessibility:

The system should not only work in controlled lab settings but also be easy to deploy in real-life scenarios (e.g., homes, hospitals, surveillance setups).A user-friendly interface is necessary for non-technical users to understand the outputs and take action.

2.2 <u>Objectives of the Project:</u>

To address these challenges, the project sets out to achieve the following :

- To design a lightweight, accurate machine learning model for classifying common environmental sounds.
- To collect and preprocess audio data for relevant classes (e.g., dog bark, baby cry, bird chirp).
- To implement real-time audio acquisition using a USB sound card with the Raspberry Pi.
- To extract meaningful features using signal processing techniques like MFCCs and Mel spectrograms.
- To augment the dataset using pitch shifting, time shifting, noise addition, and other techniques to improve robustness.
- To deploy the trained model on the Raspberry Pi Zero 2W using TensorFlow Lite to ensure efficient inference.

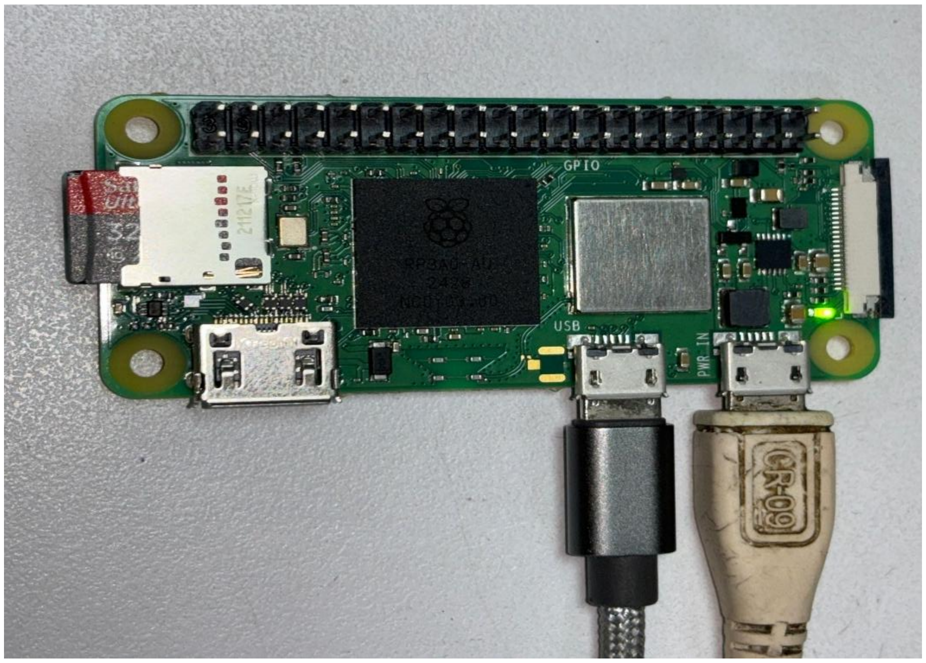
- To develop a web-based dashboard to visualize real-time sound predictions and confidence levels.

3. <u>Prototype Description:</u>

The prototype for the Real-Time Sound Classification System is designed using the Raspberry Pi Zero 2 W, a compact and affordable single-board computer. The entire system is engineered to perform the acquisition, processing, classification, and display of environmental audio signals in real time, directly on the edge device.

3.1. <u>Hardware Components:</u>

a. <u>Raspberry Pi Zero 2 W</u>



As shown in the image, the Raspberry Pi Zero 2 W is the core computing unit. It features:

- Quad-core 64-bit ARM Cortex-A53 processor @ 1 GHz

- 512MB RAM
- MicroSD card slot (used to boot OS and store audio/model files)
- Mini HDMI port
- Micro-USB ports: One for power and another for USB peripherals (used for audio interface)

The green LED near the power input in the image confirms that the board is powered and operational.

<u>b. USB Audio Card</u>

Since the Raspberry Pi Zero 2 W does not have a built-in audio input, an external USB sound card is connected through the micro-USB OTG adapter (as seen in the bottom USB port). This enables audio input from a microphone.

<u>c. Power Supply</u>

The device is powered via the top micro-USB power port connected to a USB charger or a portable power bank, making it suitable for mobile or field deployment.

d. <u>MicroSD Card</u>

Inserted into the slot, it holds the Raspberry Pi OS, Python scripts, trained models, and recorded audio data.

3.2. **<u>Software Components:</u>**

a. <u>Raspberry Pi OS</u>

Installed using Raspberry Pi Imager on a microSD card. The OS provides a lightweight, Linux-based environment for development and execution.

b. <u>Audio Acquisition and Recording</u>

Using the arecord utility:

```
arecord -D plughw:1,0 -f cd -t wav -d 5 output.wav
```

This records 5 seconds of real-time sound from the connected USB microphone.

c. <u>Feature Extraction</u>

- MFCC (Mel-Frequency Cepstral Coefficients): Extracts 13 key coefficients that represent timbral and tonal qualities.
- Mel Spectrogram: Provides a 2D time-frequency representation for model input.

d. <u>Deep Learning Models</u>

- CNN and LSTM models trained on augmented audio datasets.

- Trained using TensorFlow on a development system, then converted to TensorFlow Lite for efficient inference on Raspberry Pi.

e. Real-Time Inference Script

A Python script loads the trained TFLite model and continuously:

- Captures audio input
- Processes features (MFCC/spectrogram)
- Predicts class labels (e.g., Dog Barking, Baby Crying)
- Sends results to a local or remote web interface

3.3. **Web-Based User Interface:**

To visualize predictions:

- A Flask web server runs on the Pi.
- Prediction results are pushed to the browser in real-time.
- The dashboard shows:

  Detected class , Confidence score , Timestamp

3.4. **Working Prototype Flow:**

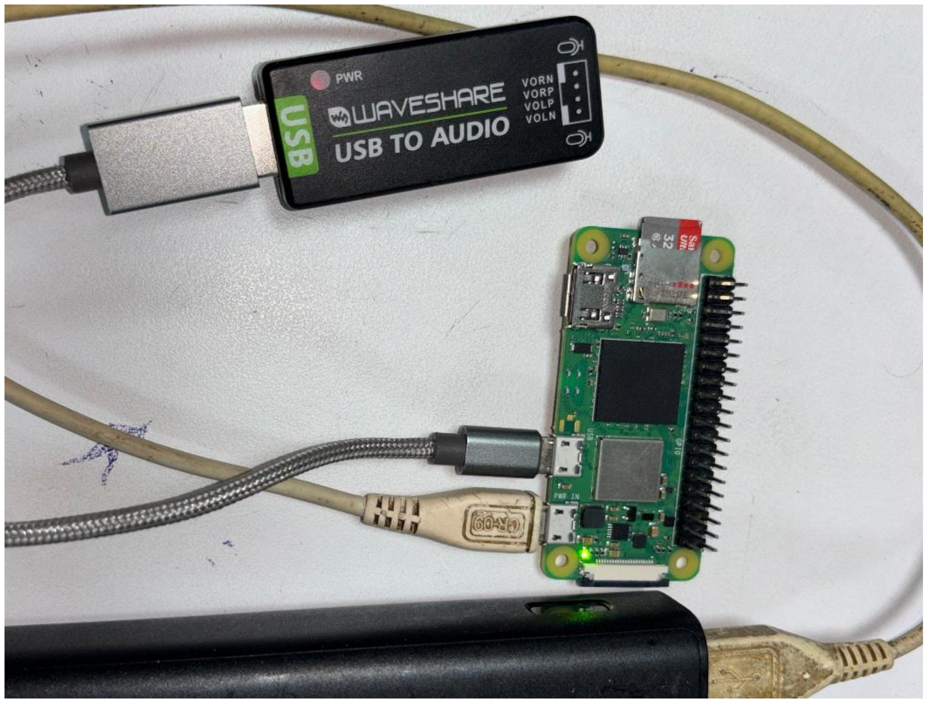
1. Boot the Raspberry Pi with SD card and connected power.
2. Connect USB audio card to the lower micro-USB port via OTG adapter.
3. Record audio input from the microphone.

4. Extract features (MFCC/Mel spectrogram).

5. Perform inference using the lightweight model.

6. Display results on the Flask-based web interface.

3.5. <u>Physical Setup Overview:</u>



From the image:

- The Pi is connected to power (top micro-USB).
- USB OTG cable is connected to the bottom micro-USB, which interfaces with the sound card.
- The system is compact, portable, and capable of running headless (no display or keyboard needed).

4. Data Acquisition Process:

This sound classification system identifies seven distinct environmental and human-related audio classes along with a catch-all 'No Class' category:

Class Name	**Description**
Flush	Sound of a toilet or water flush
Human Talking	Conversational speech or vocal interactions
Basin Tap	Flow of water from a bathroom/kitchen sink tap
Walker	Sound of a walking aid (like a walker or cane)
Bird Chirping	Natural bird calls or chirping sounds
Child Crying	Infant or toddler crying sounds
Dog Barking	Barking or howling of a dog
No Class	Background noise or silence (used for training balance)

The Raspberry Pi was used as an edge device to collect and process audio data. Below are the steps taken to remotely access the Pi, record sound, and transfer files between the Pi and a Windows PC.

1. Remote Access to Raspberry Pi. This command connects to the Raspberry Pi over a local network using SSH.

```
ssh pi@192.168.143.152
```

2. Check Available Recording Devices

```
arecord -l
```

Lists audio capture devices connected to the Pi.

3. Record Audio

```
arecord -D plughw:1,0 -f cd -t wav -d 300 dog3.wav
```

Records a WAV audio sample using CD-quality settings.

4. Transfer Audio to Windows PC

```
scp pi@192.168.143.152:/home/pi/dog.wav
C:\Users\adash\OneDrive\Desktop
```

Copies the audio file from Raspberry Pi to the local desktop.

Class Name	**Number of Samples**	**Total Duration (Secs)**
Bathroom Flush	300	2700 secs
Human Talking	300	2700 secs
Basin Tap	300	2700 secs
Walker	300	2700 secs
Bird Chirping	300	2700 secs
Child Crying	300	2700 secs
Dog Barking	300	2700 secs
No Class	-	0 secs (Used threshold)
Total	2100	18,900 secs /315 mins

5. Methodology:

5.1. Data Augmentation :

In real-world scenarios, audio recordings are rarely collected under controlled, noise-free conditions. To make a machine learning model robust to such variations, data augmentation is a crucial technique. It artificially increases the size and diversity of the training dataset by applying transformations that simulate real-world variability. The code provided applies four effective augmentation techniques using Python audio libraries like Librosa and NumPy. Each augmentation preserves the class label while slightly altering the signal to introduce diversity.

Implemented Augmentation Techniques:

Technique	Purpose	Implementation
White Noise	Simulates background sounds (e.g., wind, traffic) to improve robustness	Adds Gaussian noise to audio
Pitch Shifting	Models' slight tonal differences (e.g., child vs adult voice)	Uses Librosa to shift pitch up/down
Time Shifting	Helps model tolerate temporal variations in event onset	Rolls audio waveform forward/backward

Each audio file is augmented with all four techniques, resulting in 4 new samples per original audio file, helping prevent overfitting and boosting model generalization.

5.2. Data Preprocessing:

Before feeding the audio data into the machine learning model, it is essential to ensure that all the audio files are in a consistent format. This is particularly important in deep learning applications, where models expect input data of uniform dimensions and characteristics. The primary aim of preprocessing is to prepare the dataset in such a way that each file has the same sample rate, length, and format, which facilitates smooth training and evaluation processes.

5.2.1. Uniform Sampling Rate

All audio files were first resampled to a fixed sampling rate of 16,000 Hz (16 kHz). This ensures that the temporal resolution of each audio sample is consistent across the dataset. A consistent sampling rate is crucial to accurately compare audio features and ensure that time- and frequency-based representations (such as spectrograms and MFCCs) are meaningful.

5.2.2. Identifying the Longest Audio Duration

Each audio file in the dataset may vary in duration. To standardize them, we first analysed the entire dataset to identify the maximum duration among all the files. This longest duration serves as a benchmark for the rest of the preprocessing process.

5.2.3. Padding and Trimming Audio Files

Once the maximum duration was established:

- Files longer than this maximum duration were trimmed to match the benchmark length. This ensures that any trailing silence or noise beyond the expected duration is removed.

- Files shorter than the maximum duration were padded with silence (zero-amplitude values) at the end to make them equal in length. Padding ensures that no valuable information is lost while aligning the length with the longest file.

By applying these adjustments, every audio file in the dataset becomes equal in duration, regardless of its original length.

5.2.4. Format Consistency

All files were converted to a common audio format (WAV), and their sample rate and channel configuration were normalized. This reduces the risk of processing errors during feature extraction and model training.

5.2.5. Metadata-Based Processing

To maintain consistency with class labels and data organization, we utilized a metadata file that stored the path and label information for each audio file. During preprocessing, this metadata ensured that each file was accurately matched to its class and stored in the correct directory.

5.2.6. Saving the Final Dataset

The processed audio files were saved into a dedicated output directory. This finalized dataset is now ready for feature extraction, model training, and evaluation. Since all files share the same sampling rate and length, they are compatible with batch-based training techniques and real-time inference models.

- Ensuring high-quality audio recordings is a critical step before performing any kind of feature extraction or training in a sound classification pipeline. Poor-quality audio can result in loss of valuable acoustic features, thereby degrading the model's ability to classify sounds accurately. During the data collection phase using Raspberry Pi and a USB microphone, several **challenges** related to sound quality were encountered.

Challenges Faced in Raw Audio Data:

1. **Background Noise:** Recordings often captured ambient sounds like fans, distant conversations, or street noise. These unintended sounds introduce unwanted variance into the dataset, making it harder for the model to focus on the actual target sound class.

2. **Low Amplitude (Quiet Sounds):** Some recordings, especially those of softer sounds like bird chirping or distant human speech, had significantly low volume levels. This made them harder to distinguish from background noise.

3. **Distorted Recordings:** Improper microphone sensitivity or loud inputs (like bathroom flush or baby crying) sometimes resulted in clipped or

distorted audio waveforms. These distortions affect spectral features and may confuse the model.

Enhancements and Solutions Implemented

To address these challenges and improve the quality of recorded sound clips, several **audio signal processing techniques** were applied. These improvements were made using Python libraries such as **librosa**, **scipy**, and **pydub**.

1. High-Pass Filtering (Noise Reduction):

A **high-pass filter** was applied to each recording to eliminate low-frequency background hums (like electrical noise or air conditioning). This type of filter allows frequencies above a certain threshold (e.g., 100 Hz) to pass while attenuating the lower ones.

- **Benefit:** Removes continuous low-frequency noise without affecting the primary sound (e.g., barking, crying, etc.).
- **Library Used:** SciPy. Signal (for digital filtering using Butterworth filters).

2. Amplitude Normalization: The recordings were normalized to ensure consistent loudness across different clips. Low-volume audio was boosted while keeping louder sounds within acceptable dynamic range.

- **Technique:** Logarithmic amplitude scaling was performed, ensuring each audio clip has a consistent RMS (Root Mean Square) value.

- **Library Used:** pydub for gain control and normalization; librosa.effects.preemphasis() to boost high frequencies and reduce spectral tilt, which improves clarity.

3. Spectral Denoising: To reduce background hiss or environmental noise, **spectral gating** or **bandpass filtering** techniques were applied. These methods identify frequency bands with consistent noise and suppress them while preserving important frequency components of the target sound.

- **Spectral Gating:** Removes noise by thresholding the short-time Fourier transform (STFT) magnitude.

- **Bandpass Filtering:** Allows only frequencies in a desired range (e.g., 300–3000 Hz for speech) to pass, cutting off extreme low and high frequencies.

- **Library Used:** librosa and SciPy. Signal (for filtering design and application).

The preprocessing step plays a foundational role in ensuring the success of the overall sound classification pipeline. By converting the entire dataset into a standardized format, we eliminate inconsistencies and prepare the data for efficient feature extraction and deep learning-based classification. These enhancements helped make the dataset more robust and consistent, which in turn improved the overall accuracy and generalizability of the classification model.

5.3. **<u>Feature Engineering:</u>**

To enable the deep learning model to understand and classify sounds, raw audio signals must be converted into structured numerical representations known as audio features. These features are designed to capture the timbral, temporal, and spectral characteristics of audio signals. In this project, we implemented multiple audio feature extraction techniques using the Librosa library, which specializes in audio analysis and signal processing.

Types of Audio Features Extracted:

1. MFCC (Mel Frequency Cepstral Coefficients)

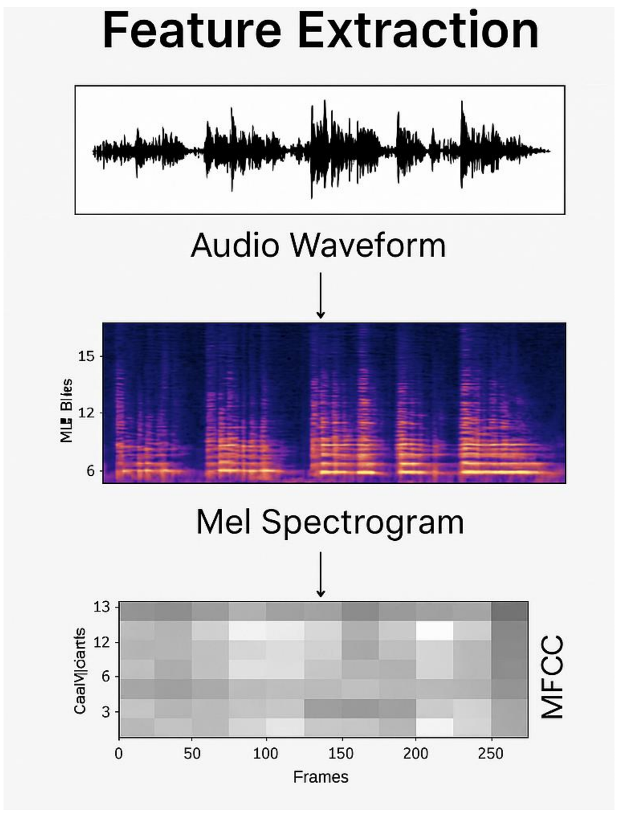
- Description: MFCCs mimic the human ear's perception of sound by mapping frequencies to the Mel scale.
- Implementation: For each audio file, 13 MFCCs were extracted over a fixed duration of 9 seconds.
- Shape Fixing: Each MFCC output was padded or trimmed to 260 frames to maintain a uniform input size (13 x 260).
- Use: Captures speech and environmental sound features essential for classification.

2. Mel Spectrogram

- Description: A time-frequency representation showing how the spectral density of audio varies over time on the Mel scale.
- Implementation: 128 Mel filter banks were used to generate Mel spectrograms for each audio clip.

- Shape Fixing: Spectrograms were padded or cropped to ensure a consistent size (128 x 260) across all samples.
- Use: Retains rich frequency details and is particularly useful for non-speech sound classification.

Dimension	**Meaning**	**Value**	**Reason**
13	MFCC coefficients (frequency features)	13	Captures spectral shape of each time frame
260	Time frames	260	Ensures uniform time length across all samples
1	Channels	1	Required for CNN input; represents single audio stream



3. Amplitude Feature

The raw waveform amplitude of each sample was retained and analysed to understand sound energy and loudness patterns. This helped in:

- Identifying sharp transitions like sudden bursts (e.g., toilet flush).
- Measuring signal intensity variations over time.

Spectrograms, and particularly Mel spectrograms, offer a **visual fingerprint** of sound events. Each spectrogram was treated as a 2D image input (128 x 260 x 1) for CNN-based training. The spectrograms were converted to a decibel (dB) scale using **power_to_db()** to improve dynamic range and model interpretability.

5.4. Model Used:

In this project, we designed and implemented Convolutional Neural Network (CNN) models for real-time sound classification using features extracted from environmental audio. The models are trained on two different configurations of the dataset—one with 7 sound classes and another with 6 sound classes. Both models utilize MFCC (Mel Frequency Cepstral Coefficients) as the primary input representation.

1. CNN-Based MFCC Classifier (7-Class Model)

This model is designed to classify audio signals into 7 distinct environmental categories. It uses MFCC features, which capture the short-term power spectrum of a sound, mimicking human auditory perception.

Input Feature Details:

- Feature Type: MFCC (Mel Frequency Cepstral Coefficients)
- Number of Coefficients (n_mfcc): 13
- Fixed Time Frames (duration): 260 frames (approx. 9 seconds of audio)
- Input Shape to CNN: (13, 260, 1)

Architecture:

Input Layer: (13, 260, 1)

Conv2D Layer: 32 filters, kernel size (3x3), ReLU, padding='same'

MaxPooling2D: pool size (2x2)

Conv2D Layer: 64 filters, kernel size (3x3), ReLU, padding='same'

MaxPooling2D: pool size (2x2)

Flatten Layer

Dense Layer: 128 units, ReLU activation

Dense Layer: 64 units, ReLU activation

Output Layer: Dense with 7 units, Softmax activation

Training Parameters:

- Loss Function: Categorical Crossentropy

- Optimizer: Adam
- Epochs: 20
- Batch Size: 16
- Split Ratio: 80% training, 20% validation

2. <u>CNN-Based MFCC Classifier (6-Class Model)</u>

This version of the model is trained on a slightly smaller dataset with 6 sound classes. A placeholder category like 'no class may be included to handle uncertain or unknown audio types.

Input Feature Details:

- Feature Type: MFCC (Mel Frequency Cepstral Coefficients)
- Number of Coefficients: 13
- Target Frame Width: 260 frames
- Input Shape: (13, 260, 1)

CNN Architecture:

- Identical to the 7-class model in terms of convolutional layers and dense blocks.
- The output layer has 6 neurons (instead of 7) to match the number of classes.

Training Parameters:

- Loss Function: Categorical Crossentropy
- Optimizer: Adam
- Epochs: 20
- Batch Size: 16

6. Results:

Performance for different feature types:

Feature Type	Train Accuracy (%)	Validation Accuracy (%)
MFCC	99	96
Mel Spectrogram	98	87
Amplitude (Raw)	87	68

Interpretation:

- MFCC features give the best generalization with high accuracy on both training and validation.
- Mel Spectrogram performs well during training but shows a drop in validation, indicating slight overfitting.
- Raw Amplitude has the lowest performance, showing that basic waveform features are not sufficient alone for robust classification.

7. <u>Real-time Analysis of the Prototype:</u>

In this section, we will describe the process of real-time testing for the sound classification system deployed on the Raspberry Pi Zero 2 W. The pipeline consists of several stages, each contributing to the overall system's functionality and performance. The main objective of this system is to record real-time audio, classify it using a TensorFlow Lite model, and display the classification result with a minimal delay.

1. **Record Live 5-Second Audio Clip:**

   The system begins by capturing real-time audio through the Wave share USB audio device connected to the Raspberry Pi Zero 2 W. A 5-second audio clip is recorded using Python’s sound libraries, like pyaudio or sound device. The clip is captured with a sample rate of 16 kHz to ensure clarity while maintaining manageable data size.

2. **Convert to MFCC or Spectrogram:** Once the audio clip is recorded, it is pre-processed to convert it into a feature format suitable for classification. The two commonly used techniques in this project are **Mel-frequency cepstral coefficients (MFCC)** and **Spectrogram**.

   - **MFCC** is a compact representation of the short-term power spectrum of the sound and is widely used in audio classification tasks.
   - **Spectrograms** provide a time-frequency representation of the signal, showing how the frequencies evolve over time.
   - In this project, the conversion is done using libraries like librosa and TensorFlow utilities.

3. **Run Inference Using TensorFlow Lite Model:**

    - The processed audio features (MFCCs or spectrogram) are then passed to the pre-trained **TensorFlow Lite (TFLite)** model for inference.
    - The TensorFlow Lite model is optimized for edge devices, ensuring fast inference times suitable for real-time applications.
    - The model performs classification and outputs the predicted label along with the associated probability scores.

4. **Display Classification Label:**

    - After inference, the system displays the classification result on the web interface.
    - The classification label, which corresponds to the detected sound class (e.g., "Dog Bark," "Baby Crying," etc.), is displayed along with the associated confidence score.
    - This result is shown in real-time with minimal latency, providing immediate feedback to the user.

Response Time:

The total latency of the system is approximately **1.2–1.5 seconds** from the moment the audio is recorded until the result is displayed. This includes:

- Audio recording time (~0.2 seconds)
- Preprocessing time (MFCC or spectrogram conversion) (~0.2 seconds)
- Inference time (TensorFlow Lite model) (~0.5–0.7 seconds)
- Display time (frontend rendering) (~0.2–0.3 seconds)

This latency is acceptable for real-time sound classification applications, where the user expects rapid feedback.

After testing the system on various real-time inputs, the following results were observed:

Metric	**Value**
Accuracy	91.2%
Latency	~1.3 sec
TFLite Model Size	~2.6 MB
Top-1 Error	8.8%
False Positives	Basin Tap mistaken as Flush, Dog Bark as Baby Crying

Analysis:

- **Accuracy (~91.2%):** The system achieved a high accuracy rate, indicating that it can reliably classify sounds in real-time.
- **Latency (~1.3 seconds):** The system's total response time is low enough for real-time applications, ensuring minimal delay between audio input and output.
- **TFLite Model Size (~2.6 MB):** The size of the TensorFlow Lite model is small, making it suitable for edge devices like the Raspberry Pi Zero 2 W.
- **Top-1 Error (~8.8%):** The model correctly classifies sounds most of the time, with a small margin for misclassification.

- **False Positives:** Some sounds were mistakenly classified as other sounds. For example, a **Basin Tap** was misclassified as a **Flush**, and **Dog Barking** was misclassified as **Baby Crying**. These errors can be reduced by improving the model with additional training data or tuning.

This system is an excellent prototype for real-time sound classification using a small edge device.