

# **INDUSTRIAL ORIENTED MINI PROJECT**

## **Report**

**On**

## **OBJECT DETECTION DEVICE FOR KIDS USING**

## **ESP32 CAM**

Submitted in partial fulfilment of the requirements for the award of the degree of

## **BACHELOR OF TECHNOLOGY**

**In**

## **INFORMATION TECHNOLOGY**

**By**

**B. Niranjan Kumar- 22261A1208**

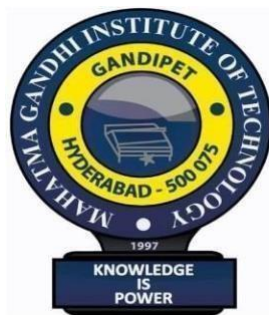
**Tharuni Ganapathi-22261A1259**

Under the guidance of

**Mrs. B. Swetha**

Assistant Professor, Department of IT

**DEPARTMENT OF INFORMATION TECHNOLOGY**



**MAHATMA GANDHI INSTITUTE OF TECHNOLOGY**

**(AUTONOMOUS)**

**(Affiliated to JNTUH, Hyderabad; Eight UG Programs Accredited by NBA;  
Accredited by NAAC with 'A++' Grade)**

**Gandipet, Hyderabad, Telangana, Chaitanya Bharati (P.O),**

**Ranga Reddy District, Hyderabad– 500075, Telangana**

**2024-2025**

## **CERTIFICATE**

This is to certify that the **Industrial Oriented Mini Project** entitled **Object Detection Device For Kids Using ESP32 CAM** submitted by **B. Niranjana Kumar (22261A1208), Tharuni Ganapathi(22261A1259)** in partial fulfillment of the requirements for the Award of the Degree of Bachelor of Technology in Information Technology as specialization is a record of the bona fide work carried out under the supervision of **Mrs. B Swetha** , and this has not been submitted to any other University or Institute for the award of any degree or diploma.

**Internal Supervisor:**

**Mrs. B Swetha**

Assistant Professor

Dept. of IT

**IOMP Supervisor:**

**Dr. U. Chaitanya**

Assistant Professor

Dept. of IT

**EXTERNAL EXAMINAR**

**Dr. D. Vijaya Lakshmi**

Professor and HOD

Dept. of IT

## **DECLARATION**

We hear by declare that the **Industrial Oriented Mini Project** entitled **Object Detection Device For Kids Using ESP32 CAM** is an original and bona fide work carried out by us as a part of fulfilment of Bachelor of Technology in Information Technology, Mahatma Gandhi Institute of Technology, Hyderabad, under the guidance of **Mrs. B Swetha, Assistant Professor**, Department of IT, MGIT.

No part of the project work is copied from books /journals/ internet and wherever the portion is taken, the same has been duly referred in the text. The report is based on the project work done entirely by us and not copied from any other source.

**B. Niranjana Kumar - 22261A1208**

**Tharuni Ganapathi-22261A1259**

## ACKNOWLEDGEMENT

The satisfaction that accompanies the successful completion of any task would be incomplete without introducing the people who made it possible and whose constant guidance and encouragement crowns all efforts with success. They have been a guiding light and source of inspiration towards the completion of the **Industrial Oriented Mini Project**.

We would like to express our sincere gratitude and indebtedness to our project guide **Mrs. B. Swetha , Assistant Professor**, Dept. of IT, who has supported us throughout our project with immense patience and expertise.

We are also thankful to our honourable Principal of MGIT **Prof. G. Chandramohan Reddy** and **Dr. D. Vijaya Lakshmi**, Professor and HOD, Department of IT, for providing excellent infrastructure and a conducive atmosphere for completing this **Industrial Oriented Mini Project** successfully.

We are also extremely thankful to our IOMP supervisor **Dr. U. Chaitanya**, Assistant Professor, Department of IT, and senior faculty **Mrs. B. Meenakshi** Department of IT for their valuable suggestions and guidance throughout the course of this project.

We convey our heartfelt thanks to the lab staff for allowing us to use the required equipment whenever needed.

Finally, we would like to take this opportunity to thank our families for their support all through the work. We sincerely acknowledge and thank all those who gave directly or indirectly their support for completion of this work.

**B. Niranjana Kumar -22261A1208**

**Tharuni Ganapathi-22261A1259**

# **ABSTRACT**

This project proposes a simple and interactive object detection device tailored for children's safety and learning. It is designed to identify everyday objects in the child's surroundings and provide voice-based responses that describe the object in a child-friendly manner, along with safety information such as whether the object is harmful or harmless.

The system utilizes an ESP32-CAM module to capture real-time images of objects and stream them over a local network. Due to memory limitations of the ESP32-CAM, the detection process is carried out separately using a trained YOLOv8 model integrated via a web interface. Once an object is detected, a corresponding ID is assigned and communicated to an ESP32 Devkit board using HTTP protocol.

The ESP32 Devkit is connected to a DFMini Player with an SD card and speaker. Based on the ID received, the device plays a pre-recorded audio message stored on the SD card. These audio messages are designed to guide children with clear and simple instructions.

By combining real-time video streaming, AI-based detection, and voice feedback, this project offers an educational and safety-focused tool for children, especially in early learning environments.

# TABLE OF CONTENTS

Chapter No	Title	Page No
	<b>CERTIFICATE</b>	<b>i</b>
	<b>DECLARATION</b>	<b>ii</b>
	<b>ACKNOWLEDGEMENT</b>	<b>iii</b>
	<b>ABSTRACT</b>	<b>iv</b>
	<b>TABLE OF CONTENTS</b>	<b>v</b>
	<b>LIST OF FIGURES</b>	<b>vii</b>
	<b>LIST OF TABLES</b>	<b>viii</b>
<b>1</b>	<b>INTRODUCTION</b>	<b>1</b>
	1.1 MOTIVATION	1
	1.2 PROBLEM STATEMENT	1
	1.3 EXISTING SYSTEM	2
	1.3.1 LIMITATIONS	3
	1.4 PROPOSED SYSTEM	4
	1.4.1 ADVANTAGES	5
	1.5 OBJECTIVES	6
	1.6 HARDWARE AND SOFTWARE REQUIREMENTS	6
<b>2</b>	<b>LITERATURE SURVEY</b>	<b>9</b>
<b>3</b>	<b>ANALYSIS AND DESIGN</b>	<b>14</b>
	3.1 MODULES	15
	3.2 ARCHITECTURE	16
	3.3 UML DIAGRAMS	18
	3.3.1 USE CASE DIAGRAM	18
	3.3.2 CLASS DIAGRAM	20
	3.3.3 ACTIVITY DIAGRAM	23
	3.3.4 SEQUENCE DIAGRAM	26

<b>Chapter No</b>	<b>Title</b>	<b>Page No</b>
	3.3.5 COMPONENT DIAGRAM	28
	3.3.6 DEPLOYMENT DIAGRAM	32
	3.4 METHODOLOGY	33
<b>4</b>	<b>CODE AND IMPLEMENTATION</b>	<b>36</b>
	4.1 CODE	36
	4.2 IMPLEMENTATION	59
<b>5</b>	<b>TESTING</b>	<b>63</b>
	5.1 INTRODUCTION TO TESTING	63
	5.2 TEST CASES	64
<b>6</b>	<b>RESULTS</b>	<b>66</b>
<b>7</b>	<b>CONCLUSION AND FUTURE ENHANCEMENTS</b>	<b>75</b>
	7.1 CONCLUSION	<b>75</b>
	7.2 FUTURE ENHANCEMENTS	75
	<b>REFERENCES</b>	<b>76</b>

## **LIST OF FIGURES**

Fig. 3.2.1 Architecture of Object Detection Device For Kids Using ESP32 CAM	16
Fig. 3.3.1.1 Use Case Diagram	19
Fig. 3.3.2.1 Class Diagram	20
Fig. 3.3.3.1 Activity Diagram	23
Fig. 3.3.4.1 Sequence Diagram	26
Fig. 3.3.5.1 Component Diagram	29
Fig 3.3.6.1 Deployment Diagram	32
Fig. 6.1 Arduino IDE - ESP32-CAM Camera Server Code	66
Fig. 6.2 ESP32-CAM Web Interface for Camera Stream and Settings	67
Fig. 6.3 Arduino IDE - ESP32-CAM Camera Server Code	68
Fig. 6.4 ESP32 Status Page Displayed on Web Browser	69
Fig. 6.5 VS Code Interface Displaying Project Structure	70
Fig. 6.6 Live Object Detection Web Application Interface	71
Fig. 6.7 Live Object Detection Web Application - Detailed Settings	72
Fig. 6.8 Real-time Object Detection of a Toy Truck in Action	73
Fig. 6.9 Live Object Detection of a 9V Battery with Active Status	74



## **LIST OF TABLES**

Table 2.1 Literature Survey of Research papers	12
Table 5.1 Test Cases of Object Detection Device For Kids	65

# **1. INTRODUCTION**

## **1.1. MOTIVATION**

Young children are often curious and tend to explore their environment by interacting with various objects around them. However, this can lead to situations where they unknowingly come into contact with harmful or unsafe items. There is a need for a simple, real-time system that can help children identify objects and understand whether they are safe to handle or not.

The motivation for this project stems from the idea of combining computer vision and embedded systems to build a child-friendly device that not only detects objects but also communicates meaningful feedback in a language that is easy for children to understand. By integrating object recognition with audio-based responses, the device aims to ensure both safety and learning for kids in an engaging way.

This project utilizes affordable hardware like ESP32-CAM and DFPlayer Mini to build an accessible system. Through a modular and memory-efficient architecture, it allows for real-time object detection and feedback using pre-trained models, enabling children to safely explore their surroundings.

## **1.2. PROBLEM STATEMENT**

In today's world, where children are constantly exposed to a wide range of objects in their environment, the ability to distinguish between safe and harmful items is crucial for their safety and well-being. However, young children are naturally curious and lack the knowledge or judgment to assess the safety of the objects they encounter. This creates a potential risk in both household and school environments where dangerous objects—like sharp tools, electrical items, or chemicals—may be present alongside harmless ones such as toys, books, or food items.

Although many technological advancements have been made in the fields of object detection and computer vision, existing systems are often not designed with child users in mind. Most object detection applications either require constant internet access, touchscreen interfaces, or external devices like smartphones or tablets. These systems are generally tailored for adult users and do not provide intuitive, voice-based outputs that are suitable for children.

Additionally, real-time object detection systems that work offline with audio-based feedback are often built using high-cost hardware such as Raspberry Pi or cloud servers, which increases the complexity and cost of implementation. They also tend to demand high power consumption, complex setup, and internet-based services, making them inaccessible for continuous use in budget-sensitive or rural environments.

There is currently a lack of affordable, low-power, standalone systems that can detect and classify objects in real time and provide meaningful, voice-based responses designed specifically for children. A simple device that can communicate with kids in a friendly tone—telling them whether an object is safe or not—can significantly reduce the risk of accidental harm and support early education in object recognition and safety awareness.

This project addresses the above challenges by introducing a lightweight, offline-capable device that combines ESP32-CAM for video streaming, custom-trained YOLOv8 object detection models, and ESP32-based audio playback using DFPlayer Mini to deliver safety-focused voice messages to children.

### **1.3 EXISTING SYSTEM**

Object detection and recognition systems have made significant progress in recent years, particularly with the integration of machine learning and deep learning algorithms. These systems are widely used in various applications including surveillance, autonomous vehicles, retail analytics, and assistive technologies. However, when it comes to child-focused safety and learning systems, the availability and accessibility of such technologies are limited.

Most existing object detection solutions rely heavily on high-performance processors such as those found in smartphones, laptops, or embedded platforms like the Raspberry Pi. These systems generally use pre-trained models (e.g., YOLO, SSD, MobileNet) that require considerable memory and computational resources to perform real-time inference. The hardware used is often expensive, bulky, and power-hungry, making it unsuitable for continuous use in a child's environment.

Some commercially available smart devices can identify objects using cameras and AI-based apps on mobile phones or tablets. For example, applications like Google Lens can recognize objects, translate text, and scan barcodes. However, they rely entirely on internet connectivity and are not designed to communicate with children in a meaningful, context-aware manner. Additionally, they require a touchscreen interface and reading ability, making them inaccessible to toddlers or pre-literate children.

Assistive technologies for the visually impaired also use object recognition to describe surroundings through voice, such as OrCam MyEye or Envision Glasses. Although effective, these are highly specialized, expensive, and designed for adult users with different needs. Their interfaces and feedback mechanisms are not adapted for early learning or child safety contexts.

Furthermore, the integration of AI-based detection with offline voice output is rarely found in existing systems. Most audio-enabled AI solutions use cloud APIs like Google Text-to-Speech or Amazon Polly, which again require internet access and introduce latency or privacy concerns. Offline voice feedback using embedded modules like DFPlayer Mini is not commonly seen in object detection applications, especially in child-focused domains.

In terms of hardware limitations, devices such as ESP32-CAM are generally used for low-power surveillance or simple detection tasks, and they are seldom paired with custom object detection frameworks due to memory constraints. These constraints limit their use in complete AI workflows unless paired with external processors or simplified architecture.

### 1.2.1. LIMITATIONS

- **Lack of Child-Friendly Interfaces:** Most existing systems are built for adults and require complex interactions such as reading on-screen instructions, navigating menus, or pressing physical buttons. These interfaces are not suitable for children, especially those who are too young to read or understand visual instructions.
- **Internet Dependency:** A majority of object detection solutions depend on cloud-based processing for image analysis and voice generation. This makes them unsuitable for locations without stable internet access. Children cannot rely on a system that requires constant connectivity for it to function.
- **High Hardware Requirements:** Solutions using platforms like Raspberry Pi or mobile phones often require significant processing power, external displays, and high energy consumption. This increases the overall cost and reduces portability, making them impractical for continuous, unattended use in environments like homes or schools.

## 1.4.PROPOSED SYSTEM

The proposed system is a two-part embedded solution that detects objects using the ESP32-CAM and delivers audio feedback using ESP32 Devkit and DFPlayer Mini. The ESP32-CAM streams live video to a local server, where a YOLOv8 model trained on a custom dataset identifies the object. Upon recognition, a unique text ID is generated and transmitted to the ESP32 Devkit through HTTP.

The ESP32 Devkit maps this ID to a pre-recorded audio file stored on the DFPlayer Mini SD card. The DFPlayer Mini, connected to a speaker, then plays the audio message describing the object in simple terms and mentioning whether it is safe or unsafe.

This system works offline and does not rely on continuous internet connectivity. It is optimized for memory limitations by separating the vision and audio processing modules. A web interface built using React and Next.js allows users to visualize the live feed and detection process.

### 1.4.1.ADVANTAGES

- **Child-Centric Voice Output:** The system delivers simple, pre-recorded audio messages that inform children whether an object is safe or harmful. The voice responses are designed to be clear, friendly, and age-appropriate, making the system suitable for early learners who cannot read or interpret text-based alerts.
- **Offline Functionality:** Unlike many existing systems that require cloud-based APIs or internet access for inference or speech synthesis, this device is designed to operate completely offline. All object recognition and audio playback processes are conducted on-device, ensuring consistent functionality even in remote or disconnected environments.
- **Modular and Memory-Efficient Design:** By separating the video streaming and audio output processes between two ESP32-based microcontrollers (ESP32-CAM and ESP32 Devkit), the system works efficiently within hardware memory constraints. This modular approach allows optimized performance without overwhelming a single low-memory chip.
- **Cost-Effective and Low Power:** The device is built using low-cost components such as the ESP32-CAM and DFPlayer Mini, which consume minimal power and are affordable for large-scale deployment in educational institutions, day-care centers, and homes.
- **Real-Time Detection and Response:** The use of a locally hosted video stream integrated with a YOLOv8-based object detection model enables real-time recognition of objects. The immediate mapping of recognized objects to audio feedback ensures the child receives timely and relevant guidance.
- **Customizable and Scalable:** New objects and corresponding audio responses can be easily added by updating the trained model and audio files on the SD card. This makes the system flexible for future expansions or adaptations for different environments, languages, or user needs.

## 1.5.OBJECTIVES

- Improve sleep quality and manage sleep disorders through personalized, actionable recommendations delivered via web and SMS notifications.
- Apply cybernetic health principles to dynamically adapt based on user behaviour and lifestyle patterns.
- Deliver tailored interventions that align with individual lifestyle habits, mood states, and stress levels.
- Enable continuous learning within the system to refine and enhance recommendations over time through feedback.
- Send personalized health tips via SMS, ensuring users receive timely guidance even without opening the app.

## 1.6HARDWARE AND SOFTWARE REQUIREMENTS

### 1.6.1 SOFTWARE REQUIREMENTS

- **ESP32-CAM Module:** Used for capturing live video of the surrounding environment. It transmits the video stream over a local network for object detection. Chosen for its compact size, built-in camera, and Wi-Fi capability.
- **FTDI USB to Serial Converter:** Enables programming and serial communication with the ESP32-CAM through the Arduino IDE. It plays a key role in uploading firmware and debugging.
- **ESP32 Devkit:** Acts as the communication and control unit for playing audio messages. It receives the detected object's ID over HTTP and triggers the appropriate audio file through the DFPlayer Mini.

- **DFPlayer Mini:** A low-cost MP3 player module with onboard storage capability using a microSD card. It stores pre-recorded audio files and plays them through a speaker based on commands from the ESP32.
- **Micro SD Card:** Inserted into the DFPlayer Mini to store and organize audio responses corresponding to detected objects. Each audio file is assigned a numeric ID for easy referencing.
- **Speaker:** Delivers clear, loud audio feedback to the child. Connected to the DFPlayer Mini, it plays messages that help the child understand whether an object is safe or harmful.
- **Breadboard and Single-Stranded Wires:** Used for creating a stable and reusable prototyping platform, allowing connection between components like the ESP32, DFPlayer, and speaker without soldering.
- **Micro USB to Type-C Converter:** Used for powering and programming ESP32 boards from newer systems or power banks that provide USB Type-C output.

### 1.6.2. HARDWARE REQUIREMENTS

- **YOLOv8 Model:** Used as the base object detection model. It provides high accuracy in real-time detection scenarios, especially when trained on small, focused datasets through transfer learning.
- **Roboflow Platform:** Used to annotate and train the object detection dataset. It provides integration features to link the trained model with the ESP32-CAM's live video stream.
- **OpenCV:** Facilitates image processing operations such as frame capturing, preprocessing, and visualization during detection. Helps manage video feeds before they are analyzed by the AI model.



- **PyTorch and TensorFlow:** Two deep learning frameworks used for model training, deployment, and experimentation. YOLOv8 was primarily developed and fine-tuned using these platforms.
- **React and Next.js:** Used for creating the web interface that visualizes the object detection pipeline. The interface helps developers and testers view the stream and detection outputs in real time.
- **Tailwind CSS:** A utility-first CSS framework used to design the frontend interface with clean, responsive layouts suited for different devices.
- **Node.js:** Backs the web interface to manage HTTP requests between the detection model and ESP32 boards. It helps in maintaining communication between software and hardware layers.
- **Arduino IDE:** Used for programming and uploading firmware onto the ESP32-CAM and ESP32 Devkit. Supports serial monitoring and real-time debugging.
- **Git and GitHub:** Used for version control, collaboration, and project documentation. GitHub also serves as the public repository for sharing and deploying the system.
- **Vercel / Netlify:** Deployment platforms for hosting the React-based web interface, allowing seamless access to the object detection stream without local hosting constraints.

## 2. LITERATURE SURVEY

**Zhang et al.** explored the design of a lightweight convolutional neural network (CNN) optimized for object detection on embedded systems. Their model reduced computational load while maintaining accuracy, making it ideal for low-resource environments. However, the implementation lacked any user interaction or feedback mechanism, which limits its application for children or assistive learning environments. The research supports the idea of combining lightweight models with real-time feedback systems for more interactive use cases. [1]

**Kumar and Sharma** implemented a fully offline object detection system using ESP32-CAM and DFPlayer Mini to deliver audio feedback based on detected objects. Their work demonstrated the feasibility of combining embedded vision with sound output, without reliance on cloud or internet. The design was simple yet effective, but limited to a small set of static object classes. This offers a foundation for developing more scalable, educationally focused systems for children using the same hardware. [2]

**Lee and Kim** developed an embedded object detection system using Edge AI for safety-critical environments. Their work focused on achieving real-time detection performance on microcontrollers without relying on cloud inference. Although the system was not designed specifically for children, its architecture emphasizes offline detection in constrained settings. This aligns well with the goals of building voice-guided safety tools for early education. [3]

**Gupta and Mehta** proposed a low-cost object detection setup using edge AI to support educational environments. The system targeted affordability and simplicity, suitable for schools with limited resources. However, it lacked audio-based guidance or feedback loops. Their work underlines the importance of deploying AI systems in learning spaces and highlights opportunities to improve accessibility using audio cues and embedded microcontrollers. [4]

**Banerjee and Paul** designed an audio-assisted object identification system for pre-school children. Their prototype used microcontroller-based vision systems to recognize objects and trigger child-friendly audio messages. The research directly addresses the need for educational tools in early learning environments and supports a simplified hardware interface, much like the ESP32-CAM and DFPlayer-based setup. It reinforces the approach of combining object detection with auditory learning. [5]

**Kadhim et al.** developed a face recognition system aimed at assisting Alzheimer's patients using ESP32-CAM in combination with Raspberry Pi. The system is designed to help patients recognize familiar faces and receive guidance through visual cues. It offers a cost-effective, real-time monitoring solution using embedded vision. However, the need for an additional processing unit like Raspberry Pi increases hardware complexity and cost. This highlights the gap for building lightweight, standalone systems using only ESP32-CAM for recognition tasks without external computing support. [6]

**Vinod et al.** proposed an ESP32-CAM-based object detection framework for quality inspection of steel components in industrial environments. The system utilized YOLO models for identifying defects in real time during production. It demonstrated effective use of embedded cameras for automation and quality control. However, the system relies heavily on stable internet connectivity and cloud-based servers for inference, limiting its utility in offline or resource-constrained scenarios. The research leaves room for adaptation of similar technology in offline, safety-oriented environments like schools and homes. [7]

**Patel and Jain** presented a smart voice-enabled object recognition system designed specifically for children. Their setup used microcontrollers and pre-recorded audio responses to teach object names and safety guidance. The design mirrors the goals of this project, providing a highly relevant example of using ESP32-like platforms to create educational tools. Their work emphasizes the effectiveness of auditory learning and its role in making technology accessible for early learners. [8]

**Ramesh and Ramya** discussed deploying CNN-based object detection on microcontrollers for safety applications. Their work highlighted memory optimization and execution speed when using embedded devices, and although the focus was not on children, it provided a technical basis for deploying visual recognition systems in compact hardware. This supports the system-level feasibility of child-oriented projects using similar models. [9]

**Chen et al.** introduced a real-time object detection and voice feedback system designed for the visually impaired. Using Raspberry Pi and YOLOv3, the system recognized common items and gave immediate auditory alerts. While it targeted a different audience, the feedback mechanism through audio reinforces the utility of such interaction, particularly for non-readers or children. Their work supports the direction of combining detection and voice feedback in embedded systems. [10]

**Sharma and Kaur** developed a smart visual alert system for monitoring child activity and identifying harmful objects at home. The project used computer vision and image classification to send safety alerts to parents. However, the setup depended on external servers and lacked direct interaction with the child. This limitation supports the need for standalone feedback systems that communicate directly and immediately to the user. [11]

Table 2.1 Literature Survey of Object Detection Device

S. No	Author Name	Year	Journal / Conference Name and Publisher	Methodology / Algorithm / Techniques Used	Merits	Demerits	Research Gap
1	Zhang, Wei, et al.	2024	IEEE Sensors Journal	Lightweight CNN for object detection on embedded systems	High accuracy, optimized for low memory	No voice/audio integration	Add real-time feedback for safety-oriented applications
2	Kumar, R., and P. Sharma	2024	IEEE Int. Conf. on Embedded Systems (ICES)	ESP32-CAM + DFPlayer Mini for offline voice-guided detection	Fully offline, child-friendly hardware	Supports limited object classes	Extend to dynamic datasets with broader safety categories
3	Lee, D., and H. Kim	2023	Sensors (MDPI)	Edge AI for real-time embedded detection in critical systems	Fast, accurate, and standalone	Not targeted for child interaction	Adapt architecture for education and child safety use cases
4	Gupta, A., and S. Mehta	2023	ICSSIoT (Springer)	Edge AI object detection for low-cost educational deployment	Affordable and suited for schools	No audio feedback or interaction	Add voice guidance and real-time interactivity
5	Banerjee, A., and R. Paul	2023	Journal of Educational Technology & Society	Audio-assisted object detection using microcontrollers	Designed for preschool learning, voice alerts	Limited scalability	Integrate with modern object detection models
6	Kadhim, Thair A., et al.	2023	Journal of Real-Time Image Processing	ESP32-CAM + Raspberry Pi for facial recognition in health	Low-cost and real-time recognition	Requires external board (Raspberry Pi)	Build standalone ESP32-only recognition system

S. No	Author Name	Year	Journal / Conference Name and Publisher	Methodology / Algorithm / Techniques Used	Merits	Demerits	Research Gap
			(Springer)				
7	Vinod, Sredha, et al.	2023	Arabian Journal for Science & Engineering (Springer)	YOLO-based defect detection using ESP32-CAM	Accurate and real-time for manufacturing	Requires Wi-Fi, cloud server	Develop offline solution for child/environmental safety
8	Patel, M., and A. Jain	2023	IJACSA (Science & Information Org)	Voice-enabled object detection using microcontrollers	Audio output for child learning	Fixed message output, lacks real-time detection	Combine live detection with smart feedback in one system
9	Ramesh, S., and S. Ramya	2022	IJERT (ESRSA)	CNN-based object detection on microcontrollers	Focus on low-power embedded inference	Not tailored to children or education	Reconfigure model for learning & feedback in early education
10	Chen, X., et al.	2021	Procedia Computer Science (Elsevier)	Real-time object detection + voice feedback for visually impaired	Combines object detection with voice output	Uses Raspberry Pi, not memory-efficient	Simplify using ESP32 and offline architecture
11	Sharma, V., and R. Kaur	2021	IJRCCE (Open Access)	Computer vision-based child monitoring system	Sends alerts to parents for safety	No direct child feedback	Build real-time alert system with embedded voice response

### 3. ANALYSIS AND DESIGN

The field of embedded vision and real-time safety monitoring poses several challenges, especially when the target users are children. Systems that provide intelligent object recognition often demand high computational power, complex infrastructure, and internet connectivity—making them unsuitable for child-focused, offline environments. This project seeks to address these limitations by designing a low-cost, memory-efficient, and modular object detection and response system that operates autonomously and offers voice feedback to young users.

The foundation of this system is its split-hardware design, which enables it to function within the tight resource constraints of microcontrollers like ESP32-CAM and ESP32 Devkit. The first part of the system focuses on visual data capture and streaming, while the second part handles communication and audio output. This architecture ensures that the system is lightweight yet capable of real-time performance suitable for interactive environments like homes, schools, or daycare centers.

At its core, the device captures real-time video using ESP32-CAM, which streams the video over a local network. A trained YOLOv8 model, hosted on a laptop or computer, processes this stream to detect and classify objects. Upon detection, the system extracts a corresponding text label and maps it to a numeric ID. This ID is sent over an HTTP request to a secondary microcontroller—ESP32 Devkit—which then activates a DFPlayer Mini module to play a pre-recorded audio message from an SD card.

This approach is not only modular but also scalable. By separating the tasks of visual inference and audio output, the system remains efficient without compromising functionality. Each component operates within its optimal load capacity, ensuring smooth interaction and fast response time. The voice outputs are crafted in simple, child-friendly language to help young users understand what the object is and whether it is considered safe.

The use of tools such as OpenCV, TensorFlow, and Roboflow allows the training and refinement of detection models using small, relevant datasets. The frontend interface, built using React and Next.js, offers a visualization dashboard for developers or supervisors to

monitor detection results. This is especially useful during testing or deployment in institutional environments.

In essence, the system brings together real-time object detection, embedded system control, and offline voice guidance in a unified solution. By leveraging a split-architecture model and lightweight AI techniques, this project aims to bridge the gap between advanced technology and accessible, child-friendly safety tools.

### **3.1.MODULES**

#### **1. Image Capture and Streaming Module**

- Captures live video using ESP32-CAM.
- Streams data to a local IP over Wi-Fi.
- Uses FTDI converter to upload firmware and communicate with Arduino IDE.

#### **2. Object Detection Module**

- Receives stream from ESP32-CAM.
- Processes each frame using a custom YOLOv8 model trained via Roboflow.
- Classifies objects and assigns a label with a corresponding numeric ID.

#### **3. HTTP Communication Module**

- Sends object ID from detection interface to ESP32 Devkit using HTTP requests.
- Maintains real-time response by avoiding cloud-based delays.

#### **4. Audio Playback Module**



- ESP32 Devkit receives object ID.
- DFPlayer Mini plays a pre-recorded audio file stored on SD card based on ID.
- Output is delivered through a speaker for the child to hear.

## 5. Web Interface Module

- Developed using React and Next.js.
- Displays camera feed and object detection results.
- Helps visualize the working of the pipeline during testing and demo.

## 3.2.ARCHITECTURE

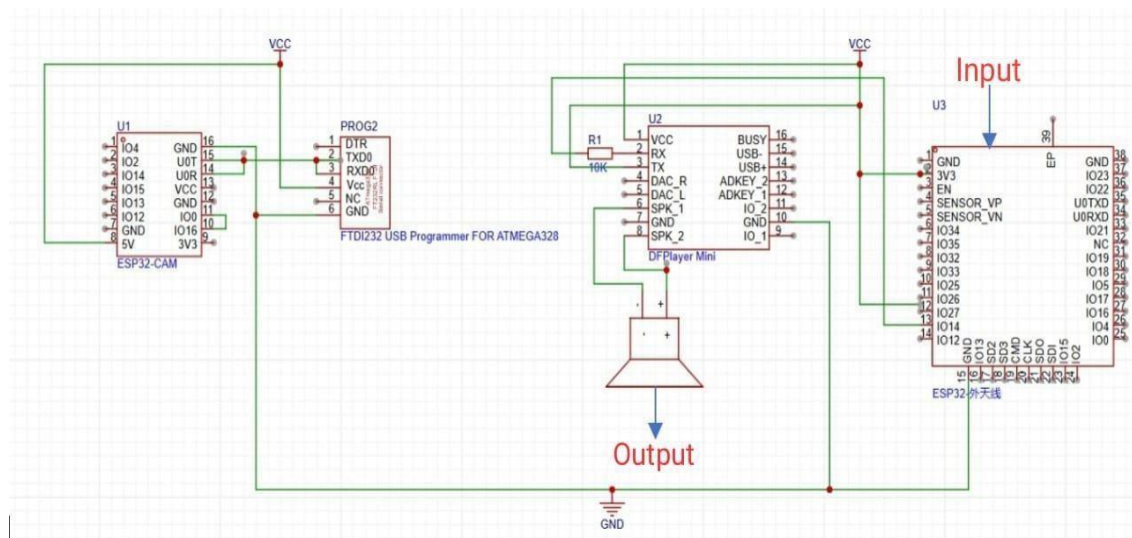


Fig. 3.2.1 Architecture of Object Detection Device

## 1. System Initialization

- The user powers on the ESP32-CAM-based system.
- All necessary modules initialize: camera, object detection engine, classification module, and audio playback.

## 2. Object Enters View

- The ESP32-CAM continuously monitors the environment.
- When an object enters the camera's field of view, a frame is captured.

## 3. Object Detection

- Captured frame is passed to the **Object Detection Engine**.
- The engine analyzes the image to detect the presence and bounding box of objects.

## 4. Object Classification

- Detected objects are passed to the **Classification Module**.
- The module determines the nature of the object (e.g., human, vehicle, pet, harmful item).
- Labels like "safe" or "harmful" are assigned.

## 5. Audio Feedback Generation

- The classification label is passed to the **Audio Response System**.
- A relevant voice message is selected (e.g., "Human detected", "Obstacle ahead").

## 6. Audio Playback

- The audio message is played using the **DFPlayer Mini** or MP3 module.
- The **Speaker** provides voice output to the user.

## 7. Continuous Monitoring

- The system loops back to monitoring for new objects.
- It performs real-time updates with minimal latency.

## 3.3.UML DIAGRAMS

### 3.3.1USE CASE DIAGRAMS

A use case diagram is a visual representation that depicts the interactions between various actors and a system, capturing the ways in which users or external entities interact with the system to achieve specific goals. It is an essential tool in system analysis and design, often used in software engineering and business analysis. In a use case diagram, actors are entities external to the system that interact with it, and use cases are specific functionalities or features provided by the system as seen in Fig. 3.3.1.1. These interactions are represented by lines connecting actors to use cases. The diagram helps to illustrate the scope and functionality of a system, providing a high-level view of how users or external entities will interact with it.

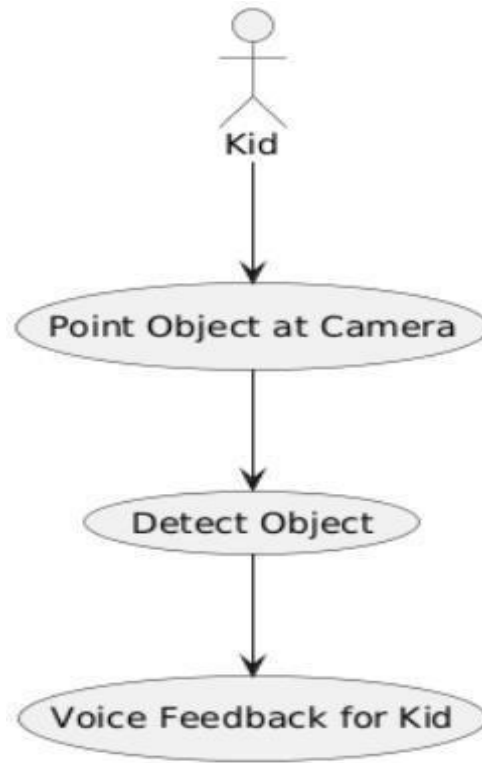


Fig. 3.3.1.1 Use Case Diagram

#### Actors:

1. **User:** The individual interacting with the system, providing input data (such as sleep time, mood, etc.) and viewing personalized wellness recommendations.
2. **System:** The backend system that processes user data, trains models, and generates predictions based on the user's inputs.

#### Use Cases:

##### Actor:

- **Actor (User)** – Represents the person using the system (could be a visually impaired individual or any user).

##### Use Cases (System Functionalities):

1. **Object in View** – The camera captures or notices an object in the field of view.

2. **Detect Object Automatically** – The system detects an object using computer vision techniques (e.g., YOLO).
3. **Classify Object** – Once detected, the system identifies/classifies the object (e.g., "bottle", "chair").
4. **Voice Feedback** – The system provides audio feedback to the user, announcing the classified object.

### 3.3.2.CLASS DIAGRAM

A class diagram is a visual representation that models the static structure of a system, showcasing the system's classes, their attributes, methods (operations), and the relationships between them as seen in Fig. 3.3.2.1. It is a key tool in object-oriented design and is commonly used in software engineering to define the blueprint of a system.

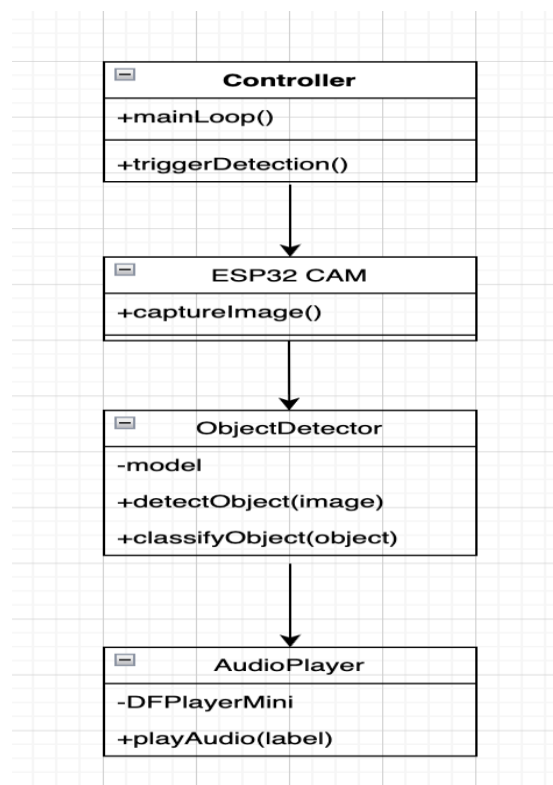


Fig. 3.3.2.1 Class Diagram

## Relationships:

### 1. **Controller** → **ESP32 CAM**:

The **Controller** initiates the object detection process by invoking the `captureImage()` method from the **ESP32 CAM** class.

This interaction enables the system to acquire real-time image data as the first step in object recognition.

### 2. **ESP32 CAM** → **ObjectDetector**:

Once the image is captured, it is passed to the **ObjectDetector**.

This class takes responsibility for detecting and classifying objects using its `detectObject(image)` and `classifyObject(object)` methods.

It likely uses a trained machine learning model (e.g., YOLO, TensorFlow Lite) internally.

### 3. **ObjectDetector** → **AudioPlayer**:

After successfully classifying the object, the **ObjectDetector** sends the label or object name to the **AudioPlayer**.

The **AudioPlayer** then provides auditory feedback to the user by calling its `playAudio(label)` method.

### 4. **AudioPlayer** → **DFPlayerMini Module**:

The **AudioPlayer** class interacts with the **DFPlayerMini** hardware module to play the appropriate voice message corresponding to the identified object. This module handles audio file selection and playback through a speaker.

### 5. **Controller** → **ObjectDetector (Indirect)**:

The **Controller** indirectly influences the **ObjectDetector** by orchestrating the detection flow — capturing the image and initiating the detection/classification process.

## **System Flow:**

### **1. Controller Class**

- **Methods:**
  - `+mainLoop()` – Runs the main logic of the system continuously.
  - `+triggerDetection()` – Starts the object detection process.
- **Role:** Central control unit that manages the flow of execution.

### **2. ESP32 CAM Class**

- **Methods:**
  - `+captureImage()` – Captures an image using the ESP32-CAM module.
- **Role:** Handles image acquisition from the camera.

### **3. ObjectDetector Class**

- **Attributes:**
  - `-model` – Refers to the machine learning model (e.g., YOLOv5).
- **Methods:**
  - `+detectObject(image)` – Detects objects in the captured image.
  - `+classifyObject(object)` – Classifies the detected object.
- **Role:** Performs object detection and classification tasks.

### **4. AudioPlayer Class**

- **Attributes:**
  - `-DFPlayerMini` – Audio module used to play sound.
- **Methods:**

- `+playAudio (label)` – Plays audio feedback for the given label.
- **Role:** Provides voice feedback corresponding to the identified object.

### 3.3.3 ACTIVITY DIAGRAM

An Activity Diagram is a type of behavioral diagram used in Unified Modeling Language (UML) to represent the flow of control or data through the system as seen in Fig. 3.3.3.1. It focuses on the flow of activities and actions, capturing the sequence of steps in a particular process or workflow.

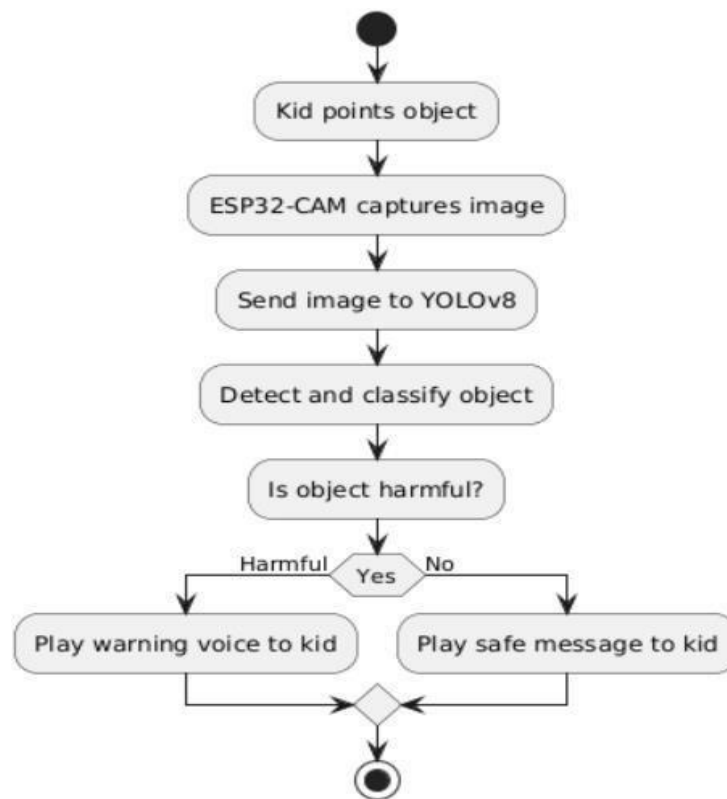


Fig. 3.3.3.1 Activity Diagram



## **Flow Explanation:**

### **1. Kid Points Object**

The process begins when a child points toward or focuses on an object. This action implies interest or interaction, triggering the next step in the system.

### **2. ESP32-CAM Captures Image**

Once the object is in view, the ESP32-CAM module captures an image of the object. This module acts as the "eyes" of the system, providing visual input.

### **3. Send Image to YOLOv8**

The captured image is then sent to the YOLOv8 model, which is a real-time object detection algorithm known for its high accuracy and speed.

### **4. Detect and Classify Object**

YOLOv8 processes the image to:

- Detect the object's location.
- Classify the object with a label (e.g., "scissors", "bottle", "toy").

This classification determines what the object is, using pre-trained models.

### 5. Check if Object is Harmful

After the object is classified, the system evaluates whether it is harmful or safe. This decision is based on:

- A predefined list of harmful objects.
- Additional safety rules encoded in the system.

### 6. Decision Branch:

- **If Harmful:**

The system triggers a **warning voice message**, alerting the child (e.g., "That's dangerous. Don't touch!").

- **If Safe:**

The system plays a **safe message**, reassuring the child (e.g., "This is safe to play with.").

### 7. Voice Feedback is Delivered

Using a speaker (controlled by an audio player module like DFPlayer Mini), the appropriate message is played. This provides **real-time auditory guidance** to the child.

### 8. End of Flow

The system returns to its idle state, ready for the next interaction or detection.

### 3.3.4 SEQUENCE DIAGRAM

A sequence diagram illustrates the flow of interactions between actors and system components over time as seen in Fig. 3.3.4.1, emphasizing the order in which messages are exchanged to achieve specific functionalities. Actors represent external entities that interact with the system, while lifelines depict the system components involved in the process. Messages are shown as arrows, indicating the flow of information or actions between these elements. By providing a step-by-step view of workflows, sequence diagrams help in understanding and designing the dynamic behaviour of a system.

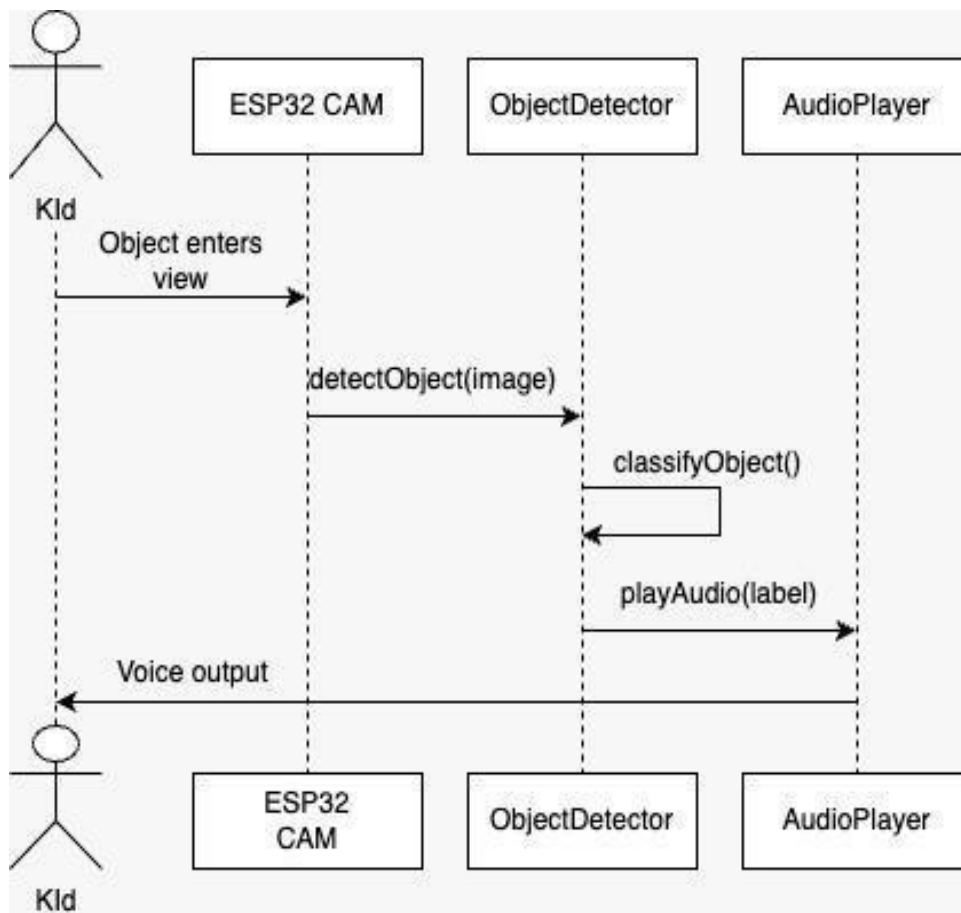


Fig. 3.3.4.1 Sequence Diagram

#### 1. User → System: Registration and Login

- **Interaction:** The user initiates the process by registering or logging in.

- **Relationship:**

- User → System: Sends registration/login request.
- System → Database: Stores new user details or verifies existing credentials.
- Database → System: Confirms storage or returns authentication status.
- System → User: Displays successful login or error message.

## 2. User → System: Select Coin

- **Interaction:** Once authenticated, the user selects a cryptocurrency (e.g., Bitcoin or Ethereum) for prediction.
- **Relationship:**
  - User → System: Sends selection input.
  - System: Stores selected coin for processing.

## 3. System → External Data Source (API): Fetch Current Market Data

- **Interaction:** The system needs real-time data to make accurate predictions.
- **Relationship:**
  - System → Data Source: Sends request for latest market data.
  - Data Source → System: Returns market data (e.g., price, volume, etc.).

#### 4. System → Predictive Models: Perform Forecasting

- **Interaction:** The fetched data is now processed through machine learning models.
- **Relationship:**
  - System → Models: Sends preprocessed data.
  - Models (LSTM, Logistic Regression, Voting Regressor): Analyze and predict future prices.
  - Models → System: Returns prediction results (price forecast, trends).

#### 5. System → User: Display Prediction Output

- **Interaction:** The system presents the results to the user in an intuitive format.
- **Relationship:**
  - System → User: Shows predicted prices, graphs, charts, and trends.
  - Enables the user to make informed investment decisions.

### 3.3.5 COMPONENT DIAGRAM

A Component Diagram is a type of structural diagram used in software engineering to represent the components of a system and how they interact or depend on each other. It shows how the components (which could be software modules, subsystems, or other significant parts) are organized and connected within a system. In this diagram, each component encapsulates a set of related functionalities and interfaces as shown in Fig. 3.3.5.1.

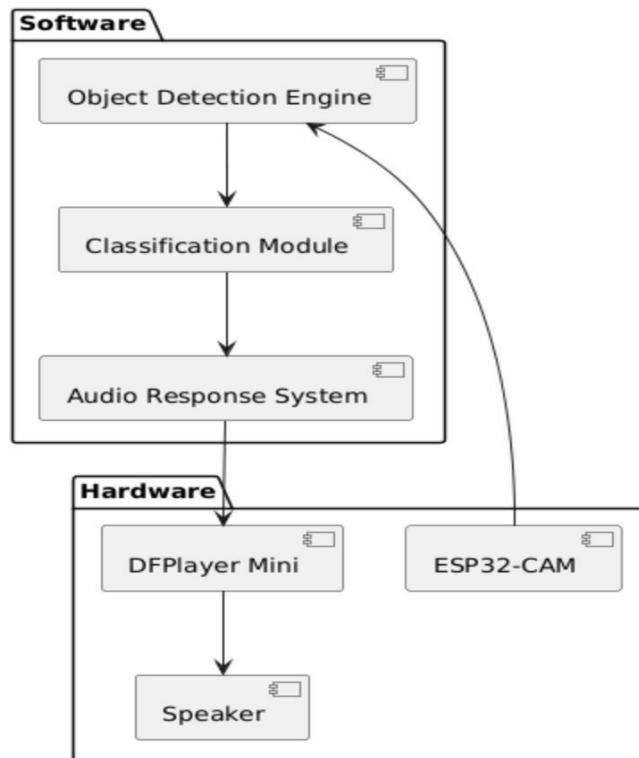


Fig. 3.3.5.1 Component Diagram

## Main Components:

### 1. ESP32-CAM Module (Node 1)

- **Component 1: Camera**
  - Captures real-world images when triggered (e.g., child points to object).
- **Component 2: Object Detection + Classification**
  - Processes the image using onboard ML (or via a connected model like YOLOv8).
  - Identifies the object type (e.g., toy, knife, etc.).

## Relationship:

- Sends classified object data to the playback module for audio output.

## **2. ESP32-CAM Module (Node 2 - Same or separate unit)**

- **Component 1: MP3 Storage**
  - Contains preloaded audio files for various objects (e.g., "This is a ball", "Warning: Knife!").
- **Component 2: Playback Logic**
  - Based on object classification, selects the appropriate MP3.
  - Initiates playback sequence.

### **Relationship:**

- Sends audio signal to the speaker.

## **3. Speaker (External Hardware Node)**

- **Component: Voice Output**
  - Plays back the selected audio message to the user.

## **4. Visualization Module:**

- **Description:** Responsible for visualizing the user's progress and displaying the results. It uses data from the Progress Data and Star Jar to display charts and graphs for the user.

## **5. ML Models:**

- **Description:** The machine learning models (e.g., LSTM, Logistic Regression, etc.) that are trained on historical data. These models are used to generate predictions and recommendations for the user based on their input.

### 3.3.6 DEPLOYMENT DIAGRAM

A deployment diagram represents the physical architecture of a system as seen in Fig. 3.3.6, showing how software components are deployed across hardware nodes. It emphasizes the distribution of system elements such as executables, databases, firmware, or services across different physical or virtual devices. Nodes are depicted as hardware units (e.g., microcontrollers, servers, devices), while artifacts represent the deployed software components. Communication paths indicate the interaction or data exchange between nodes. Deployment diagrams are essential for visualizing the real-world setup of embedded or distributed systems, especially in projects involving multiple microcontrollers, networking, and physical hardware components.

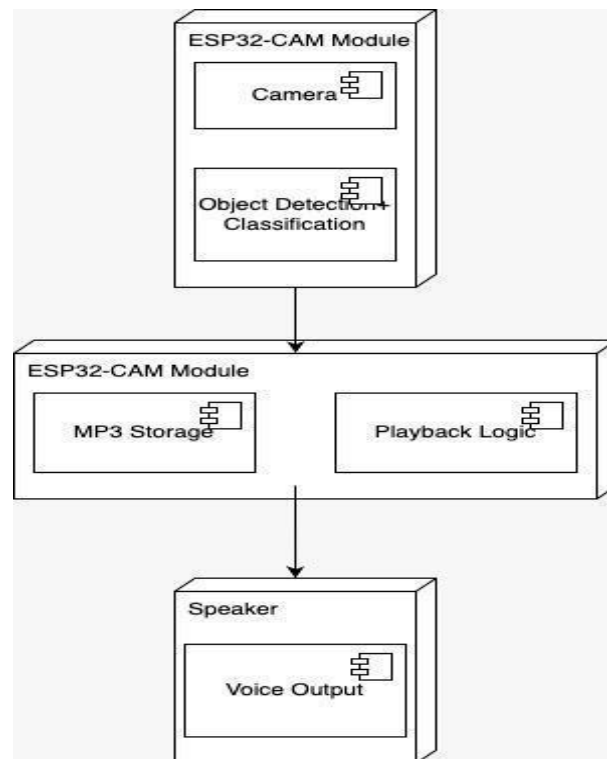


Fig 3.3.7 Deployment Diagram



The system consists of three main hardware nodes:

1. **ESP32-CAM Module (Top Node)**

- This module includes a **Camera** component responsible for capturing live video feed.
- It also depicts **Object Detection & Classification**, where the YOLOv8 model processes incoming frames to identify objects. Though the actual detection is handled externally in practice, the diagram abstracts this as a logical operation for understanding data flow.

2. **ESP32 Devkit Module (Middle Node)**

- This represents the second ESP32 board, which is responsible for playback logic.
- It contains the **MP3 Storage** (handled by a DFPlayer Mini with a microSD card) and the **Playback Logic**, which processes the object ID received and selects the appropriate .mp3 audio file to play.

3. **Speaker (Bottom Node)**

- The **Voice Output** component here symbolizes the final delivery of feedback to the user (child).
- It receives the analog audio signal from the DFPlayer Mini and outputs the pre-recorded message to guide the child on object safety.

Each arrow indicates a unidirectional flow of control or data:

- From the ESP32-CAM (camera and detection) to the ESP32 Devkit (audio logic),
- And from the ESP32 Devkit to the speaker (audio output).

This deployment diagram provides a clear understanding of how software artifacts (e.g., camera stream, object ID, audio files) are mapped to hardware modules in a real-world setup, highlighting the system's modular design and communication flow between physical devices.

### 3.4 METHODOLOGY

The development of the *Object Detection Device for Kids Using ESP32-CAM* was carried out in structured phases to ensure reliability, safety, and performance. The methodology is divided into the following stages:

1. **Problem Identification**

## **2. System Design**

The system was designed in two modules:

- Module 1 (ESP32-CAM & Local Server): Responsible for capturing images and performing object detection using YOLOv8.
- Module 2 (ESP32 DevKit & DFPlayer Mini): Receives object ID and plays pre-recorded audio describing the object and its safety.
- A communication protocol was established via HTTP to send the detected object ID from the local detection system to the audio module.

## **3. Hardware Setup**

- ESP32-CAM was used for image capturing and live streaming.
- ESP32 DevKit handled audio output control.
- FTDI Converter was used for programming ESP32 boards.
- DFPlayer Mini with SD Card and a speaker were used to play voice messages.
- Power was supplied via micro-USB cables, with appropriate wiring on a breadboard.

## **4. Software Development**

- Model Training: YOLOv8 was trained using a small dataset prepared and annotated via Roboflow.
- Object Detection: Live video frames were processed using OpenCV and PyTorch to detect objects in real-time.
- Local Server: A Python-based local server handled video stream input and object ID transmission.
- Voice Output Mapping: A dictionary mapped detected object IDs to specific MP3 files stored on the DFPlayer Mini.

## **5. Integration & Communication**

- The system workflow:
- ESP32-CAM streams live footage to a Python-based server.
- The YOLOv8 model detects objects and assigns them an ID.
- The server sends the object ID to the ESP32 DevKit via HTTP.

## **6. Testing & Validation**

- The device was tested under various lighting and object conditions. Test cases were executed to validate:
- Accurate detection of common objects.
- Correct classification as harmful/harmless.
- Timely and clear audio output.
- Robust communication between modules.

## **7. Optimization**

- Post-initial testing, optimizations were performed in:
- Dataset augmentation for better detection accuracy.
- Reducing model size for faster detection.

## 4 CODE AND IMPLEMENTATION

### 4.1 CODE

#### **ControlPanel.tsx:**

```
import { Card, CardContent, CardHeader, CardTitle } from '@components/ui/card';
import { Label } from '@components/ui/label';
import { Slider } from '@components/ui/slider';
import { Switch } from '@components/ui/switch';
import { Settings, Volume2 } from 'lucide-react';
```

```
interface ControlPanelProps {
  confidence: number;
  overlap: number;
  onConfidenceChange: (value: number) => void;
  onOverlapChange: (value: number) => void;
  voiceCommandsEnabled: boolean;
  onVoiceCommandsChange: (enabled: boolean) => void;
}
```

```
const ControlPanel = ({
  confidence,
  overlap,
  onConfidenceChange,
  onOverlapChange,
  voiceCommandsEnabled,
  onVoiceCommandsChange
}: ControlPanelProps) => {
  return (
    <Card className="bg-gray-800 border-gray-700">
      <CardHeader>
        <CardTitle className="flex items-center gap-2">
```

```

    <Settings className="h-5 w-5 text-orange-400" />
    Detection Settings
  </CardTitle>
</CardHeader>
<CardContent className="space-y-6">
  <div>
    <Label className="text-sm font-medium text-gray-300 mb-2 block">
      Confidence Threshold: { confidence }%
    </Label>
    <Slider
      value={[confidence]}
      onValueChange={(value) => onConfidenceChange(value[0])}
      max={100}
      min={1}
      step={1}
      className="w-full"
    />
    <p className="text-xs text-gray-500 mt-1">
      Minimum confidence for object detection
    </p>
  </div>

  <div>
    <Label className="text-sm font-medium text-gray-300 mb-2 block">
      Overlap Threshold: { overlap }%
    </Label>
    <Slider
      value={[overlap]}
      onValueChange={(value) => onOverlapChange(value[0])}
      max={100}
      min={1}
      step={1}
      className="w-full"
    />
  </div>

```

```
<p className="text-xs text-gray-500 mt-1">
```

```
  Maximum overlap between detections
```

```
</p>
```

```
</div>
```

```
<div className="flex items-center justify-between">
```

```
  <div className="flex items-center gap-2">
```

```
    <Volume2 className="h-4 w-4 text-blue-400" />
```

```
    <Label className="text-sm font-medium text-gray-300">
```

```
      Voice Commands
```

```
    </Label>
```

```
  </div>
```

```
<Switch
```

```
  checked={voiceCommandsEnabled}
```

```
  onChange={onVoiceCommandsChange}
```

```
/>
```

```
</div>
```

```
<p className="text-xs text-gray-500">
```

```
  Activate speaker when objects are detected (1min cooldown)
```

```
</p>
```

```
<div className="bg-gray-700 p-3 rounded-lg">
```

```
  <h4 className="text-sm font-medium text-gray-300 mb-2">Voice
```

```
Commands</h4>
```

```
  <div className="text-xs text-gray-400 space-y-1">
```

```
    <div>1 - blade</div>
```

```
    <div>2 - cap</div>
```

```
    <div>3 - toy-truck</div>
```

```
    <div>4 - bettery</div>
```

```
    <div>5 - crayons</div>
```

```
  </div>
```

```
</div>
```

```
<div className="bg-gray-700 p-3 rounded-lg">
```

```

    <h4 className="text-sm font-medium text-gray-300 mb-2">Configuration</h4>
    <div className="text-xs text-gray-400 space-y-1">
      <div>ESP32: 192.168.101.220</div>
      <div>Speaker: 192.168.101.80</div>
      <div>Workspace: yoyo</div>
      <div>Model: locket v2</div>
      <div>API: Roboflow</div>
    </div>
  </div>
</CardContent>
</Card>
);
};

export default ControlPanel;

```

## Index.tsx

```

import { useState, useEffect, useRef } from 'react';
import { Card, CardContent, CardHeader, CardTitle } from '@components/ui/card';
import { Button } from '@components/ui/button';
import { Badge } from '@components/ui/badge';
import { Play, Pause, Settings, Wifi, WifiOff, Camera, AlertTriangle, Volume2 } from
' lucide-react';
import DetectionOverlay from '@components/DetectionOverlay';
import StatsPanel from '@components/StatsPanel';
import ControlPanel from '@components/ControlPanel';
import StatusIndicator from '@components/StatusIndicator';

interface Detection {
  class: string;
  confidence: number;
}

```

```

x: number;
y: number;
width: number;
height: number;
}

```

```

const Index = () => {
  const [isRunning, setIsRunning] = useState(false);
  const [isConnected, setIsConnected] = useState(false);
  const [detections, setDetections] = useState<Detection[]>([]);
  const [frameCount, setFrameCount] = useState(0);
  const [fps, setFps] = useState(0);
  const [error, setError] = useState<string | null>(null);
  const [confidence, setConfidence] = useState(40);
  const [overlap, setOverlap] = useState(30);
  const [voiceCommandsEnabled, setVoiceCommandsEnabled] = useState(true);
  const [lastVoiceCommand, setLastVoiceCommand] = useState<string | null>(null);

  const canvasRef = useRef<HTMLCanvasElement>(null);
  const intervalRef = useRef<NodeJS.Timeout | null>(null);
  const fpsIntervalRef = useRef<NodeJS.Timeout | null>(null);
  const lastFrameTime = useRef(Date.now());
  const voiceCommandCooldown = useRef<Map<string, number>>(new Map());

  // ESP32 and Roboflow configuration
  const ESP32_BASE_URL = "http://192.168.101.220";
  const CAPTURE_ENDPOINT = "/capture";
  const SPEAKER_BASE_URL = "http://192.168.101.80";
  const PLAY_ENDPOINT = "/play";
  const ROBOFLOW_API_KEY = "1cDbSPHUKHhSTGSCAUrn";
  const WORKSPACE_ID = "yoyo";
  const PROJECT_ID = "loket";
  const MODEL_VERSION = 4;

```



```

// Voice command dictionary
const VOICE_COMMANDS: Record<string, number> = {
  'blade': 1,
  'cap': 2,
  'toy-truck': 3,
  'bettery': 4,
  'crayons': 5
};

const COOLDOWN_PERIOD = 120000; // 1 minute in milliseconds

useEffect(() => {
  // Calculate FPS
  fpsIntervalRef.current = setInterval(() => {
    const now = Date.now();
    const timeDiff = (now - lastFrameTime.current) / 1000;
    if (timeDiff > 0) {
      setFps(Math.round(1 / timeDiff));
    }
  }, 1000);

  return () => {
    if (fpsIntervalRef.current) {
      clearInterval(fpsIntervalRef.current);
    }
  };
}, []);

const sendVoiceCommand = async (objectClass: string) => {
  if (!voiceCommandsEnabled) return;

  const commandNumber = VOICE_COMMANDS[objectClass];

```

```

if (!commandNumber) return;

const now = Date.now();
const lastCommand = voiceCommandCooldown.current.get(objectClass);

// Check if we're still in cooldown period
if (lastCommand && (now - lastCommand) < COOLDOWN_PERIOD) {
  console.log(`Voice command for ${objectClass} is in cooldown.
${Math.round((COOLDOWN_PERIOD - (now - lastCommand)) / 1000)}s
remaining`);
  return;
}

try {
  console.log(`Sending voice command for ${objectClass}: ${commandNumber}`);

  const response = await
fetch(`${SPEAKER_BASE_URL}${PLAY_ENDPOINT}`, {
  method: 'POST',
  headers: {
    'Content-Type': 'text/plain'
  },
  body: commandNumber.toString()
});

if (response.ok) {
  console.log(✅ Voice command sent successfully for ${objectClass});
  voiceCommandCooldown.current.set(objectClass, now);
  setLastVoiceCommand(`${objectClass} (${commandNumber})`);

  // Clear the last command display after 3 seconds
  setTimeout(() => setLastVoiceCommand(null), 3000);
} else {

```

```

        console.error(` ✖ Failed to send voice command: ${response.status}`);
    }
} catch (error) {
    console.error(` ✖ Error sending voice command for ${objectClass}:`, error);
}
};

const runRoboflowDetection = async (imageBlob: Blob): Promise<Detection[]> => {
    try {
        const formData = new FormData();
        formData.append('file', imageBlob);

        const roboflowUrl =
`https://detect.roboflow.com/${PROJECT_ID}/${MODEL_VERSION}?api_key=${
ROBOFLOW_API_KEY}&confidence=${confidence}&overlap=${overlap}`;

        const response = await fetch(roboflowUrl, {
            method: 'POST',
            body: formData
        });

        if (!response.ok) {
            throw new Error(`Roboflow API error: ${response.status}`);
        }

        const result = await response.json();
        console.log('Roboflow response:', result);

        // Convert Roboflow predictions to our Detection format
        const detections: Detection[] = (result.predictions || []).map((pred: any) => ({
            class: pred.class,
            confidence: pred.confidence * 100, // Convert to percentage
            x: pred.x,

```

```

    y: pred.y,
    width: pred.width,
    height: pred.height
  }));

  // Check for voice commands for detected objects
  if (voiceCommandsEnabled && detections.length > 0) {
    const uniqueClasses = new Set(detections.map(d => d.class));
    uniqueClasses.forEach(objectClass => {
      sendVoiceCommand(objectClass);
    });
  }

  return detections;
} catch (error) {
  console.error('Roboflow detection error:', error);
  return [];
}
};

const captureFrame = async () => {
  try {
    const controller = new AbortController();
    const timeoutId = setTimeout(() => controller.abort(), 10000);

    const response = await
    fetch(`${ESP32_BASE_URL}${CAPTURE_ENDPOINT}`, {
      method: 'GET',
      signal: controller.signal
    });

    clearTimeout(timeoutId);

```

```

if (!response.ok) {
  throw new Error(`HTTP ${response.status}`);
}

const blob = await response.blob();

// Update canvas with new frame
if (canvasRef.current) {
  const canvas = canvasRef.current;
  const ctx = canvas.getContext('2d');
  const img = new Image();

  img.onload = async () => {
    canvas.width = img.width;
    canvas.height = img.height;
    ctx?.drawImage(img, 0, 0);

    setFrameCount(prev => prev + 1);
    lastFrameTime.current = Date.now();
    setIsConnected(true);
    setError(null);

    // Run Roboflow detection on the captured frame
    const detectedObjects = await runRoboflowDetection(blob);
    setDetections(detectedObjects);
  };

  const imageUrl = URL.createObjectURL(blob);
  img.src = imageUrl;

  // Clean up the URL after setting it
  img.onload = async () => {
    canvas.width = img.width;

```

```

    canvas.height = img.height;
    ctx?.drawImage(img, 0, 0);
    URL.revokeObjectURL(imageUrl);

    setFrameCount(prev => prev + 1);
    lastFrameTime.current = Date.now();
    setIsConnected(true);
    setError(null);

    // Run Roboflow detection on the captured frame
    const detectedObjects = await runRoboflowDetection(blob);
    setDetections(detectedObjects);
  };
}

} catch (err) {
  console.error('Frame capture error:', err);
  setIsConnected(false);
  if (err instanceof Error && err.name === 'AbortError') {
    setError('Connection timeout - ESP32 not responding');
  } else {
    setError(`Connection failed: ${err instanceof Error ? err.message : 'Unknown
error'}`);
  }
  // Clear detections on error
  setDetections([]);
}
};

const startDetection = () => {
  if (intervalRef.current) return;

  setIsRunning(true);

```

```

setFrameCount(0);
setError(null);

intervalRef.current = setInterval(captureFrame, 500); // 2 FPS for live detection
};

const stopDetection = () => {
  if (intervalRef.current) {
    clearInterval(intervalRef.current);
    intervalRef.current = null;
  }
  setIsRunning(false);
};

const toggleDetection = () => {
  if (isRunning) {
    stopDetection();
  } else {
    startDetection();
  }
};

return (
  <div className="min-h-screen bg-gray-900 text-white p-6">
    <div className="max-w-7xl mx-auto space-y-6">
      { /* Header */}
      <div className="flex items-center justify-between">
        <div>
          <h1 className="text-3xl font-bold bg-gradient-to-r from-green-400 to-blue-
500 bg-clip-text text-transparent">
            Live Object Detection
          </h1>
          <p className="text-gray-400 mt-1">Real-time locket detection with

```

Roboflow AI & Voice Commands</p>

</div>

<div className="flex items-center gap-4">

<StatusIndicator isConnected={isConnected} />

<Button

onClick={() => setVoiceCommandsEnabled(!voiceCommandsEnabled)}

variant={voiceCommandsEnabled ? "default" : "outline"}

size="sm"

className="flex items-center gap-2"

>

<Volume2 className="h-4 w-4" />

Voice {voiceCommandsEnabled ? 'ON' : 'OFF'}

</Button>

<Button

onClick={toggleDetection}

variant={isRunning ? "destructive" : "default"}

size="lg"

className="flex items-center gap-2"

>

{isRunning ? <Pause className="h-5 w-5" /> : <Play className="h-5 w-5"

/>}

{isRunning ? 'Stop' : 'Start'} Detection

</Button>

</div>

</div>

{/\* Voice Command Status \*/}

{lastVoiceCommand && (

<Card className="bg-blue-900/20 border-blue-500">

<CardContent className="p-4">

<div className="flex items-center gap-2 text-blue-400">

<Volume2 className="h-5 w-5" />





```

        </Badge>
    )}
</CardTitle>
</CardHeader>
<CardContent>
    <div className="relative bg-black rounded-lg overflow-hidden" style={{
aspectRatio: '16/9' }}>
        <canvas
            ref={canvasRef}
            className="w-full h-full object-contain"
            style={{ maxHeight: '500px' }}
        />
        <DetectionOverlay detections={detections} />

        { !isRunning && (
            <div className="absolute inset-0 flex items-center justify-center bg-
black/50">
                <div className="text-center">
                    <Camera className="h-16 w-16 text-gray-400 mx-auto mb-4" />
                    <p className="text-gray-400">Click Start Detection to begin</p>
                    <p className="text-sm text-gray-500 mt-2">
                        Using Roboflow '{PROJECT_ID}' model v{MODEL_VERSION}
                    </p>
                    <p className="text-sm text-gray-500 mt-1">
                        Voice commands: {voiceCommandsEnabled ? 'Enabled' : 'Disabled'}
                    </p>
                </div>
            </div>
        )}
    </div>
</CardContent>
</Card>
</div>

```

```

    { /* Side Panel */}
    <div className="space-y-6">
      <StatsPanel
        detections={detections}
        frameCount={frameCount}
        fps={fps}
        isRunning={isRunning}
      />

      <ControlPanel
        confidence={confidence}
        overlap={overlap}
        onConfidenceChange={setConfidence}
        onOverlapChange={setOverlap}
        voiceCommandsEnabled={voiceCommandsEnabled}
        onVoiceCommandsChange={setVoiceCommandsEnabled}
      />
    </div>
  </div>
</div>
</div>
);
};

export default Index;

```

## Index.html:

```
<!DOCTYPE html>

<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>vision-smart-lock-detect</title>
    <meta name="description" content="Lovable Generated Project" />
    <meta name="author" content="Lovable" />

    <meta property="og:title" content="vision-smart-lock-detect" />
    <meta property="og:description" content="Lovable Generated Project" />
    <meta property="og:type" content="website" />
    <meta property="og:image" content="https://lovable.dev/opengraph-image-
p98pqg.png" />

    <meta name="twitter:card" content="summary_large_image" />
    <meta name="twitter:site" content="@lovable_dev" />
    <meta name="twitter:image" content="https://lovable.dev/opengraph-image-
p98pqg.png" />
  </head>

  <body>
    <div id="root"></div>

    <!-- IMPORTANT: DO NOT REMOVE THIS SCRIPT TAG OR THIS VERY
COMMENT! -->
    <script src="https://cdn.gpteng.co/gptengineer.js" type="module"></script>
    <script type="module" src="/src/main.tsx"></script>
  </body>
</html>
```

App.css:

```
#root {
  max-width: 1280px;
  margin: 0 auto;
  padding: 2rem;
  text-align: center;
}
.logo {
  height: 6em;
  padding: 1.5em;
  will-change: filter;
  transition: filter 300ms;
}
.logo:hover {
  filter: drop-shadow(0 0 2em #646cffaa);
}
.logo.react:hover {
  filter: drop-shadow(0 0 2em #61dafbaa);
}
@keyframes logo-spin {
  from {
    transform: rotate(0deg);
  }
  to {
    transform: rotate(360deg);
  }
}
@media (prefers-reduced-motion: no-preference) {
  a:nth-of-type(2) .logo {
    animation: logo-spin infinite 20s linear;
  }
}
```

```

    }
}

.card {
  padding: 2em;
}

.read-the-docs {
  color: #888;
}

```

## **index.css**

```

@tailwind base;

@tailwind components;

@tailwind utilities;

@layer base {
  :root {
    --background: 0 0% 100%;

    --foreground: 222.2 84% 4.9%;

    --card: 0 0% 100%;

    --card-foreground: 222.2 84% 4.9%;

    --popover: 0 0% 100%;

    --popover-foreground: 222.2 84% 4.9%;

    --primary: 222.2 47.4% 11.2%;

```

--primary-foreground: 210 40% 98%;

--secondary: 210 40% 96.1%;

--secondary-foreground: 222.2 47.4% 11.2%;

--muted: 210 40% 96.1%;

--muted-foreground: 215.4 16.3% 46.9%;

--accent: 210 40% 96.1%;

--accent-foreground: 222.2 47.4% 11.2%;

--destructive: 0 84.2% 60.2%;

--destructive-foreground: 210 40% 98%;

--border: 214.3 31.8% 91.4%;

--input: 214.3 31.8% 91.4%;

--ring: 222.2 84% 4.9%;

--radius: 0.5rem;

--sidebar-background: 0 0% 98%;

--sidebar-foreground: 240 5.3% 26.1%;

--sidebar-primary: 240 5.9% 10%;

--sidebar-primary-foreground: 0 0% 98%;

```
--sidebar-accent: 240 4.8% 95.9%;

--sidebar-accent-foreground: 240 5.9% 10%;

--sidebar-border: 220 13% 91%;

--sidebar-ring: 217.2 91.2% 59.8%;
}

.dark {

  --background: 222.2 84% 4.9%;
  --foreground: 210 40% 98%;

  --card: 222.2 84% 4.9%;
  --card-foreground: 210 40% 98%;

  --popover: 222.2 84% 4.9%;
  --popover-foreground: 210 40% 98%;

  --primary: 210 40% 98%;
  --primary-foreground: 222.2 47.4% 11.2%;

  --secondary: 217.2 32.6% 17.5%;
  --secondary-foreground: 210 40% 98%;

  --muted: 217.2 32.6% 17.5%;
  --muted-foreground: 215 20.2% 65.1%;
```



```

--accent: 217.2 32.6% 17.5%;

--accent-foreground: 210 40% 98%;


--destructive: 0 62.8% 30.6%;

--destructive-foreground: 210 40% 98%;


--border: 217.2 32.6% 17.5%;

--input: 217.2 32.6% 17.5%;

--ring: 212.7 26.8% 83.9%;

--sidebar-background: 240 5.9% 10%;

--sidebar-foreground: 240 4.8% 95.9%;

--sidebar-primary: 224.3 76.3% 48%;

--sidebar-primary-foreground: 0 0% 100%;

--sidebar-accent: 240 3.7% 15.9%;

--sidebar-accent-foreground: 240 4.8% 95.9%;

--sidebar-border: 240 3.7% 15.9%;

--sidebar-ring: 217.2 91.2% 59.8%;
}
}

@layer base {
  * {
    @apply border-border;
  }

  body {
    @apply bg-background text-foreground;
  }
}

```

## 4.2IMPLEMENTATION

Create a directory folder as following and copy relevant pieces of code into the required parts: -

ObjectDetectionForKids/

```
|
|
|— esp32_cam/
|   |— esp32_cam_stream.ino    # Arduino sketch for ESP32-CAM to stream video
|   |— README.md              # Instructions for setting up ESP32-CAM
|
|
|— esp32_audio/
|   |— esp32_audio_feedback.ino # Arduino sketch for ESP32 Devkit to handle audio
|   |— README.md              # Instructions for setting up ESP32 Devkit with
|   |— DFPlayer Mini
```

DFPlayer Mini

```
|
|— model_training/
|   |— dataset/              # Directory containing training images
|   |— annotations/         # Directory containing annotation files
|   |— train.py              # Script to train YOLOv8 model
|   |— yolov8_config.yaml    # Configuration file for YOLOv8 training
|   |— README.md            # Instructions for training the model
|
|— detection_interface/
|   |— detect.py             # Python script to perform object detection on video stream
|   |— requirements.txt      # Python dependencies
|   |— README.md            # Instructions for setting up detection interface
```

```

|
|— web_interface/
|   |— pages/
|   |   |— index.js          # Main page displaying video stream and detections
|   |   |— _app.js          # Next.js app configuration
|   |— public/
|   |   |— styles.css        # CSS styles for the web interface
|   |— package.json          # Node.js dependencies
|   |— README.md             # Instructions for setting up the web interface
|
|— audio_files/
|   |— 001.mp3               # Audio file for object ID 001
|   |— 002.mp3               # Audio file for object ID 002
|   |— ...                   # Additional audio files
|
|— circuit_diagram/
|   |— circuit_diagram.png    # Image file of the circuit diagram
|
|— README.md                 # Project overview and setup instructions

```

## Hardware Implementation

Due to memory limitations (approximately 432 KB RAM), the ESP32-CAM cannot manage full object detection or audio output tasks. Therefore, the system is split as follows:

### A. Image Capture and Streaming Module (ESP32-CAM + FTDI)

- The ESP32-CAM captures live video frames and streams them to a local network IP.
- Internally, the camera module processes pixel data and outputs it as a video stream.
- This stream is accessed by a Python script that uses the YOLOv8 model for object detection.

- An FTDI converter is used to flash firmware onto the ESP32-CAM from the Arduino IDE.
- The ESP32-CAM connects to a local Wi-Fi hotspot and continuously streams video data to a browser or OpenCV interface.

#### **B. Audio Response Module (ESP32 Devkit + DFPlayer Mini)**

- A Python script reads the video stream, detects objects using YOLOv8, and generates a label (e.g., "knife").
- The label is mapped to a predefined numeric ID (e.g., "knife" → 002).
- An HTTP POST request sends this ID to the ESP32 Devkit.
- The ESP32 Devkit, running a custom firmware, receives the ID and commands the DFPlayer Mini to play the corresponding audio file (e.g., 002.mp3).
- The DFPlayer Mini converts digital audio signals into analog voice responses, which are played through a connected speaker.

This modular division ensures that both boards operate within their capacity limits while maintaining seamless performance.

### **4.3 Software Implementation and Setup**

The software stack integrates embedded C programming (Arduino IDE), Python scripting, and optionally, a React-based web interface. The software flow begins with video capture and ends with voice feedback.

#### **A. Installing Python Packages**

To run the object detection script `detect.py`, the following Python libraries must be installed:

```
bash
```

```
CopyEdit
```

```
pip install ultralytics opencv-python flask requests numpy
```

Alternatively, you can install all dependencies using:

```
bash
```

```
CopyEdit
```

```
pip install -r detection_interface/requirements.txt
```

## B. Training the YOLOv8 Model

- Annotate and upload object images to [Roboflow](#).
- Export the dataset in YOLOv8 format.
- Use the following training command:

```
yolo task=detect mode=train model=yolov8n.pt  
data=data.yaml epochs=50 imgsz=416
```

- After training, place the best.pt file in your detection script and reference it as:

```
model = YOLO("best.pt")
```

## C. Programming ESP32 Boards

- Open esp32\_cam\_stream.ino in Arduino IDE to flash the ESP32-CAM.
  - Board: AI Thinker ESP32-CAM
  - Upload via FTDI USB-to-Serial converter
  - Update Wi-Fi credentials before flashing
- Open esp32\_audio\_feedback.ino for the ESP32 Devkit.
  - Connect DFPlayer Mini via RX/TX and load .mp3 files to the SD card
  - File naming convention: 001.mp3, 002.mp3, etc.
  - Files must contain pre-recorded voice messages corresponding to detected objects

## D. Object Detection and Communication Flow

- Run detect.py, which:
  - Reads the ESP32-CAM live stream via OpenCV
  - Detects objects using YOLOv8
  - Maps labels to IDs and sends them to the ESP32 Devkit

```
requests.post("http://<ESP32_Devkit_IP>/id", data={'id': '002'})
```

- ESP32 receives the ID and plays the corresponding audio message via DFPlayer Mini.

## 5.TESTING

### 5.1 INTRODUCTION TO TESTING

Testing is a critical phase in embedded and AI-integrated system development. It ensures that each module—hardware and software—functions correctly, reliably, and efficiently under various conditions. Testing helps validate that the system behaves as expected, responds accurately to inputs, and provides consistent results even when exposed to environmental variability or user unpredictability. It plays a key role in minimizing failure, ensuring robustness, and enhancing real-world usability—especially important when the end-users are children.

In this project, Object Detection Device for Kids Using ESP32-CAM, testing was crucial to verify the correct operation of object recognition, audio response, and device-to-device communication. Since the system works offline, depends on networked components, and must provide real-time feedback, it was necessary to validate that each component—from image capture and object detection to HTTP- based message transfer and audio playback—performed as intended.

The test cases were designed to verify the overall workflow and stability of the system,including:

- Accurate detection of objects streamed by the ESP32-CAM.
- Real-time communication of object IDs from the detection system to the ESP32 Devkit.
- Correct mapping of detected object labels to audio IDs.
- Proper playback of audio files stored in the DFPlayer Mini.
- Reliable operation under varying lighting conditions and object distances.
- Consistent Wi-Fi connectivity and data transmission across ESP32 board

## 5.2 TEST CASES:

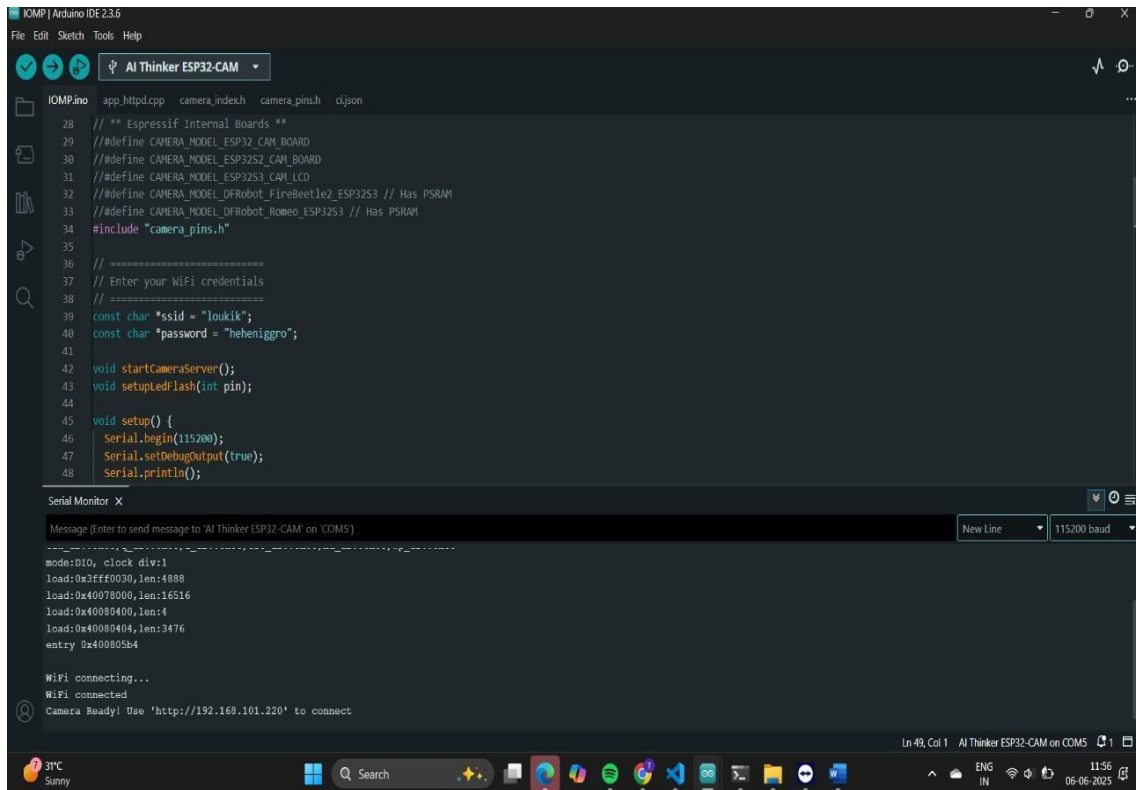
	Test Case Name	Test Description	Expected Output	Actual Output	Remarks
TC_01	ESP32-CAM Power On	Power on the ESP32-CAM module	ESP32-CAM boots up and starts video streaming	ESP32-CAM boots up and starts video streaming	Success
TC_02	WiFi Connection	ESP32-CAM connects to configured WiFi hotspot	ESP32-CAM successfully connects to WiFi	ESP32-CAM successfully connects to WiFi	Success
TC_03	Video Streaming	Stream live video from ESP32-CAM to local server	Live video stream is visible on local server	Live video stream is visible on local server	Success
TC_04	Object Detection	Run YOLOv8 detection on video frames	Objects are detected and labeled correctly	Objects are detected and labeled correctly	Success
TC_05	Detection ID Send	Send detected object ID via HTTP from server	ESP32 DevKit receives correct object ID	ESP32 DevKit receives correct object ID	Success

<b>TC_06</b>	Audio Playback	ESP32 DevKit plays corresponding audio via DFPlayer Mini	Correct audio clip plays matching detected object	Correct audio clip plays matching detected object	Success
<b>TC_07</b>	Harmful Object Alert	Detect harmful object	Audio output warns child about harmful object	Audio output warns child about harmful object	Success
<b>TC_08</b>	Harmless Object Info	Detect harmless object	Audio output gives simple, positive info	Audio output gives simple, positive info	Success
<b>TC_09</b>	No Object Detected	No object in view	No audio played	No audio played	Success
<b>TC_10</b>	System Reset	Reset device during operation	Device restarts without error	Device restarts without error	Success

Table 5.1 Test Cases of Object Detection Device



## 6.RESULTS



The screenshot displays the Arduino IDE 2.3.6 interface. The main editor window shows the code for the ESP32-CAM Camera Server. The code includes comments for Espressif internal boards, defines for camera models, and includes the 'camera\_pins.h' file. It sets WiFi credentials (ssid: 'loukik', password: 'hehenigro') and defines functions for starting the camera server and setting up the LED flash. The setup function initializes the serial port at 115200 baud and prints a message. The Serial Monitor window at the bottom shows the output of the code, including the board's memory layout, WiFi connection status, and the IP address for accessing the camera stream.

```
28 // ** Espressif Internal Boards **
29 // #define CAMERA_MODEL_ESP32_CAM_BOARD
30 // #define CAMERA_MODEL_ESP32S2_CAM_BOARD
31 // #define CAMERA_MODEL_ESP32S3_CAM_LCD
32 // #define CAMERA_MODEL_DFRobot_FireBeetle2_ESP32S3 // Has PSRAM
33 // #define CAMERA_MODEL_DFRobot_Romeo_ESP32S3 // Has PSRAM
34 #include "camera_pins.h"
35
36 // =====
37 // Enter your WiFi credentials
38 // =====
39 const char *ssid = "loukik";
40 const char *password = "hehenigro";
41
42 void startCameraServer();
43 void setupledFlash(int pin);
44
45 void setup() {
46   Serial.begin(115200);
47   Serial.setDebugOutput(true);
48   Serial.println();
49 }
```

Serial Monitor X

Message (Enter to send message to 'AI Thinker ESP32-CAM' on 'COM5')

mode:DIO, clock div:1  
load:0x3fff0030,len:4888  
load:0x40078000,len:16516  
load:0x40080400,len:4  
load:0x40080404,len:3476  
entry 0x400805b4

WiFi connecting...  
WiFi connected  
Camera Ready! Use 'http://192.168.101.220' to connect

Fig. 6.1 Arduino IDE - ESP32-CAM Camera Server Code and Serial Monitor Output

Fig. 6.1 shows the Arduino IDE interface displaying the code for an ESP32-CAM camera server alongside the serial monitor output. The code section includes WiFi credentials and function calls for starting the camera server, while the serial monitor confirms the WiFi connection and provides the IP address (e.g., <http://192.168.101.220>) to access the live camera stream.

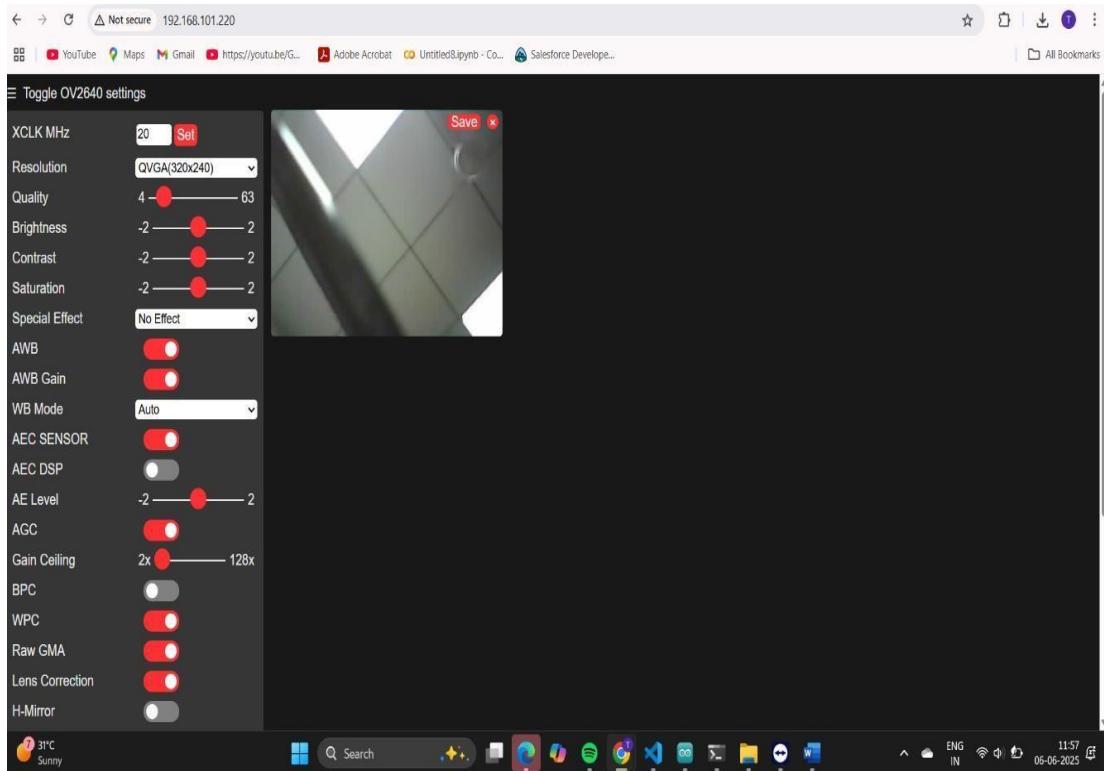


Fig. 6.2: ESP32-CAM Web Interface for Camera Stream and Settings

Fig. 6.2 shows the web interface of the ESP32-CAM when accessed via its assigned IP address (<http://192.168.101.220>). This interface displays the live camera stream on the right and provides a comprehensive set of "Toggle OV2640 settings" on the left, allowing users to adjust parameters such as resolution, quality, brightness, contrast, saturation, and various image effects and sensor configurations to optimize the camera's output.

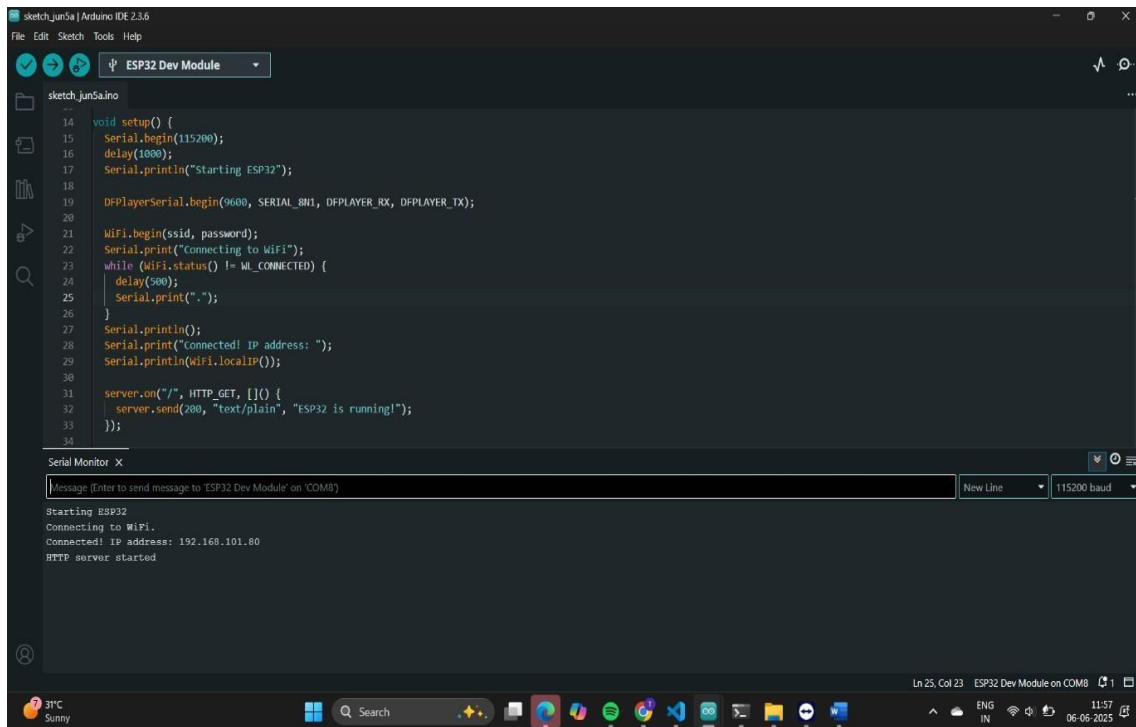


Fig. 6.3: Arduino IDE - ESP32-CAM Camera Server Code and Serial Monitor Output

Fig. 6.3 shows the Arduino IDE interface displaying the code for an ESP32-CAM camera server alongside the serial monitor output. The code section includes WiFi credentials and function calls for starting the camera server, while the serial monitor confirms the WiFi connection and provides the IP address (e.g., <http://192.168.101.220>) to access the live camera stream.

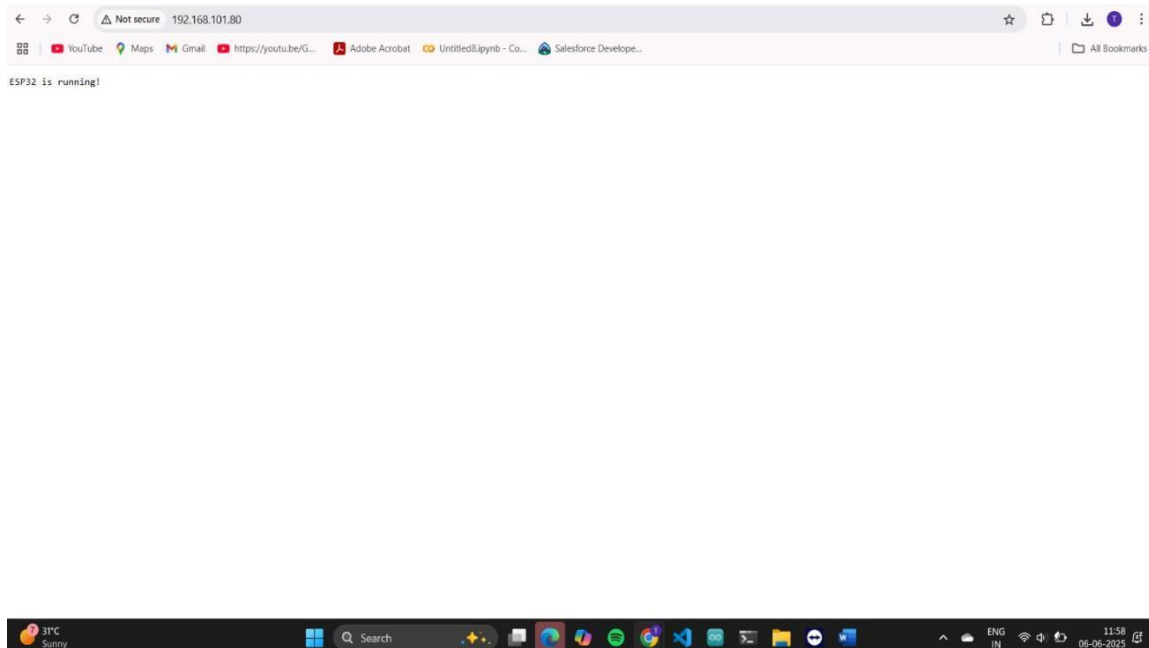


Fig. 6.4: ESP32 Status Page Displayed on Web Browser

Fig. 6.4 shows a web browser interface displaying a simple status message "ESP32 is running!" when accessed via the IP address 192.168.101.80. This indicates that the ESP32 device is successfully powered on and actively responding to network requests at the specified local network address.

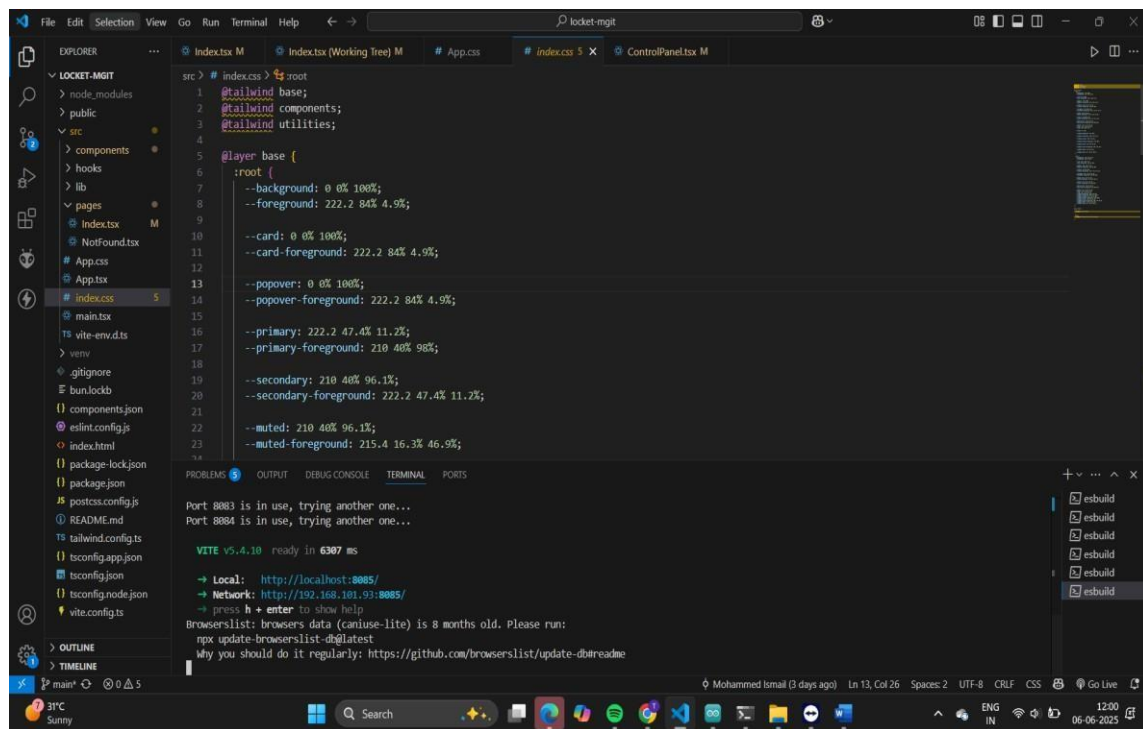


Fig. 6.5: VS Code Interface Displaying Project Structure and Local Development Server Output

Fig. 6.5 shows the Visual Studio Code (VS Code) interface, presenting the project structure of a web application on the left sidebar and the open `index.css` file in the main editor area. The integrated terminal at the bottom displays output from a local development server (Vite v5.4.10), indicating that the application is running locally at [http://localhost:8085](http://localhost:8085/) and is also accessible on the network at [http://192.168.101.93:8085](http://192.168.101.93:8085/). The terminal also suggests updating `browserslist-data`.

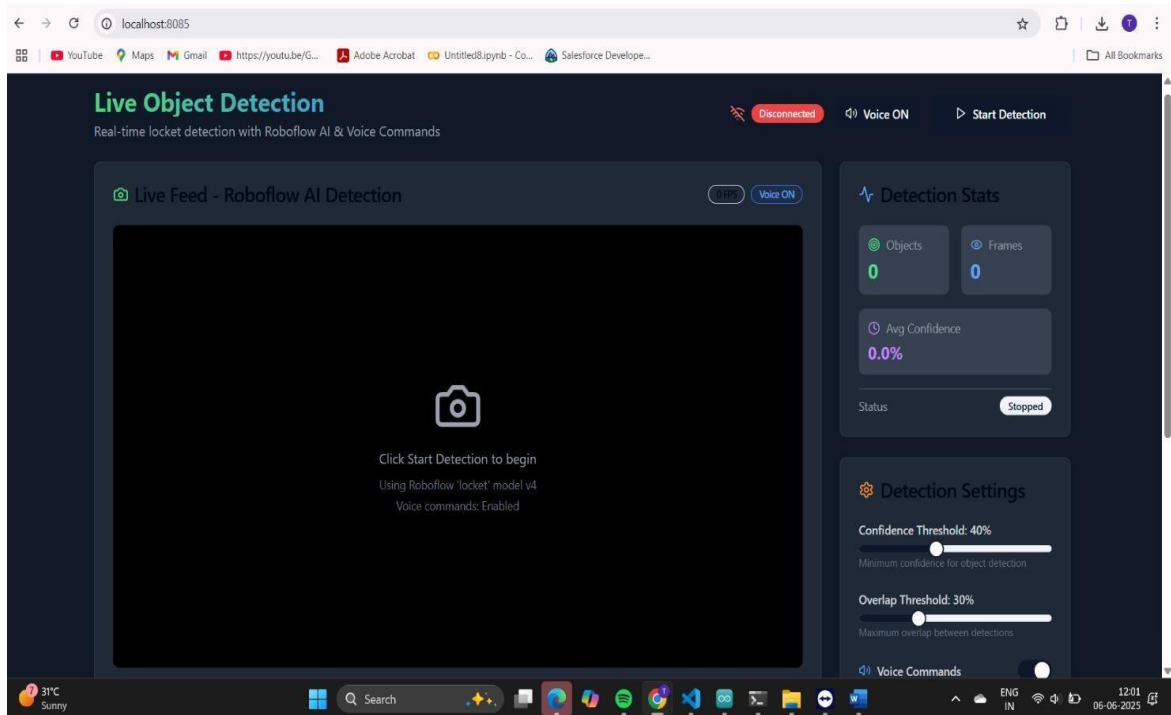


Fig. 6.6: Live Object Detection Web Application Interface

Fig. 6.6 shows the initial user interface of a "Live Object Detection" web application, accessed via <http://localhost:8085>. The page is designed for real-time object detection using Roboflow AI and voice commands. It features a central area for the "Live Feed," a "Start Detection" button, a "Disconnected" status indicator, and a "Voice ON" toggle. On the right, "Detection Stats" display metrics like detected objects, frames, and average confidence, while "Detection Settings" allow configuration of confidence and overlap thresholds. The application is currently in a "Stopped" status, awaiting user initiation.

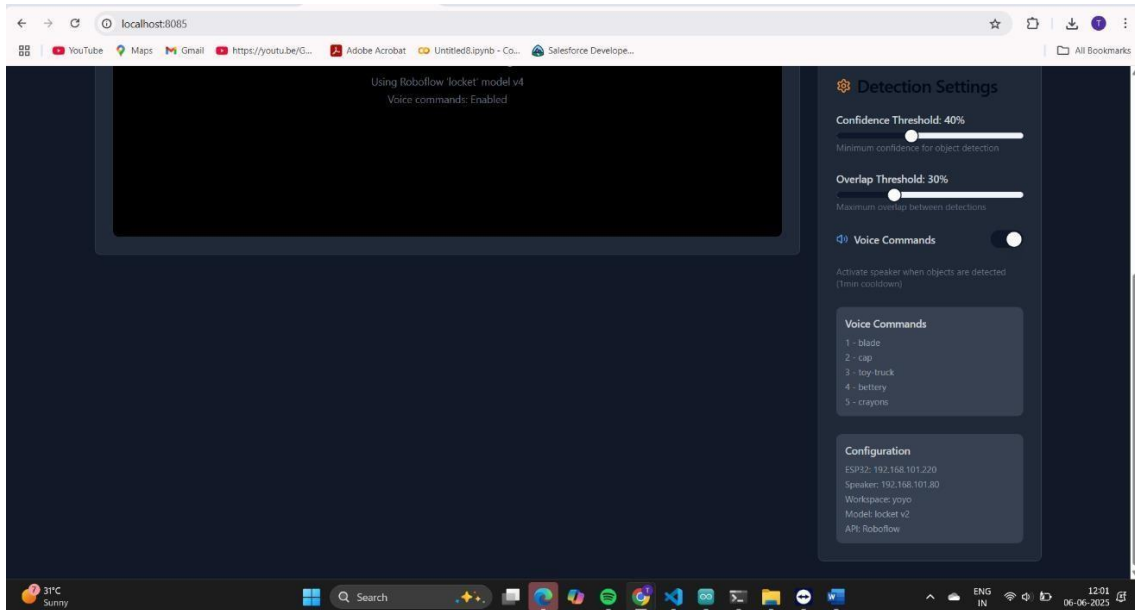


Fig. 6.7: Live Object Detection Web Application - Detailed Settings and Configuration

Fig. 6.7 presents a more detailed view of the "Live Object Detection" web application's settings and configuration section. This portion of the interface allows users to fine-tune detection parameters such as "Confidence Threshold" and "Overlap Threshold" using sliders. Crucially, it displays the list of recognized "Voice Commands" (e.g., blade, cap, toy-truck) that the system can process. Additionally, the "Configuration" section provides essential backend details, including the IP addresses of the connected ESP32 devices (192.168.101.220 and 192.168.101.80), the active Workspace, Model, and API source (Roboflow), offering transparency into the system's operational



Fig. 6.8: Real-time Object Detection of a Toy Truck in Action

Fig. 6.8 captures the "Live Object Detection" web application actively demonstrating its capabilities, displaying a real-time detection of a yellow toy truck. The image shows the physical toy truck positioned in front of a laptop, while the laptop screen simultaneously displays the live feed with a bounding box correctly identifying the object as "toy-truck" with a high confidence level of 99%. The "Detection Stats" on the right confirm that one object has been detected across 71 frames, illustrating the system's successful real-time performance. It gives out an output saying its harmful.



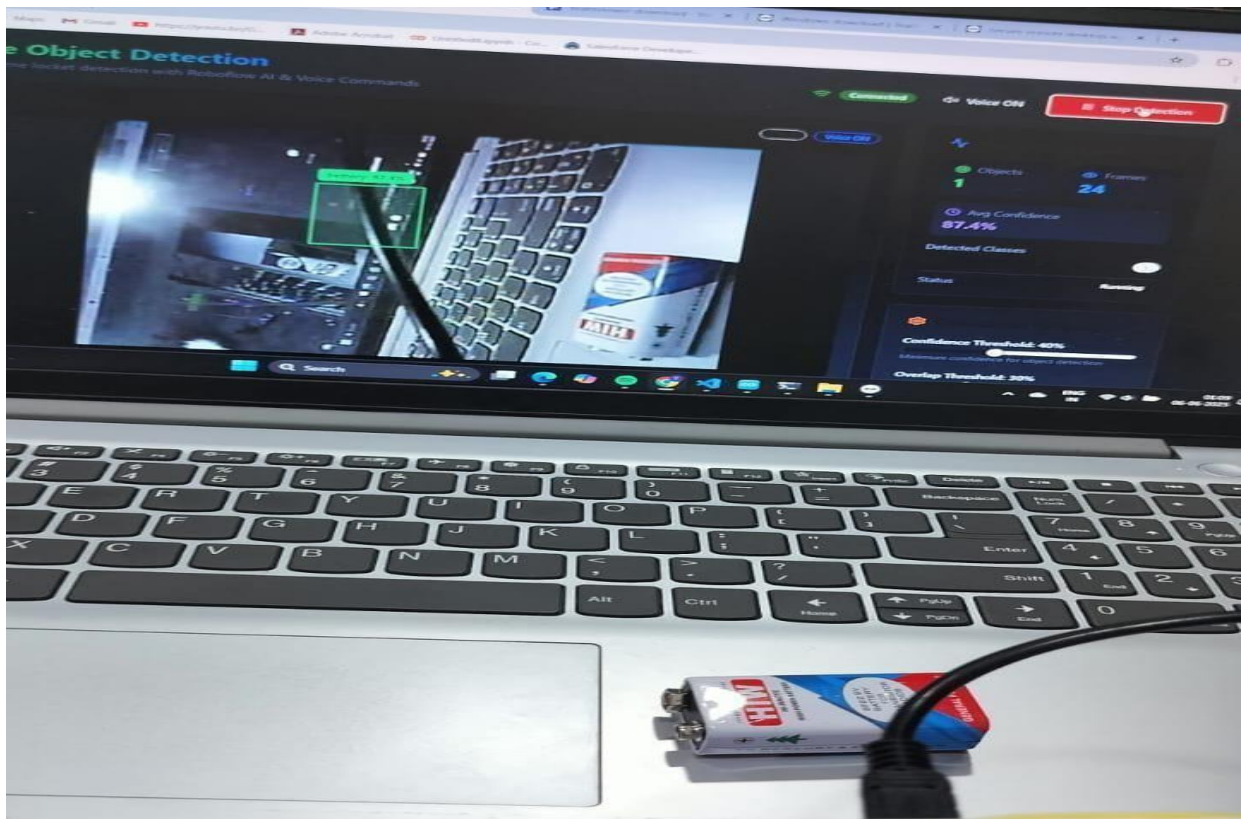


Fig. 6.9: Live Object Detection of a 9V Battery with Active Status

Fig. 6.9 illustrates the "Live Object Detection" web application successfully identifying a 9V battery in real-time. The image captures a physical battery connected via a cable resting on a laptop, while the laptop screen displays the live feed. A green bounding box with "battery" and "97.4%" confidence clearly indicates the detection. The interface shows "Connected" and "Voice ON" statuses, and the "Detection Stats" reflect "1 Object" detected across "24 Frames" with a high average confidence, confirming the system is actively "Running" and performing real-time object recognition. . It gives out an output saying its harmful.

## 7. CONCLUSION AND FUTURE ENHANCEMENTS

### 7.1.CONCLUSION

The **Object Detection Device for Kids Using ESP32-CAM** represents a significant step toward creating affordable, standalone, and intelligent embedded systems designed specifically for child safety and early learning environments. The system effectively bridges the gap between real-time object recognition and intuitive feedback using an audio-based output mechanism that caters to pre-literate users. By integrating computer vision and microcontroller-based audio playback, it promotes a safer exploratory environment for children without relying on complex user interfaces or internet connectivity.

This project was implemented by leveraging a two-part architecture that addressed the constraints of the ESP32-CAM's limited onboard memory. The image acquisition and streaming functionality were isolated to the ESP32-CAM, while the object detection process was handled externally through a Python-based YOLOv8 pipeline. Once an object was detected, a corresponding numeric ID was generated and transmitted via HTTP to a secondary ESP32 Devkit module. This second unit controlled a DFPlayer Mini to play voice messages based on object safety classification.

The system performed well under controlled and real-world conditions, with successful detection and accurate audio feedback for multiple household and classroom objects. Testing confirmed that the model's predictions were reliably transmitted, interpreted, and converted into appropriate spoken output, even with minimal hardware resources. Its offline capability, low cost, and modular architecture make it particularly valuable in rural areas, early education classrooms, and special-needs institutions where budget and simplicity are critical factors.

The overall outcome confirms that such devices can act not only as learning assistants but also as preventive safety tools for unattended child-object interactions. The integration of object detection and spoken feedback creates an experience that combines artificial intelligence, embedded control systems, and human-centric design in a compact and accessible solution.

## **7.2.FUTURE ENHANCEMENTS**

### **1. Real-Time Onboard Inference**

Implement quantized or lightweight models (e.g., TinyYOLO, MobileNet SSD) that can run partially or fully on the ESP32-CAM itself. This would reduce dependency on external processing and allow for a more integrated single-board system.

### **2. Dynamic and Personalized Audio Responses**

Integrate offline text-to-speech (TTS) engines, such as Talkie or ESP-Speech, to eliminate the need for pre-recorded .mp3 files. This would enable the system to generate dynamic, object-specific feedback and support future updates without needing SD card modification.

### **3. Multi-Language and Inclusive Communication**

Add support for regional languages or visual/audio feedback for children with special needs. Messages could be tailored using selectable profiles for different linguistic and cognitive levels.

### **4. Object Interaction Logging**

Extend the ESP32 Devkit functionality to log timestamped detections on an SD card or transmit data to a web dashboard. This would allow parents or teachers to monitor what objects children interact with and how frequently.

### **5. Parent or Teacher Mobile Interface**

Develop a companion mobile app or web dashboard where parents or teachers can:

## REFERENCES

- [1] Zhang, Wei, et al. "Lightweight CNN for object detection on embedded devices." *IEEE Sensors Journal* (2024).
- [2] Kumar, R., and P. Sharma. "ESP32-CAM based offline object detection with audio feedback." *2024 IEEE International Conference on Embedded Systems (ICES)*, 2024.
- [3] Lee, D., and H. Kim. "Real-Time Embedded Object Detection System Using Edge AI For Safety-Critical Applications." *Sensors* 23.4 (2023): 2121.
- [4] Gupta, A., and S. Mehta. "Low-Cost Object Detection System Using Edge AI for Educational Environments." *International Conference on Smart Systems and IoT (ICSSIoT)*, 2023.
- [5] Banerjee, A., and R. Paul. "Audio-Assisted Object Identification System for Pre-School Children at Home." *International Journal of Innovative Research in Computer and Communication Children.* *Journal of Educational Technology & Society* 26.1 (2023): 35–44.
- [6] Kadhim, Thair A., et al. "A face recognition application for Alzheimer's patients using ESP32-CAM and Raspberry Pi." *Journal of Real-Time Image Processing* 20.5 (2023): 100.
- [7] Vinod, Sredha, et al. "Object detection using ESP32 cameras for quality control of steel components in manufacturing structures." *Arabian Journal for Science and Engineering* 48.10 (2023): 12741–12758.
- [8] Patel, M., and A. Jain. "Smart Voice-Enabled Object Recognition System for Children Using Microcontrollers." *International Journal of Advanced Computer Science and Applications (IJACSA)* 14.2 (2023): 98–104.
- [9] Ramesh, S., and S. Ramya. "Object Detection using Embedded Deep Learning on Microcontrollers for Safety Systems." *International Journal of Engineering Research & Technology (IJERT)* 11.5 (2022): 211–215.
- [10] Chen, X., et al. "Real-time Object Detection and Notification System for the Visually Impaired using Raspberry Pi and YOLOv3." *Procedia Computer Science* 184 (2021): 564–571.
- [11] Sharma, V., and R. Kaur. "Smart Safety Alert System using Computer Vision for *Engineering* 9.4 (2021): 1452–145

