JETIR.ORG ISSN: 2349-5162 | ESTD Year : 2014 | Monthly Issue
JOURNAL OF EMERGING TECHNOLOGIES AND INNOVATIVE RESEARCH (JETIR)
An International Scholarly Open Access, Peer-reviewed, Refereed Journal

Object identification using ESP32-CAM and Google Vision API: A cloud-assisted embedded system

[1]Mrs. Asha R, [2]Ms. Thanmaya K P, [2]Mr. Tejas R,[2]Mr. Vishal H S, [2]Mr. Tejas P S

[1]Assistant Professor, [2]Student, Dept.of ECE
[1]Department of Electronics and Communications Engineering,
Vidya Vikas Institute of Engineering and Technology, Mysuru, Karnataka, India

Abstract : The Object Identification project leverages the ESP32-CAM module combined with cloud-based image processing via the Google Vision API to develop a real-time object detection system. It integrates hardware components like the ESP32-CAM, OLED display, and power supply with software including Node.js for server management and Google Vision API for image analysis. The ESP32-CAM captures images with its OV2640 camera and sends them to a Node.js server, which forwards them to the Vision API for object identification. The API returns labels with confidence scores, which the ESP32-CAM displays on the OLED screen in real time. This system supports live feeds, offering applications in surveillance, assistive technologies, and industrial automation. Node.js efficiently handles communication and data formatting between devices and the cloud. The modular architecture allows future enhancements like motion sensors or advanced recognition models. Combining low-cost hardware with cloud AI, the project demonstrates scalable, intelligent object detection for practical real-world use cases. Additionally, the system's lightweight embedded design ensures portability and low power consumption, making it suitable for deployment in remote or resource-constrained environments. The use of cloud-based AI eliminates the need for heavy local computation, reducing hardware costs while maintaining high accuracy. This approach also enables easy updates and improvements to the detection algorithms without modifying the embedded hardware.

IndexTerms **-ESP32-CAM, Google Vision API, Object Identification, Node.js server**

I. INTRODUCTION

Object detection and recognition refer to the ability of a system to identify and locate objects within digital images or video streams. It plays a crucial role in numerous applications, such as surveillance, autonomous vehicles, and augmented reality. Traditional object detection systems often rely on powerful computing platforms with dedicated graphics processing units (GPUs) to achieve real-time performance. However, these systems can be expensive, power-hungry, and less portable [2].

Recent advancements in embedded systems and cloud computing have opened up new possibilities for object detection using low-cost hardware and cloud-based services. This project aims to leverage these advancements to develop an object identification system using the ESP32-CAM module and the Google Cloud Vision API [4][6].

The ESP32-CAM is a low-cost development board that features a camera module and built-in Wi-Fi capabilities. It is capable of capturing images and transmitting them over a network, making it suitable for various Internet of Things (IoT) applications [3]. The Google Cloud Vision API is a cloud-based image analysis service that can detect and label objects within images using machine learning models [6].

The goal of this project is to create a real-time object identification system that captures images using the ESP32-CAM, sends them to a server running a Node.js application, and utilizes the Google Cloud Vision API to identify and label objects within the images. The results are then displayed on an OLED screen connected to the ESP32-CAM. This system combines the affordability and versatility of embedded hardware with the power of cloud-based image analysis to provide an efficient and scalable solution for object identification.

II. Literature Review

Many methods are used in real-time object detection. Some of the methods are: YOLO, YOLO-LITE, and Edge YOLO.

YOLO-LITE is a light-weight object detection model designed for real-time performance on devices with limited computational resources. It is a simplified version of the original YOLO (You Only Look Once) architecture and is optimized for faster inference without significantly compromising accuracy. YOLO-LITE is suitable for embedded systems, mobile devices, and IoT applications where computational efficiency is crucial [3].

YOLOv5 is a highly popular and efficient object detection model developed by Ultralytics. It offers improved accuracy and speed compared to previous versions of YOLO. YOLOv5 supports a variety of model sizes (e.g., YOLOv5s, YOLOv5m, YOLOv5l, YOLOv5x), allowing users to choose the best trade-off between performance and computational cost. It is widely used in real-world applications due to its robustness and ease of deployment [9].

Edge YOLO is a variant of the YOLO object detection model optimized for edge devices. It aims to deliver high-speed inference and low-latency performance on hardware with limited processing power. Edge YOLO achieves this by employing lightweight network architectures and pruning techniques to reduce model size while maintaining accuracy. It is ideal for applications requiring on-device processing, such as smart cameras, drones, and autonomous vehicles [12].

III. Methodology And Implementation

The proposed object recognition system combines embedded hardware with cloud-based image analysis to enable real-time object detection. It consists of four core components: the ESP32CAM module for image capture, a Node.js server for image handling, the Google Cloud Vision API for detection, and an OLED display for visual output. The architecture is illustrated in Figure 1. [1] [2] [12].

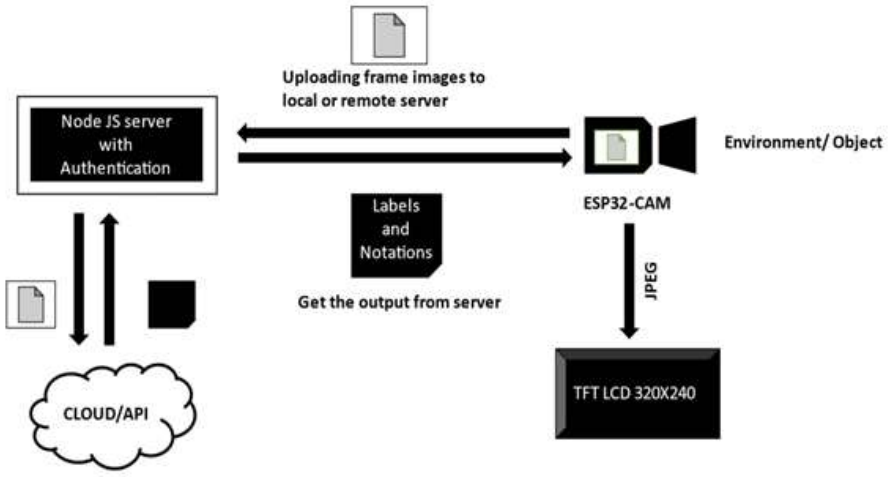

Figure 1: Block Diagram of the Object Recognition System

A. System Architecture Overview

- **ESP32-CAM Module:** Captures images using the onboard OV2640 camera and transmits them wirelessly over HTTP to a local server. It uses built-in Wi-Fi for network communication and acts as the system's sensing unit.
- **Node.js Server:** A lightweight server built using Node.js listens for incoming images. It encodes the images in base64 format and sends them to the Google Cloud Vision API. It also handles the API response and forwards object labels to the ESP32-CAM.
- **Google Cloud Vision API:** Deployed on the Google Cloud Platform, this API performs object detection and returns labels and confidence scores in JSON format. It significantly reduces processing load on the microcontroller.
- **OLED Display:** A 128x64 I2C OLED module displays the detected objects and their confidence levels in real time, offering user-friendly feedback.

B. Development Workflow

The following sequential process was followed:

- The ESP32-CAM captures and compresses the image (JPEG format).

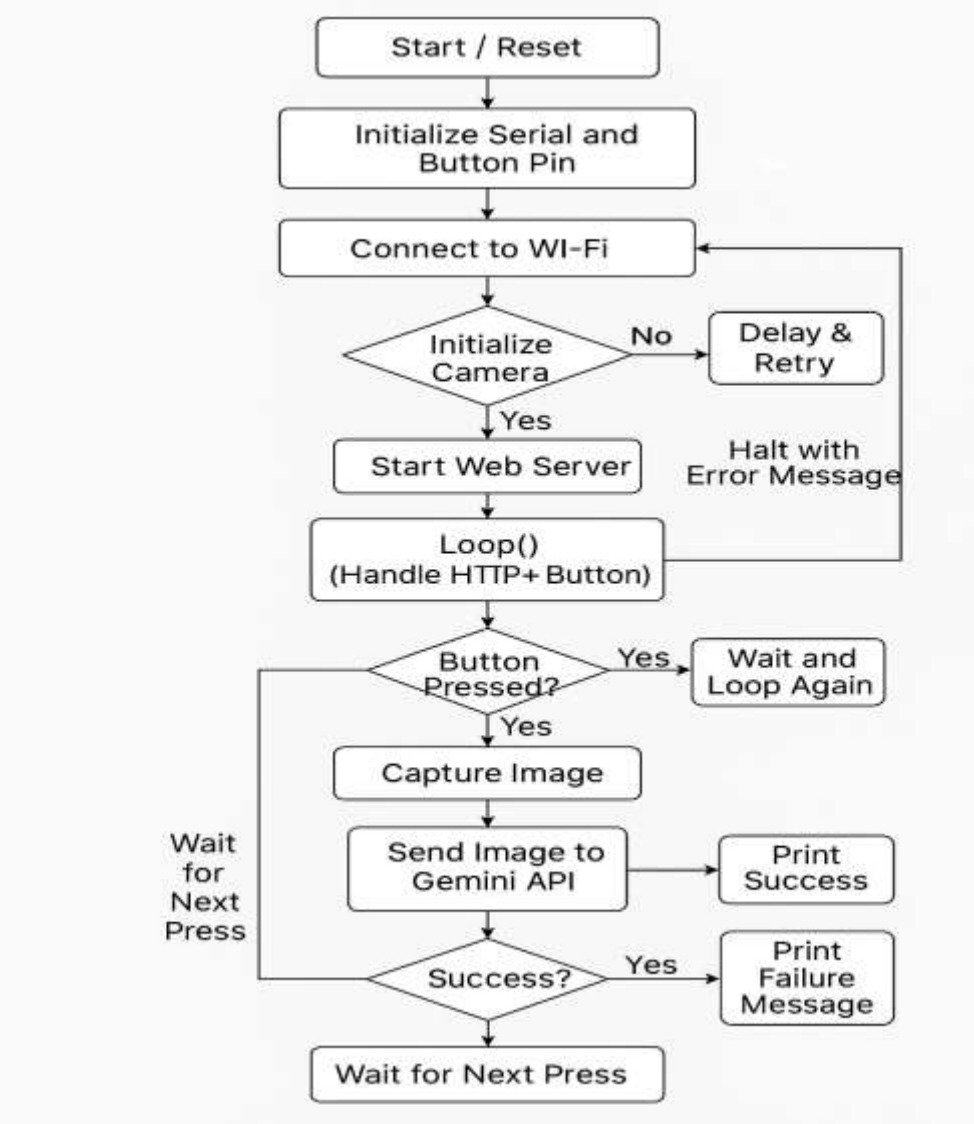

Figure 2: Flowchart of the Node.js Server Handling Image Upload and API Interaction

- It transmits the image to the Node.js server using HTTP POST.
- The server converts the image to base64 and sends it to the Google Vision API.
- Upon receiving object labels, the server parses the response.
- The labels are sent back to the ESP32-CAM and displayed on the OLED.

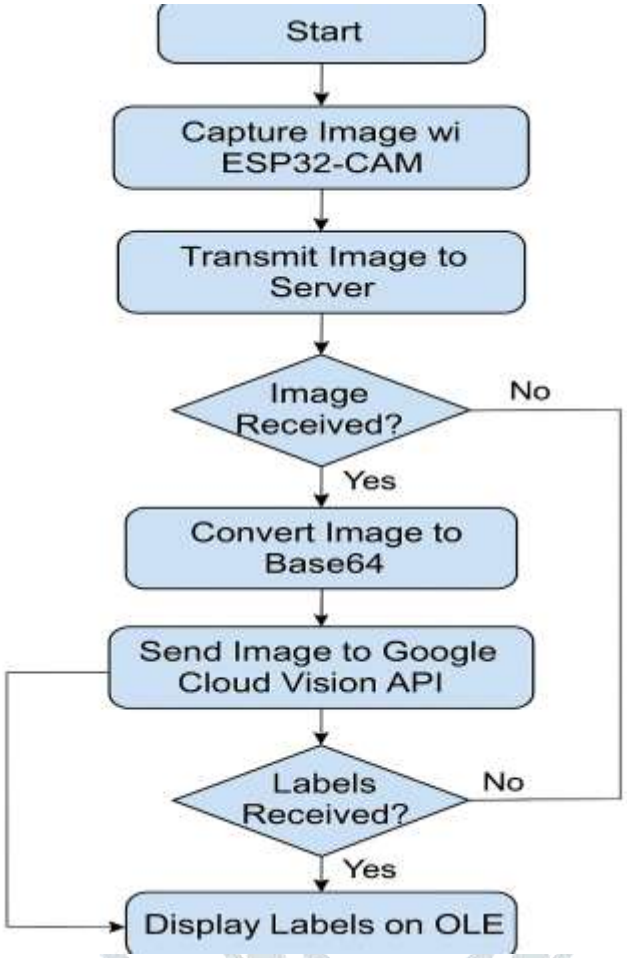

Figure :3 Flowchart of the Arduino IDE Handling ESP32 CAM and OLED

This complete cycle averages 1.7 seconds, demonstrating low latency and suitability for real-time applications. The implementation ensures efficient use of bandwidth and processing resources.

C. Tools and Technologies Used

- Hardware: ESP32-CAM, OLED Display (128x64, I2C)
- Software: Arduino IDE, Node.js v18.16.0
- Libraries: Express.js, Axios, Adafruit GFX, SSD1306, Wi-Fi libraries for ESP32
- Cloud Services: Google Cloud Vision API v1

D. Optimization and Performance

To enhance responsiveness, image resolution was optimized, and the maximum number of labels from the Vision API was limited. These adjustments ensured real-time operation with minimal bandwidth usage and reliable detection performance.

The detailed workflow of the server-side logic, from receiving the image to sending back the labels, is illustrated in Figure 2 & 3, which represents the backend code flowchart.

IV. Results and Discussion

The *Results and Discussion* section presents the outcomes of the object identification system using the ESP32-CAM and Google Vision API. It evaluates the system's performance under different test conditions, focusing on accuracy, efficiency, and robustness in real-time object recognition. Various scenarios such as dynamic environments, lighting conditions, background complexity, and resolution variations were tested to assess the overall system reliability and limitations [5].

A. Experimental Setup and Testing

To validate the robustness of the proposed system, the setup was tested under controlled and real-world environments. The experiments were designed to cover a wide range of conditions including object movement, background clutter, and varying light levels [10].

Test Variables:

- **Object Types**
  - *Static*: Books, tools, containers.
  - *Dynamic*: Moving people and vehicles which introduced motion blur, reducing detection accuracy.
- **Backgrounds**
  - *Clean*: Plain surfaces helped in better detection.
  - *Cluttered*: Complex backgrounds introduced visual noise, resulting in occasional false positives.
- **Lighting Conditions**
  - *Daylight*: Delivered best recognition accuracy.
  - *Dim Light / Artificial*: Shadows and reflections caused moderate inconsistencies in detection.
- **Distances and Resolutions**
  - Resolutions tested: 320x240, 640x480, and 1280x720.
  - Detection at far distances showed reduced accuracy in low-resolution settings.

B. Result Analysis

The outcomes were assessed using three core performance metrics the results are shown in Figure 4, 5 and 6.

- **Detection Accuracy**: Varied based on lighting, motion, and resolution. The system showed high reliability in optimal conditions.
- **Response Time**: End-to-end latency between image capture and label display was minimal (typically under 2 seconds), making the system nearly real-time.
- **Impact of Image Quality**: High-resolution images enhanced accuracy but increased processing time. Medium resolution (640x480) offered the best balance.

Figure :4 Detection of Dynamic Objects

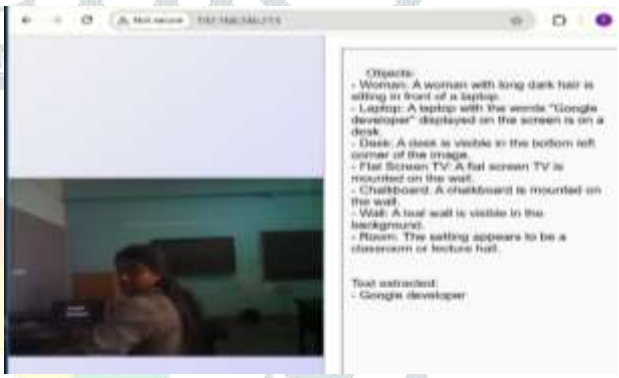

Figure :5 Object Detection in Cluttered Backgrounds

Figure :6 Detection Under Varying Lighting Conditions

C. Summary Table: Accuracy Under Different Test Scenarios

Table 1: Object Detection Accuracy Under Various Test Scenarios

Test Scenario	Object Type	Lighting	Background	Resolution	Accuracy
Controlled Environment	Static	Daylight	Clean	640x480	95–98%
Variable Lighting	Static	Mixed (Dim/Bright)	Clean	640x480	80–85%
Cluttered Background	Static	Daylight	Cluttered	640x480	70–80%
High Contrast Background	Static	Daylight	High Contrast	640x480	75–82%
Dynamic Objects	Moving	Daylight	Clean	1280x720	78–84%
Dim Light	Static	Dim	Clean	1280x720	75–80%
Far Distance + Low Resolution	Static	Daylight	Clean	320x240	60–70%
Near Distance + High Resolution	Static	Daylight	Clean	1280x720	98–99%
Realistic Cluttered Room	Static/Dynamic	Mixed	Cluttered	640x480	70–80%

V.Challenges and Applications

The implementation of an object detection system using the ESP32-CAM and Google Vision API, while innovative and effective, presents several challenges that affect its performance in real-time environments.

Challenges Faced

- Computational Limitations: The ESP32-CAM, while cost-effective and compact, has limited processing power and memory. This restricts its ability to perform on-device AI computations, making it heavily dependent on cloud services for object detection tasks.
- Communication Delays: The need to transmit captured images over Wi-Fi to a Node.js server and then to the Google Vision API introduces latency. Network fluctuations or poor connectivity can delay responses and reduce overall system efficiency.

- Lighting Conditions: Varying lighting environments significantly impact detection accuracy. In dim or mixed lighting, the quality of captured images decreases, leading to misidentifications or lower confidence scores.
- Motion and Background Clutter: The presence of dynamic objects or visually complex backgrounds introduces challenges such as motion blur or visual noise. This reduces the system's ability to isolate and identify key objects.

Applications and Use Cases

- Surveillance Systems: The system can be adapted for low-cost surveillance solutions in homes, small offices, or restricted areas, where real-time object identification is needed.
- Assistive Technology: It can aid visually impaired individuals by identifying and vocalizing nearby objects, improving daily accessibility.
- Industrial Monitoring: The camera setup can help track specific tools, parts, or safety gear on manufacturing floors, enhancing operational safety and efficiency.
- Educational Tools: This project demonstrates practical integration of embedded systems and AI, making it ideal for academic use in electronics and computer science curricula.

VI.Conclusion and Future Scope

This project has successfully demonstrated the implementation of a real-world object identification system using the ESP32-CAM in conjunction with Google's Vision API as in Figure 7. The integration of hardware and cloud-based software allowed for efficient detection and labelling of various objects in different environmental conditions. Tests conducted under varied scenarios—ranging from controlled lighting to complex, cluttered environments—showed that the system performs reliably, especially with static objects and under good lighting conditions. The latency between image capture and display was minimal, and the addition of a flash module further enhanced image clarity in low-light situations. The project validates the feasibility of combining embedded microcontrollers with advanced cloud-based AI services to build low-cost, scalable, and practical solutions. The system's adaptability, speed, and ease of deployment make it a strong candidate for applications in security, accessibility, and industrial automation.

Figure 7. Google vision API & ESP32 CAM

For future improvements, the system can be enhanced with local AI inference (e.g., using TensorFlow Lite or Edge Impulse), sensor fusion (integrating motion or proximity sensors), and larger or touch-enabled displays for better user interaction. Additionally, reducing reliance on cloud infrastructure by shifting some processing to the edge can lower latency, improve privacy, and make the system more robust in offline scenarios. These advancements will help expand its use in real-time applications and make it more sustainable for long-term deployment.

VII.References

[1] R. Pakdel and J. Herbert, “A Cloud-based Data Analysis Framework for Object Recognition,” 2024.

[2] Z. Zou, K. Chen, Z. Shi, Y. Guo, and J. Ye, “Object Detection in 20 Years: A Survey,” 2024.

[3] A. Guo, K. Sun, and Z. Zhang, “A Lightweight YOLOv8 Integrating FasterNet for Real-Time Underwater Object Detection,” 2023.

[4] D. Lorenčík and P. Sinčák, “Cloud-based Object Recognition: A System Proposal,” 2023.

[5] R. Al Sobbahia and J. Teklia, “Comparing Deep Learning Models for Low-Light Natural Scene Image Enhancement and Their Impact on Object Detection and Classification,” 2022.

[6] S. Vinod, P. Shakor, F. Sartipi, and M. Karakouzian, “Object Detection Using ESP32 Cameras for Quality Control of Steel Components in Manufacturing Structures,” 2022.

[7] J. Jing, S. Liu, G. Wang, W. Zhang, and C. Sun, “Recent Advances on Image Edge Detection: A Comprehensive Review,” 2022.

[8] S. S. A. Zaidi, M. S. Ansari, A. Aslam, N. Kanwal, M. Asghar, and B. Lee, “A Survey of Modern Deep Learning Based Object Detection Models,” 2021.

[9] B. Heisele, I. Riskov, and C. Morgenstern, “Components for Object Detection and Identification,” 2021.

[10] Z.-Q. Zhao, P. Zheng, S. Xu, and X. Wu, “Object Detection with Deep Learning: A Review,” 2021.

[11] I. Kilic and G. Aydin, “Traffic Sign Detection and Recognition Using TensorFlow’s Object Detection API with a New Benchmark Dataset,” in Proc. Int. Conf. Electrical Engineering (ICEE), Istanbul, Turkey, 2020.

[12] N. Mehendale, “Object Detection Using ESP32 CAM,” 2020.

[13] S. Liang, H. Wu, L. Zhen, Q. Hua, S. Garg, G. Kaddoum, M. M. Hassan, and K. Yu, “Edge YOLO: Real-Time Intelligent Object Detection System Based on Edge-Cloud Cooperation in Autonomous Vehicles,” 2019.

[14] J. Jing, S. Liu, G. Wang, W. Zhang, and C. Sun, “Recent Advances on Image Edge Detection: A Comprehensive Review,” 2019.

[15] Y. Amit, P. Felzenszwalb, and R. Girshick, “Object Detection,” 2018.

[16] A. Howard et al., “MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications,” 2017.

[17] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, and others, “Going Deeper with Convolutions,” 2015.

[18] Google Cloud, “Google Cloud Vision API Documentation,” Google, 2024.

[19] M. Abadi et al., “TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems,” 2016.

[20] H. Caesar, V. Bankiti, A. H. Lang, et al., “nuScenes: A Multimodal Dataset for Autonomous Driving,” 2020.