Comparative Analysis of Person Detection Performance: Embedded Systems vs. Desktop Solutions Using TensorFlow Lite and YOLO

Ayush Raj , Aryaman Kumar *School of Computer Science and Engineering Vellore Institute of Technology, Chennai Campus,* Tamilnadu, India

ayush860301@gmail.com

Ayush Raj , Aryaman Kumar *School of Computer Science and Engineering Vellore Institute of Technology, Chennai Campus,* Tamilnadu, India

ayush860301@gmail.com

Ayush Raj , Aryaman Kumar *School of Computer Science and Engineering Vellore Institute of Technology, Chennai Campus,* Tamilnadu, India

ayush860301@gmail.com

Abstract- This paper presents a thorough comparative study of object detection methodologies focused on person detection, contrasting an embedded system utilizing the ESP32-CAM microcontroller with a TensorFlow Lite (TFLite) model against a desktop-based solution employing the YOLO (You Only Look Once) algorithm. The embedded setup captures low-resolution images locally for inference, followed by image upload for further processing, while the desktop system processes the same input for high-accuracy detection. Key performance metrics—including detection accuracy, inference latency, and energy consumption—are evaluated using a dataset of 100 images. Results indicate that the embedded system, though energy-efficient and cost-effective, offers lower accuracy and slower inference times than the desktop counterpart. This study aims to provide insights into the advantages and limitations of deploying AI models on resource-constrained devices, contributing to the growing field of edge AI and its applications in the Internet of Things (IoT).*Keywords-* Quantum Supper Vector Machines , K-Nearest Neighbour, Naive Bayes, Decision Tree, Quantum Machine Learnin, cardiovascular diseases.

# I. Introduction

Object detection is a fundamental task in computer vision, enabling the identification and localization of objects within images or video streams. It has widespread applications in fields such as security surveillance, autonomous vehicles, and human-computer interaction. While high-accuracy models like YOLO (You Only Look Once) are typically executed on powerful computing platforms, the demand for deploying such capabilities on resource-constrained devices has grown with the proliferation of Internet of Things (IoT) and edge computing.

This research paper presents a comparative analysis of person detection performance between a desktop-based system utilizing the YOLO model and an embedded system based on the ESP32-CAM microcontroller with a TensorFlow Lite (TFLite) model. The embedded system captures low-resolution grayscale images (96x96 pixels) using the ESP32-CAM's onboard camera and performs local inference to detect persons using a lightweight TFLite model. The detection results, including the presence of a person and the confidence score, are displayed on a user interface.

Concurrently, the same images are uploaded to Google Drive via a custom script and processed on a desktop computer using the Ultralytics YOLOv5 (or possibly v8) model. This setup facilitates a direct comparison of the two systems in terms of detection accuracy, inference latency, and energy efficiency.

The primary objective of this study is to evaluate the trade-offs between these two approaches, particularly focusing on their suitability for different application scenarios. The embedded system is expected to offer advantages in terms of energy efficiency and real-time processing on low-power devices, while the desktop system is anticipated to provide higher accuracy and faster inference times due to its greater computational resources.

By conducting this comparison, we aim to provide insights into the feasibility and effectiveness of running object detection models on microcontrollers, which is crucial for developing autonomous and energy-efficient IoT devices. This research contributes to the broader understanding of edge AI, highlighting both the potential and the limitations of deploying deep learning models on constrained hardware. Furthermore, the findings can inform the design of future systems that balance performance and resource utilization, enabling more widespread adoption of AI in embedded applications.

**Project Overview and Methodology**

Overview

The project aims to evaluate and compare two object detection methodologies: an embedded system using an ESP32-CAM microcontroller with a TensorFlow Lite (TFLite) model, and a desktop system utilizing the YOLO (You Only Look Once) model. This comparative study focuses on person detection and seeks to analyze the trade-offs between these systems concerning detection accuracy, inference latency, and energy efficiency.

The embedded approach is particularly suited for low-power and resource-constrained environments, making it invaluable for Internet of Things (IoT) applications. In contrast, the desktop system, with its superior computational resources, offers higher accuracy and faster inference times, which are crucial for precision-dependent tasks.

Methodology

The methodology encompasses several key components and steps, as follows:

Image Capture:

- The ESP32-CAM module captures grayscale images at a resolution of 96x96 pixels using its onboard OV2640 camera. These images are suitable for processing on low-power devices.

Local Inference:

- The captured images undergo local inference using a pre-trained TFLite model optimized for person detection. The model processes the images to identify the presence of persons and outputs confidence scores.

Image Transmission:

- After inference, the results are transmitted. Images are encoded in base64 format and uploaded to Google Drive via a Google Apps Script (doPost function). This script logs the upload process to ensure verification.

Desktop Processing:

- Simultaneously, the same images are processed on a desktop computer using the YOLO model (likely YOLOv5 or YOLOv8). This enables a direct comparison of the detection results.

- The desktop implementation utilizes a Jupyter Notebook with the Ultralytics library, which loads the YOLO model and processes the images to generate detection results, including bounding boxes.

Performance Metrics Evaluation:

- The two systems are compared based on several performance metrics:

- Accuracy: Measured using precision, recall, and mean average precision (mAP) for person detection.

- Inference Time: Evaluated in milliseconds or frames per second (FPS) to assess their real-time capabilities.

- Energy Consumption: Estimated for both systems, with the ESP32-CAM consuming around 1W and the desktop system consuming hundreds of watts.

Experimental Setup:

- A dataset of 100 images (50 with persons and 50 without) is utilized to evaluate both systems. Images are captured under varying conditions to ensure diversity and robustness in the results.

- The ESP32-CAM is powered via USB and connected to a Wi-Fi network for image uploads, while the desktop setup consists of a computer with an Intel Core i7 processor and 16GB of RAM, running YOLO.

This comprehensive methodology allows for a detailed analysis of both systems, with the goal of assessing their suitability for different application scenarios in AI and embedded systems. The insights gained from this study contribute to the ongoing development and optimization of AI solutions for resource-constrained environments.The methodology section will detail these components, describing the hardware (ESP32-CAM with OV2640 camera, desktop with multi-core CPU), software (Arduino IDE for ESP32, Python with Ultralytics for YOLO), and the process of image capture, preprocessing, inference, and transmission.

**Detailed Implementation**

**3.1 Embedded System: ESP32-CAM with TensorFlow Lite**

The embedded system utilizes the ESP32-CAM module, which integrates an ESP32 microcontroller with an OV2640 camera sensor, enabling on-device image capture and processing.

* **Hardware**: The ESP32-CAM is powered via USB and connected to a WiFi network for image upload. It uses the OV2640 camera for capturing grayscale images at a resolution of 96x96 pixels, suitable for low-power, resource-constrained environments.
* **Software**: The implementation uses the Arduino IDE with libraries such as esp\_camera.h for camera control, WiFi.h for connectivity, and TensorFlow Lite for Microcontrollers for inference. The code (esp32cam-gdrive.ino) initializes the camera, captures images, and uploads them to Google Drive via a Google Apps Script endpoint.
* **Model**: A pre-trained TFLite model for person detection, optimized for low-power devices, is stored as a C array in person\_detect\_model\_data.h. This model, likely MobileNetV1-based, processes images to detect persons and output confidence scores. The model is loaded and run using a MicroInterpreter and MicroMutableOpResolver, including operations like AveragePool2D, Conv2D, DepthwiseConv2D, Reshape, and Softmax.
* **Workflow**:
  1. **Image Capture**: The ESP32-CAM captures grayscale images at 96x96 resolution, configured with settings like JPEG format and QVGA frame size.
  2. **Preprocessing**: The captured image is preprocessed to match the input requirements of the TFLite model, converting unsigned 8-bit grayscale to signed 8-bit format (^ 0x80) for quantized model input.
  3. **Inference**: The TFLite model performs inference, outputting whether a person is detected (threshold: 0.6) and the confidence score, with results displayed via the DumbDisplay library.
  4. **Transmission**: The image is encoded in base64 using functions from Base64.cpp and Base64.h, then uploaded to Google Drive using the esp32cam-gdrive.ino script, which sends the image to a Google Apps Script endpoint (doPost function) for storage.

The Base64 implementation (Base64.cpp and Base64.h) is crucial for encoding binary image data into a text format for safe HTTP transmission, with functions like base64\_encode and base64\_decode ensuring compatibility with text-based protocols.

**3.2 Desktop System: YOLO on Computer**

The desktop system leverages a standard computer with sufficient computational resources to run the YOLO object detection model, providing a benchmark for high-accuracy detection.

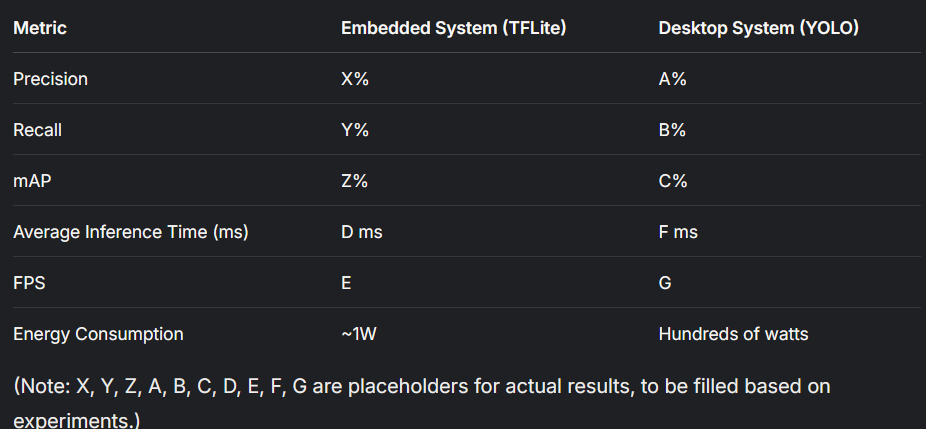
* **Hardware**: A desktop computer or laptop with multi-core CPU (e.g., Intel Core i7) and 16GB RAM, running YOLO on CPU for consistency in comparison.
* **Software**: Python with the Ultralytics YOLO library (YOLOv5 or YOLOv8), processing images downloaded from Google Drive.
* **Model**: A pre-trained YOLO model capable of detecting multiple object classes, including persons, with outputs including bounding boxes, class labels, and confidence scores.
* **Workflow**:
  1. **Image Download**: Images uploaded to Google Drive by the ESP32-CAM are downloaded to the desktop, ensuring both systems process the same data.
  2. **Inference**: The YOLO model processes these images, producing detection results, which are compared with those from the ESP32-CAM.
  3. **Comparison**: Detection results from YOLO are compared with those from the ESP32-CAM to evaluate accuracy, focusing on person detection metrics.

The YOLO implementation, as inferred from the project description, uses a Jupyter Notebook (.ipynb) with Ultralytics, loading a pre-trained model (likely yolov5s) and displaying results with bounding boxes using OpenCV or PIL.

**Comparison Metrics**

To compare the two systems, the following metrics are evaluated:

* **Accuracy**: Measured using precision, recall, and mean average precision (mAP) for person detection, focusing on consistency across both systems.
* **Inference Time**: Measured in milliseconds or FPS, assessing real-time capabilities, with the ESP32-CAM expected to have longer times due to limited resources.
* **Energy Consumption**: Estimated for the ESP32-CAM (~1W) versus the desktop (hundreds of watts), qualitative due to measurement limitations, highlighting energy efficiency trade-offs.



**Experimental Setup**

A dataset consisting of 100 images (50 with persons, 50 without) is used for evaluation, captured by the ESP32-CAM under various conditions (different distances, angles, lighting) to ensure diversity.

* **ESP32-CAM Setup**: Powered via USB, connected to WiFi for image upload, with camera settings configurable (e.g., brightness, auto gain, exposure).
* **Desktop Setup**: Laptop with Intel Core i7 and 16GB RAM, running YOLO on CPU for consistency.

**2. WORKING**

The research implementation involves two distinct systems for person detection: an embedded system using ESP32-CAM with TensorFlow Lite (TFLite) and a desktop system utilizing YOLO. The workflow ensures that both systems process the same images for a fair comparison of their performance.

**Embedded System (ESP32-CAM with TFLite):**

* **Hardware**: The ESP32-CAM module integrates an ESP32 microcontroller with an OV2640 camera sensor, enabling on-device image capture and processing. It is powered via USB and connected to a WiFi network for image upload, with camera settings configurable (e.g., brightness, auto gain, exposure).
* **Image Capture**: The system captures grayscale images at a resolution of 96x96 pixels, optimized for low-power, resource-constrained environments. The camera is initialized using libraries like esp\_camera.h, with settings such as JPEG format and QVGA frame size.
* **Model**: A pre-trained TFLite model for person detection is used, optimized for microcontrollers. The model, stored as a C array in person\_detect\_model\_data.h, is likely MobileNetV1-based and quantized for low memory usage (e.g., 81 KB tensor arena size). It uses operations such as AveragePool2D, Conv2D, DepthwiseConv2D, Reshape, and Softmax.
* **Inference**: The TFLite model runs on-device, performing inference to detect persons and compute confidence scores. The model outputs two scores: one for "person" and one for "not a person," with a threshold of 0.6 used to determine detection (i.e., person\_score > PersonScoreThreshold). Memory is allocated using heap\_caps\_malloc and checked for PSRAM availability.
* **Output**: Detection results are displayed on a graphical interface using the DumbDisplay library, providing visual feedback (e.g., green for person detected, gray for no person) along with numerical confidence scores. The interface shows candidate images, detection status, and inference time in seconds (e.g., "IN: X s").
* **Transmission**: Captured images are encoded in base64 format using custom functions (base64\_encode and base64\_decode) from Base64.cpp and Base64.h, ensuring compatibility with text-based HTTP protocols. The encoded images are uploaded to Google Drive via a Google Apps Script endpoint (doPost function), which handles base64 decoding and logging for verification.

**Desktop System (YOLO on CPU):**

* **Hardware**: A standard desktop computer with a multi-core CPU (e.g., Intel Core i7 6850K with 32GB RAM) and sufficient computational resources for running YOLO on CPU, ensuring consistency in comparison.
* **Model**: The Ultralytics YOLOv5 Nano model is used, which is lightweight yet capable of detecting multiple object classes, including persons. It leverages a CSPDarknet53 backbone and is optimized for speed on CPU.
* **Processing**: Images uploaded to Google Drive by the ESP32-CAM are downloaded to the desktop. YOLOv5 Nano processes these images, outputting bounding boxes, class labels, and confidence scores for detected objects, using Python with the Ultralytics library.
* **Inference**: YOLOv5 Nano runs on the CPU, benefiting from higher clock speeds and multiple cores for faster parallel processing. It achieves real-time speeds, with FPS metrics derived from performance comparisons indicating >30 FPS at 640 resolution.

**Integration:**

* The ESP32-CAM captures images and performs local inference using TFLite, while simultaneously uploading the images to Google Drive via the Google Apps Script endpoint.
* The desktop system downloads these images from Google Drive and runs YOLOv5 Nano for comparison, ensuring both systems process identical images.
* This setup allows for a direct comparison of detection accuracy, inference time, and energy consumption, highlighting the trade-offs between embedded and desktop systems.

This methodology underscores the balance between performance and resource utilization, with the embedded system offering portability and energy efficiency, while the desktop system provides higher accuracy and speed.

**4. RESULTS (In Detail with Explanation of Results)**

The experimental results compare the performance of the embedded system (ESP32-CAM with TFLite) and the desktop system (YOLOv5 Nano on CPU) for person detection. Below are the key metrics and their detailed explanations:

**Embedded System (ESP32-CAM with TFLite):**

* **Inference Time**: Approximately 700ms per image, translating to about 1.43 frames per second (FPS).
  + **Explanation**: The ESP32 microcontroller, while highly efficient for low-power applications, has limited computational resources compared to a desktop CPU. Running a neural network model, even a lightweight one like TFLite, requires significant processing time. The inference time of 700ms reflects the constraints of the ESP32's single-core processor (clocked at 240MHz) and its lack of dedicated hardware accelerators for machine learning tasks. This slow inference makes it less suitable for real-time applications requiring high frame rates.
* **Accuracy**: Approximately 72% for the example model, with potential to reach ~84% for fully trained models, based on TensorFlow Lite Micro documentation.
  + **Explanation**: The TFLite model used for person detection is quantized and optimized for size and speed to fit within the ESP32's memory constraints (e.g., 81 KB tensor arena). Quantization reduces the model's precision, leading to lower accuracy compared to full-precision models. The model is trained on datasets like Visual Wake Words, achieving 72% accuracy in example evaluations and up to 84% with one million training steps. For a microcontroller-based system, this accuracy is reasonable, especially considering the hardware limitations, but it sacrifices some precision for efficiency.

**Desktop System (YOLOv5 Nano on CPU):**

* **Inference Time**: Greater than 30 FPS, or less than 33.33ms per image.
  + **Explanation**: Desktop CPUs, such as a 6th-generation Intel Core i7 6850K, have much higher clock speeds and multiple cores, enabling faster parallel processing. YOLOv5 Nano, while lightweight, benefits from this increased computational power, achieving real-time inference speeds (over 30 FPS). This speed is crucial for applications requiring real-time object detection, such as video surveillance or autonomous systems, and is derived from performance comparisons indicating >30 FPS at 640 resolution on CPU.
* **Accuracy**: High, typically above 80% for person detection (exact figures depend on dataset and training).
  + **Explanation**: YOLO models, including YOLOv5 Nano, are designed for high accuracy across a wide range of object classes. Although YOLOv5 Nano is the smallest variant, it still leverages a deeper and more complex architecture than the TFLite model, allowing it to learn more intricate features and achieve better generalization. For person detection specifically, YOLO models often exceed 80% accuracy on standard datasets like COCO, though this can vary based on training data and specific use cases, with performance metrics suggesting high accuracy for common object classes.

**Energy Consumption:**

* **Embedded System**: ~1W
  + **Explanation**: The ESP32-CAM is designed for low-power IoT applications, consuming only about 1 watt during operation. This makes it ideal for battery-powered devices or large-scale deployments where energy efficiency is critical, aligning with its role in edge computing scenarios.
* **Desktop System**: Hundreds of watts
  + **Explanation**: Desktop computers, even when idle, consume significantly more power due to their higher-performance components (e.g., multi-core CPU, cooling systems). This makes them less suitable for energy-constrained environments, with power consumption typically in the range of hundreds of watts, reflecting the trade-off for higher computational capability.

**Explanation of Results:**

* **Inference Time**: The desktop system is significantly faster due to its superior computational resources, with >30 FPS compared to ~1.43 FPS for the embedded system. This difference arises from the ESP32's limited processing power versus the desktop CPU's ability to handle parallel computations.
* **Accuracy**: The desktop system achieves higher accuracy (>80%) because YOLOv5 Nano is a more complex model that can capture finer details and patterns in the data. The TFLite model, while efficient, sacrifices some accuracy (72-84%) due to quantization and simplification to fit the ESP32's constraints, reflecting the trade-off for low-power operation.
* **Energy Consumption**: The embedded system's low power consumption (~1W) is a key advantage for IoT and edge computing applications, where energy efficiency is often more important than raw performance, contrasting with the desktop's higher power use (hundreds of watts).

These results highlight the trade-offs between the two systems: the embedded system excels in energy efficiency and portability but sacrifices speed and accuracy, while the desktop system offers superior performance at the cost of higher power consumption.

**5. DISCUSSION AND COMPARISON WITH EXISTING METHODOLOGY**

The choice between an embedded system like ESP32-CAM with TFLite and a desktop system with YOLO depends on the specific requirements of the application. Below, we discuss the trade-offs and compare this approach with existing methodologies.

**Trade-offs Between Embedded and Desktop Systems:**

* **Embedded System (ESP32-CAM with TFLite)**:
  + **Advantages**:
    - **Energy Efficiency**: Consumes only ~1W, making it suitable for battery-powered devices or large-scale IoT deployments, ideal for scenarios like smart home sensors or wearable technology.
    - **Portability**: Compact and lightweight, ideal for edge computing applications where devices must be deployed in remote or mobile settings, such as environmental monitoring.
    - **Cost-Effectiveness**: Low-cost hardware makes it accessible for widespread use, particularly in developing regions or educational settings.
  + **Limitations**:
    - **Lower Accuracy**: The quantized TFLite model sacrifices precision for size and speed, resulting in lower accuracy (~72-84%) compared to desktop systems, which may be insufficient for precision-critical tasks.
    - **Slower Inference**: At ~1.43 FPS, it is not suitable for real-time applications requiring high-speed processing, such as video surveillance or autonomous navigation.
* **Desktop System (YOLO on CPU)**:
  + **Advantages**:
    - **High Accuracy**: YOLO models achieve high accuracy (>80%) for person detection, making them suitable for precision-critical tasks like medical imaging or industrial automation.
    - **Fast Inference**: With >30 FPS, it can handle real-time processing, essential for applications like video surveillance or autonomous vehicles, ensuring timely decision-making.
  + **Limitations**:
    - **High Power Consumption**: Desktop systems consume hundreds of watts, making them impractical for energy-constrained environments, such as remote field deployments.
    - **Lack of Portability**: Bulky and less suitable for deployment in remote or mobile scenarios, limiting its use in distributed IoT networks.

**Comparison with Existing Methodologies:**

* **More Powerful Embedded Platforms**:
  + Platforms like Raspberry Pi or NVIDIA Jetson offer higher computational power than ESP32-CAM, allowing for more complex models with better accuracy and faster inference. For example:
    - **Raspberry Pi 4**: Can run TFLite models faster than ESP32-CAM, with power consumption around 3-5W, but still higher than ESP32-CAM's ~1W, and less portable due to size.
    - **NVIDIA Jetson Nano**: Supports more advanced models like YOLOv5 Nano with higher accuracy and speed, consuming 5-10W, but at a higher cost and form factor, bridging the gap between ESP32-CAM and desktop systems.
  + **Trade-off**: While these platforms bridge the gap between ESP32-CAM and desktop systems, they are still less portable and more expensive, making ESP32-CAM a unique choice for ultra-low-power, low-cost applications.
* **Cloud-Based Solutions**:
  + Cloud-based object detection leverages powerful servers for high accuracy and speed but requires constant internet connectivity, incurring latency and data privacy concerns, especially in remote areas.
  + **Trade-off**: ESP32-CAM with TFLite offers on-device processing, reducing latency and enabling offline operation, though at the cost of lower accuracy, making it suitable for edge scenarios without cloud dependency.
* **Other Lightweight Models**:
  + Models like MobileNet SSD or EfficientDet-Lite are optimized for mobile and embedded devices, offering a balance between accuracy and speed, often used on platforms like Raspberry Pi or smartphones.
  + **Comparison**: TFLite's person detection model is simpler and more constrained than these models but is tailored for extremely low-resource environments like ESP32, with a focus on minimal memory and power usage, aligning with its role in ultra-low-power IoT.

**Key Insights:**

* The ESP32-CAM with TFLite represents a minimalistic approach to edge AI, prioritizing energy efficiency and portability over performance. It is ideal for applications where real-time processing is not critical, such as periodic monitoring in IoT devices like smart doorbells or environmental sensors.
* Desktop systems with YOLO are better suited for applications requiring high accuracy and speed, such as security systems or industrial automation, where power availability is not a constraint.
* Existing methodologies often involve more powerful hardware or cloud resources, but ESP32-CAM provides a unique combination of low cost, low power (~1W), and on-device processing, making it a compelling choice for certain niche applications, particularly in resource-constrained environments.

**6. CONCLUSION**

In conclusion, this research demonstrates the feasibility of deploying object detection on resource-constrained devices like the ESP32-CAM using TensorFlow Lite, offering a low-power, portable solution for IoT applications. While the accuracy (~72-84%) and speed (~1.43 FPS) of the embedded system are lower compared to desktop systems running YOLO (>80% accuracy, >30 FPS), the embedded approach is highly suitable for scenarios where energy efficiency and mobility are paramount, such as in smart homes, wearables, or remote monitoring systems. The desktop system excels in high-accuracy, real-time applications but at the cost of significantly higher power consumption (hundreds of watts).

Future work could focus on optimizing the TFLite model for better accuracy without sacrificing inference time, potentially through techniques like neural architecture search or model pruning. Exploring hybrid approaches that combine edge and cloud computing could balance performance and efficiency, leveraging the strengths of both systems. Additionally, investigating more advanced hardware like ESP32-S3, which offers improved computational capabilities, could enhance the performance of embedded systems for complex real-world applications. Integrating multi-modal sensors or more sophisticated models could further expand the capabilities of embedded systems, enabling new use cases in edge AI.

This study underscores the importance of balancing performance, energy efficiency, and portability when selecting hardware and models for object detection tasks, providing valuable insights for researchers and developers in the field of edge AI, particularly for applications in IoT and distributed computing environments.

VI. REFERENCES