Assistive Pill Counter

Using ESP32-CAM and Edge Impulse

Abstract

The ESP32-CAM–based Assistive Pill Counter is an embedded computer vision system designed to automatically detect and count pill-shaped objects using a low-cost camera module and an on-device machine learning model. The system assists manual medicine filling by detecting and counting pills placed on a tray. It eliminates manual counting errors and provides results through a local web interface without requiring cloud connectivity.

Introduction

Assistive pill counting is a common requirement in healthcare and pharmaceutical workflows. Traditional approaches rely on manual counting or centralized cloud-based vision systems, which introduce latency, privacy risks, and operational cost.

Edge AI enables inference directly on constrained devices, improving responsiveness and ensuring data locality. This project demonstrates a production-style Edge AI pipeline running entirely on an ESP32-CAM, from image acquisition to inference and user interaction.

Problem Statement

Manual pill counting is:

- Error-prone
- Time-consuming
- Difficult to scale

Goal: Design an offline, real-time, low-cost Edge AI solution for pill detection and counting.

Project Objectives

- Deploy an **object detection model** on a microcontroller-class device
- Perform **on-device image preprocessing and inference**
- Design a **modular embedded firmware architecture**
- Apply **Embedded MLOps principles**
- Provide a **browser-based interface** without cloud dependency

System Overview (Edge AI Perspective)

All stages of the AI pipeline run **locally** on the ESP32-CAM:

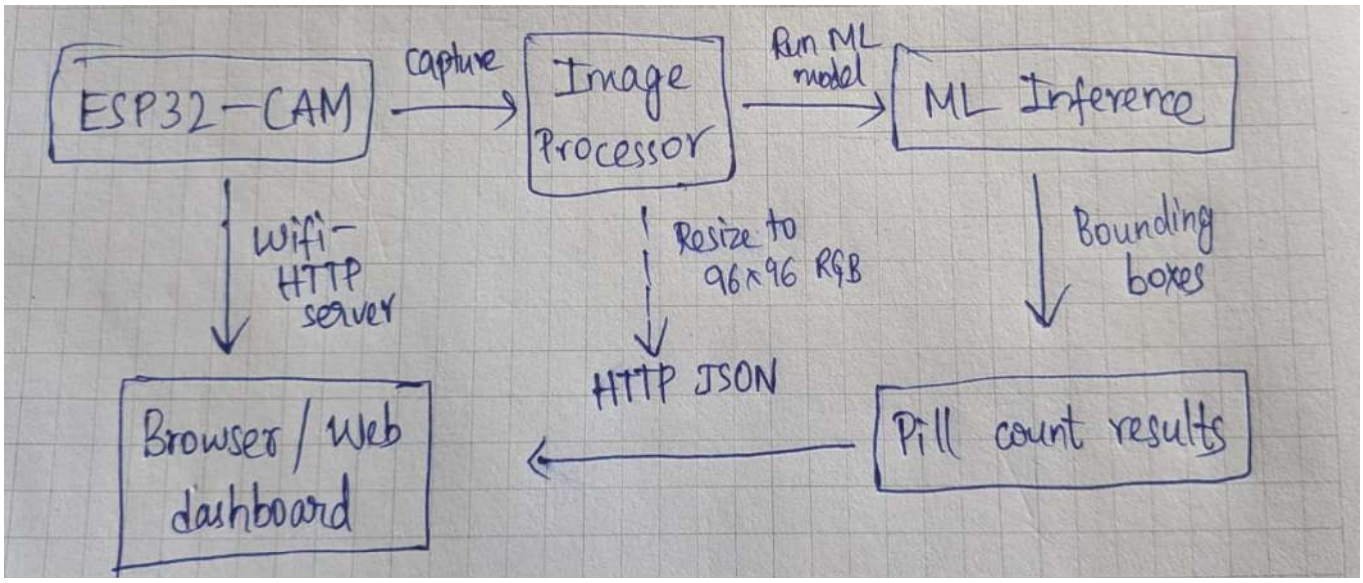
1. Image capture (camera sensor)
2. Image preprocessing (JPEG → RGB → resize)
3. ML inference (TensorFlow Lite Micro)

4. Post-processing (bounding box filtering & counting)
5. Result visualization (local web server)

No images or inference data leave the device.

System Architecture

Architecture :
The system follows a layered embedded architecture, separating hardware access, preprocessing, inference, application logic, and user interaction to improve maintainability and scalability.



- **Hardware Layer :**

  ESP32-CAM microcontroller
  OV2640 camera sensor
  On-chip RAM/PSRAM
  Wi-Fi module (integrated)

This layer is responsible for capturing raw visual data from the environment. The camera captures JPEG frames and stores them in the ESP32 framebuffer (PSRAM). The ESP32 also integrates Wi-Fi for communication and web server support.

- **Edge processing and Inference layer :**

1. Image acquisition : Captures JPEG image via esp_camera_fb_get(),Stores frame in memory
2. Pre processing pipeline : Transforms raw camera output into a format usable by ML model,
   JPEG → RGB conversion using fmt2rgb888(),
   Downsampling to 96×96 resolution,

Channel normalization,
Flattening into input tensor

3. Machine Learning Inference : Uses the **Edge Impulse FOMO object detection model**

Model type: FOMO,
Input: 96×96×3 RGB,
Output: bounding boxes:{ x, y, width, height, confidence, label }

4. Post-processing : After inference, Filters bounding boxes by confidence threshold (> 0.6),Counts number of detected pills

- **Application & Control Layer :**

This layer manages system logic and coordination.

Triggering camera capture on user request
Running ML inference on demand
Managing memory buffers & freeing frames
Preparing JSON responses with results
Streaming JPEG images to browser

- **User Interaction Layer (Web UI) :**

This layer provides user access through a web browser.

UI Features:
"Capture" button
Display of latest captured image
Display of pill count
Live refresh via JavaScript

Technologies Used

- **Hardware**: ESP32-CAM (AI Thinker)
- **Framework**: Arduino (PlatformIO)
- **ML Platform**: Edge Impulse
- **Inference Engine**: TensorFlow Lite Micro
- **Model Type**: FOMO Object Detection
- **Networking**: WiFi + ESPAsyncWebServer
- **Language**: C++

Edge Impulse Workflow

Edge Impulse is used as an **end-to-end Edge ML platform**, not just a training tool:

- Image data ingestion
- Labeling for object detection
- Feature extraction (image pipeline)
- Model training (FOMO)
- Quantization (INT8)
- Deployment with Arduino

The output is a **versioned inference library** integrated directly into firmware.

Model Characteristics

- **Input Size**: 96 × 96 RGB
- **Model Type**: FOMO (Fast Object Detection)
- **Quantization**: INT8
- **Runtime**: TensorFlow Lite Micro
- **Inference Location**: On-device (ESP32)

This configuration balances **accuracy, latency, and memory usage**.

Embedded Image Preprocessing Pipeline

Due to hardware constraints, preprocessing is implemented manually in firmware:

- Capture image in JPEG format
- Convert JPEG → RGB888
- Resize to 96×96 (nearest-neighbor)
- Normalize pixel values
- Feed into inference signal buffer

This pipeline mirrors the training configuration in Edge Impulse.

Inference and Post-Processing

Inference is executed using `run_classifier()` from the Edge Impulse SDK.

Post-processing steps:

- Extract bounding boxes
- Apply confidence threshold
- Count detected pills

All logic executes deterministically on the device.

Web-Based Interaction Layer

A lightweight web server runs directly on the ESP32:

Features

- `/` – Browser-based UI
- `/capture` – Trigger inference
- `/image` – Display captured image

This enables:

- No external applications
- Cross-platform access
- Simple debugging and demonstration

Software Engineering Practices :

- Version-controlled firmware and model artifacts
- Modular separation of camera, preprocessing, inference, and UI
- Deterministic memory management
- Reproducible builds using PlatformIO

This approach aligns with **production embedded systems**, not experimental ML code.

Challenges and Mitigations

Challenges :
- Limited RAM on ESP32
- Image format conversion (JPEG → RGB)
- Real-time inference latency
- Lighting sensitivity
- Model generalization

Solutions Implemented :

- Used PSRAM for image buffers
- Reduced input resolution to 96×96
- Used FOMO for efficient detection
- Threshold tuning
- Quantized model deployment

Limitations and Future Work

- Multi-class pill detection
- Object tracking for robustness
- Improved lighting invariance
- OTA model updates
- Integration with industrial dispensing systems

Conclusion

This project demonstrates the practical feasibility of deploying a complete computer vision pipeline on a resource-constrained edge device using modern Edge AI tooling. By integrating the ESP32-CAM with an Edge Impulse–trained object detection model, the system successfully performs real-time pill detection and counting entirely on-device, without reliance on cloud services.

From an Edge AI and embedded software engineering perspective, the work highlights the importance of system-level design decisions such as input resolution selection, quantized model deployment, memory management using PSRAM, and deterministic preprocessing

pipelines. Rather than treating machine learning as a standalone component, the project integrates inference tightly with firmware architecture, emphasizing robustness, reproducibility, and maintainability.

The adoption of an Embedded MLOps workflow enables controlled model versioning, repeatable deployment, and systematic retraining based on real-world performance. The lightweight web interface further demonstrates how edge intelligence can be exposed to end users in an intuitive and platform-independent manner.

Overall, this project illustrates that Edge AI is fundamentally a software engineering and systems problem, where success depends on the coordinated design of hardware, firmware, inference pipelines, and deployment processes. The resulting system serves as a foundation for future extensions in healthcare automation and industrial embedded AI applications.