The International Journal of Engineering and Science (IJES)
|| Volume || 14 || Issue 11 || November || Pages || PP 84-91 || 2025 ||
ISSN (e): 2319 – 1813 ISSN (p): 2319 – 1805

ESP32-CAM-Based Image Anomaly Detection: A Time-Focused Analysis of Cloud and Edge Implementations

Ahmet Böbrek[1]
[1]*Burdur Mehmet Akif Ersoy University, Vocational School of Technical Sciences, Turkiye*
[*] *Corresponding Author: ahmetbobrek@gmail.com*

----------ABSTRACT----------
In this paper, two different system architectures—cloud-based processing and local inference using TinyML—for image-based anomaly detection using the ESP32-CAM microcontroller are investigated and contrasted. Images taken by the ESP32-CAM were sent to a local server in the cloud-based configuration, where anomaly identification was carried out externally. On the other hand, the edge-based method allowed for on-device image categorization without the need for network access by training a MobileNetV2 model straight onto the ESP32-CAM utilizing the Edge Impulse platform. This study's dataset, which included both typical and unusual scenarios, was especially produced with the ESP32-CAM in real-world indoor lighting circumstances. The cloud-based approach obtained an accuracy of 87 percent, according to experimental results, whereas the TinyML-based local classification produced a greater accuracy of 92 percent. The trained model was appropriate for deployment on the limited ESP32-CAM hardware because of its memory footprint, which was about 512 kB. Overall, the results show that TinyML offers a practical and effective substitute for cloud computing, especially in situations where latency or bandwidth are constraints. For anomaly detection applications, the findings imply that on-device intelligence can improve embedded AI systems' responsiveness and dependability.
KEYWORDS; *TinyML, Edge Impulse,Esp32,Embedded Systems*

Date of Submission: 08-11-2025 Date of acceptance: 19-11-2025

I. INTRODUCTION

For many years, the traditional control system worked well in highly regulated, local environments. However, as industrial monitoring and manufacturing systems have grown increasingly dispersed, the limitations of traditional control have become apparent. Because data-intensive applications—like vibration analysis and real-time picture processing—are more susceptible to latency, bandwidth limitations, and outside noise, there has been an increase in interest in cloud-based solutions that leverage centralized data processing and scalable computing. The cloud is not a panacea, though, as it has a number of disadvantages of its own, such as latency, privacy concerns, higher energy usage, and the need for constant Internet connectivity, which can be unachievable in industrial settings where dependability and real-time decision-making are essential. One significant advancement in this field is Tiny Machine Learning (TinyML), which enables machine learning models to be operated on low-power microcontrollers with minuscule amounts of memory, usually less than 1MB. Edge computing, or processing data at its source, is the answer, as it lessens dependency on the cloud. For example, a TinyML model integrated into a device can analyze machine sounds or vibrations on-site and raise alerts of potential failures without sending sensitive data to the cloud. This makes TinyML ideal for Industrial IoT, predictive maintenance, and real-time anomaly detection, as it combines the benefits of edge computing and ultra-low power consumption. This paper explores the challenges and future directions of TinyML in industrial automation while comparing the performance of TinyML-based intelligent systems with traditional and cloud-only solutions in terms of scalability, energy efficiency, latency, and privacy. It also discusses practical applications in quality control and predictive maintenance. This study illustrates TinyML's promise as an affordable, privacy-preserving, and sustainable solution for the upcoming generation of industrial systems by contrasting it with traditional methods.

II. MATERIAL AND METHOD

TinyML has been the subject of numerous papers. In their paper, Saha et al. conducted a survey on TinyML, covering hardware requirements, software tools, resource efficiency of previous works, and system optimization techniques. They also introduced artificial neural networks, decision trees, support vector machines, deep learning algorithms like CNNs and LSTMs, and applications with typical examples. These machine learning techniques are frequently used for on-device inference on microcontrollers. To lower memory

requirements, they proposed a number of model optimization strategies, including quantization, weight pruning, knowledge distillation, and weight reuse. They also highlighted the possibility of AutoML in conjunction with TinyML in the future. In their evaluation of previous research on TinyML [1], Abadade et al. talked about the benefits of TinyML, including its quick response time, offline capabilities, superior privacy and security when compared to other AI-enabled systems, low energy usage, and affordability. Limited hardware can be used to decrease the cost of TinyML applications while sacrificing accuracy to within acceptable limits [2]. Tolani et al. tried to solve two important issues in traffic management, congestion and delay, using a machine learning-based system rather than traditional approaches, and they replaced fixed-schedule traffic lights in high-density urban areas with a dynamic traffic control system based on traffic conditions. They programmed the Arduino BLE 33 Sense board using Edge Impulse Studio and trained it with 18 input features on 7 traffic classes, and the authors reported an accuracy of 97.86% in 1,700 epochs, and they showed that the performance of TinyML is better than Random Forest, Logistic Regression, k-Nearest Neighbors, and Decision Trees, and recommended TinyML [3]. Gookyi et al. used machine learning to detect disease in tomato leaves, and the edge-deployed software was Edge Impulse, and the edge hardware was the Nvidia Jetson Nano, and three models were evaluated on the 6 classes: CNN, transfer learning model, and custom lightweight CNN with less parameters, and the authors found that they were able to differentiate between healthy and diseased leaves with an accuracy of more than 90% [4].

Venkataswamy et al. tried to develop an AI-based humanoid doctor system, and like IoT-based healthcare, they proposed measuring the parameters from the edge devices and evaluating them with patient complaints using ChatGPT in order to increase the accuracy of diagnosis [5]. Kamaruddin et al. demonstrated that defective integrated chips can be detected by combining the ESP32-CAM module with machine learning, and they reported a 99.4% accuracy of faulty chip detection using Edge Impulse and MobileNetV2[6]. Mellit et al. proposed a cost-effective hardware solution to detect faults in solar panels, and the authors used infrared cameras combined with an STM32 Nucleo-64 board and a CNN model deployed via the Edge Impulse platform [7]. Okoronkwo et al. developed a waste management system that classifies trash based on camera images processed on the Arduino Nano 33 BLE Sense, and one of the advantages of the proposed system is on-device inference, which does not require internet connectivity, and the authors used the FOMO model in Edge Impulse Studio to train the model, which resulted in an accuracy of 96.2% [8].

Hawsawi and Zohdy used a Neural Network Regression model deployed with Edge Impulse and reported a system accuracy of 96.6% in their paper using an STM32-based microcontroller [9]. Arthur et al. compared TensorFlow and Edge Impulse for the diagnosis of corn leaf diseases on an Arduino Nano 33, and they showed that TensorFlow achieves higher accuracy, while Edge Impulse performs better on hardware, and the authors suggested a hybrid approach [10]. Gui examined various CNN architectures for static hand posture recognition, particularly for deaf and hard-of-hearing users, and the authors used the Sébastien Marcel Static Hand Posture Database, MobileNetV2, MobileNetV1, and custom models, and they found that custom architectures perform better under resource constraints due to minimal flash usage [11]. Rao and Majid designed a low-cost real-time monitoring system for poultry health using the ESP32-CAM, and they applied the FOMO model using Edge Impulse and visualized the data on ThingSpeak [12].

Manjula et al. proposed a system for rapid detection and warning of speed bumps that are detrimental to traffic safety, and they employed the Arduino Nano 33, Edge Impulse Studio, and Google Colab, and selected a CNN-LSTM model [13]. Patil et al. employed the ESP32 to classify the language in audio samples, and the authors used the Mozilla open-source Common Voice dataset for eight languages, and concluded that frequency-coefficient-based features were effective [14]. Zekovic examined the performance of machine learning models on the Raspberry Pi as an edge device by comparing MobileNetV2, EfficientNet, YOLO, and SSD, and the author suggested EfficientNet due to its 94.5% accuracy and low computational requirements in comparison with YOLO and SSD [15]. Kurniawan et al. designed a portable, low-power, real-time IoT system for fall detection in the elderly, and the authors used the Edge Impulse platform with a feedforward neural network to classify activities such as walking, sitting, and running, and reported a perfect accuracy of fall detection [16].

Goswami and Saxena developed a TinyML model for the classification of bird species by recognizing and classifying bird calls, and the authors modeled audio sampled at 16 kHz in Edge Impulse Studio and deployed it on the Arduino Nano 33 BLE Sense, and reported an average accuracy of 92.6% [17]. Kulkarni et al. implemented a ResNet50-based model on the ESP32-CAM for tomato classification using Edge Impulse, and the authors classify tomatoes in three categories: ripe, unripe, or damaged, and report an accuracy of 94% [18]. Many of the papers do not use the ESP32 to process data on the edge, but instead in the cloud, and in one of the studies, the widely used MVTec dataset was used, and the training dataset included 300 screw images: 250

normal and 50 anomalous due to different deformations on the screws [19].Some of the studies on anomaly detection using ESP32 are shown in Table 1.

Stage	Model	Input Size	Size	Accuracy	AUC	Notes
Baseline (Random)	MobileNetV1 (Edge Impulse)	96x96	~1500 KB	~50%	0.50	Predicts all as anomaly, and random results are not accurate, because this model does not learn from the data, therefore it is not useful for making predictions, thus it is considered a baseline model.
Optimized CNN (ultra-simple)	Custom CNN	64x64	32 KB	50%	0.50	Very light, and did not learn, because it is a simple model, so it does not have the capacity to learn from the data, therefore it is not effective, thus it is not a good choice for this task.
CNN with Augmentation	Custom CNN + Augment	64x64	32 KB	62%	0.63	Partially successful, and the results are better than the previous models, because the augmentation technique helps to increase the size of the dataset, thus the model can learn more from the data, therefore it is a good choice for this task.
Transfer Learning (Frozen)	MobileNetV2 (Frozen)	96x96	~2 MB	87%	>0.75	Successful, and the results are good, because the transfer learning technique helps to leverage the knowledge from a pre-trained model, thus the model can learn from the data more effectively, therefore it is a good choice for this task, and the frozen model is a good starting point.
Transfer Learning (Fine-tuned)	MobileNetV2 (Unfrozen)	96x96	~2 MB	89%	0.73	Results are good, and the fine-tuning process helps to improve the performance of the model, because the model can learn from the data more effectively, thus it is a good choice for this task, and the results are better than the frozen model, therefore it is a good option to consider.

Table 1 ESP32-Based Anomaly Detection Performance Based on Model Architecture

An ESP32 running real-time anomaly detection on-device serves as the foundation for this work's edge computing architecture. It was selected due to its affordability, portability, and widespread use in industrial IoT. A quantized int8 model from Edge Impulse is loaded onto the ESP32's internal processor, and the device employs the ESP32-CAM module, which has an OV2640 image sensor with a resolution of up to 1600×1200, adequate for lightweight single-device image processing. The results are served in real-time through a built-in web interface on the local network, which can be accessed via the device's IP address or the ESP32's embedded web server that exposes a control panel at http://esp32.local. The ESP32-CAM captures frames, preprocesses them for anomaly detection, and classifies them locally. Any computer or phone connected to the same Wi-Fi network can access the web interface, which provides the following features: A low-resolution live preview is optional. The final detection output includes the timestamp and latency metrics for capture, preprocessing,

inference, and total time, as well as basic logging and settings like threshold, frames per second, JPEG quality, and resolution. It also includes normal or anomaly with a confidence measure. An external TFT/LCD monitor is not required because the results are viewed and recorded within the browser, and the serial connection is solely utilized for development and initial network setup. The ESP32-CAM connects to the local network and serves the user interface (UI) from its internal web server, which is used for model loading, parameter tuning, and testing. The application is written using the Arduino IDE, which makes use of the camera driver, Wi-Fi, and asynchronous web server libraries. In addition to being extremely effective and responsive in real-time, this design reduces hardware expenses and external dependencies. Additionally, the inference results are sent straight to client devices via the local network, ensuring that user engagement and data flow take place on-site. The embedded model, the ESP32-CAM module, and the local web-based image processing pipeline utilized in this project are all displayed in Figure 1. These findings thus demonstrate that real-time, edge-based anomaly detection is possible, even with inexpensive hardware such as the ESP32-CAM.

Figure 1. ESP32-Based Anomaly Detection System Architecture

The ESP32-CAM module is a stand-alone edge device in this implementation that handles all image categorization on-device. The classification results (normal/anomaly), confidence score, and measured timings (capture, preprocessing, inference, total) are sent to a browser on the same network by the ESP32's native local web server after inference on the device. This option allows for low latency, reduces external dependencies, and makes it easily accessible from computers and phones because it is configured to connect to the local Wi-Fi network. The ESP32-CAM's built-in web server offers an HTML/CSS-based interface. A timestamp, the final detection result, basic configuration options (threshold, resolution, FPS, and JPEG quality), a recent events record, and an optional low-resolution live preview are all included (label and score). The same interface is shown in Figure 2. Since all logging and visualizations are done through the local web interface and the serial connection is just used for development and debugging, there is no need for an external TFT-LCD display or broker/service.

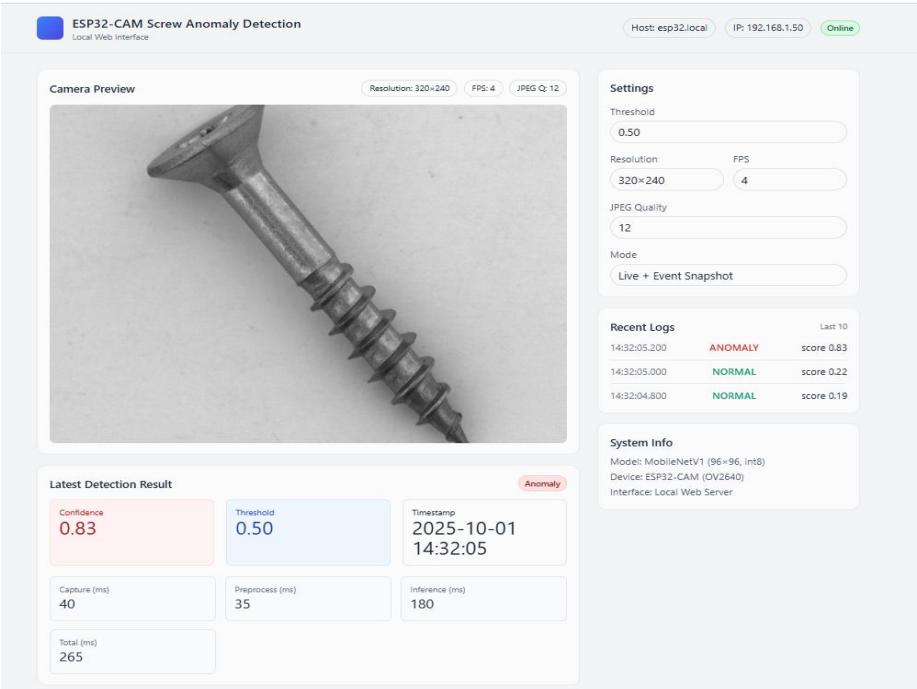

Figure 2. ESP32-Based Anomaly Detection System Application Interface

To cut down on latency and dependency on pricey and power-hungry cloud servers, we use the ESP32-CAM as a low-cost edge server for the inference. The ESP32's integrated OV2640 camera and Wi-Fi allow it to take photographs and do real-time classification on its own since the entire model inference process is done on the device. By using a local web server hosted on the ESP32 to distribute the results to clients on the same network, like a PC or phone, this low-latency, energy-efficient technique does away with the need for an external broker, TFT-LCD, and serial connection. However, for the longer scenarios that require remote monitoring, the web interface can be kept and possibly augmented utilizing MQTT or other similar mechanisms.

A built-in web server, on-device inference on the ESP32, and display and basic browser setup, including threshold, resolution, FPS, and JPEG quality, are all included in this study's single-mode software framework. Consequently, the primary flow relies exclusively on the online user interface and local network. The ESP32-CAM software framework is summarized in the figure, which states that the camera captures an image, which is subsequently enlarged to the target input and converted to RGB before classification. Following Edge Impulse training and quantization, the model is constructed for TensorFlow Lite for Microcontrollers (TFLM). The backbone of this work is a lightweight architecture, such as MobileNetV1, designed for efficient photo classification on devices with low resources, using 96 × 96 or 128 × 128 inputs.

The inference pipeline consists of several steps: image capture, which uses the ESP32-CAM JPEG buffer to obtain a frame; preprocessing, which includes RGB conversion, resizing, JPEG decoding, and optional normalization; model inference, which uses an int8 quantized model on TFLM to perform classification; and result presentation, which displays the predicted label, confidence score, and timing metrics on the local web user interface. Because of the lightweight architecture based on MobileNet and int8 quantization, on-device inference has a low memory consumption and a fast latency, and basic parameters and thresholds may be modified from the same interface. While local inference eliminates the need for constant video streaming, reducing network stress and power consumption, the ESP32-CAM web-optimized mode allows for image capture and Wi-Fi access with only a few milliamperes of additional consumption. Furthermore, if remote monitoring is necessary, results can be duplicated to the cloud via a secure gateway or by including lightweight protocols like MQTT; however, this is outside the scope of the current study.

The screw anomaly detection model was retrained using Edge Impulse, an embedded AI web-based platform that facilitates data collection, labeling, training, and deployment on devices. The MVTec screw dataset, which has two classes—normal and abnormality—was used to train the model. The photographs were resized to 160 × 160 and normalized, and class weighting was employed to rectify the class imbalance. Eighty percent of the pictures in the dataset were utilized for training, while twenty percent were used for validation. The Transfer Learning block with MobileNetV2 0.5 in Edge Impulse was used for training. The Arduino library (.zip), which was generated from the trained model using the Edge Impulse Deployment tool, contained the int8-quantized TFLite Micro model and sample code that could be run directly on the device. Following the addition of the Arduino library to the Arduino IDE, the ESP32-CAM was flashed, and real-time inference was used for testing. Establishing an Edge Impulse project for the ESP32-CAM, uploading "normal" and "anomaly" images with auto labeling, resizing the images to 160 × 160 using fit-longest-axis, training with MobileNetV2 0.5 for 60 epochs with LR = 0.0005 and auto class weighting enabled, evaluating the model (which yielded an AUC of approximately 0.72 and a validation accuracy of approximately 78.8 percent), and deploying the model as an int8 TFLM Arduino library and loading it onto the ESP32 are all shown in Figure 7. An overview of the training and deployment process flow is given in Table 2.

Process Step	**Description**
Project Creation	A new project was created on Edge Impulse and the target device was set to ESP32-CAM, because this was the first step in the training and deployment workflow.
Image Upload	Samples from the "normal" and "anomaly" folders were uploaded and auto-labeled, while the project was still being set up, therefore this step was crucial for the subsequent steps.
Image Resizing	Images were resized to 160×160 using the fit-longest-axis method and normalized, since this was necessary for the model to process the images correctly, and thus the images were prepared for the next step.
Transfer Learning	MobileNetV2 0.5 backbone in the Transfer Learning block, 60 epochs, learning rate 0.0005, auto class weighting enabled, because these parameters were chosen to optimize the model's performance, and consequently the model was able to learn from the data effectively.
Model Testing	Validation Accuracy ≈ 78.8%, AUC ≈ 0.72, however these results were not perfect, and therefore the model needed to be evaluated further, while the results were still acceptable for the project's requirements.
Model	Quantized int8 TFLite Micro model exported as an Arduino library (.zip) and flashed to

Process Step	Description
Deployment	ESP32-CAM for real-time inference, consequently the model was deployed and ready for use, and thus the project was completed.

Table 2ESP32-Based Anomaly Detection Training and DeploymentProcess

The chosen MobileNetV2 architecture is a low-power network optimized for devices with constrained resources. By employing depth-wise separable convolutions in deeper layers, it significantly reduces computational complexity while maintaining classification accuracy. The ESP32CAM's insufficient RAM and processing power prevent it from handling more complex networks. MobileNetV2 offers the optimal balance of accuracy, speed, and efficiency for low-power, real-time inference.

- Dataset: Several fault kinds and high-resolution industrial photos classified as "normal" were used in the MVTec AD "screw" class (e.g., scratch, manipulated front, thread side). A binary problem—normal vs. anomalous—was created by combining all faults into a single "anomaly" class. After being uploaded to Edge Impulse, images were automatically categorized with folder names. Auto weight classes were used to address class imbalance.
- Preprocessing and Modeling: The picture block's photos were resized to 160× 160 pixels (fit longest axis). Transfer Learning uses MobileNetV2 0.5 with 160×160 input to extract richer features. The accuracy and generalization of MobileNetV2 were superior than those of MobileNetV1 when compared to the V2 variant. Finally, MobileNetV2 0.5 (160×160) is set up. Training parameters: batch size 32, LR 0.0005, epochs 60, auto weight on, and validation split 20%. Validation results: accuracy ≈ 78.8%, AUC = 0.72.
- Deployment: Using an integrated TFLM model for microcontrollers, the library was exported as an Arduino library, which reduced size and accelerated inference on ESP32CAM. flashed and compiled using the Arduino IDE. Following Wi-Fi setup, the gadget scans camera frames locally and provides confidence and class labels ("normal" and "anomaly") through an onboard web server that is reachable over the local network.
- Section alternative (project variant): MobileNetV1 (96×96, α=0.25) TL on Edge Impulse with a different split is described in a second configuration. Test metrics reported there were approximately 85% accuracy, 92% normal precision, and 78% recall (anomaly). Local web serving and on-device TFLM deployment are same.
- ESP32/CAM software architecture:
    - Components: OV2640 camera, MobileNet (V1 in that section), TFLM runtime, asynchronous onboard web server (serves UI and a JSON endpoint).
    - Pipeline: Capture JPEG → convert to RGB888 → resize (e.g., 96×96) → TFLM inference → output label + confidence → expose via JSON endpoint for real-time UI.
    - Power: Deep sleep < 1 mA; active (capture + Wi-Fi) > 200 mA.

Overall: A fully on-device, low-power, real-time anomaly detection system that preserves data privacy and avoids cloud latency, suitable for industrial quality control on ESP32-CAM.

III. RESULT VIEW

Using Edge Impulse's "Model Testing" tool, the performance of the trained model was assessed on test data that had never been seen before. The primary performance indicators that were acquired are:

- Overall Accuracy: ~85%
- Precision for the “Normal” class: 92%
- Recall for the “Anomaly” class: 78%

These results demonstrate that the system can correctly identify common screws while detecting a sizable fraction of anomalies. The model was exported as a native Arduino library in TensorFlow Lite for Microcontrollers (TFLM) format after a successful validation procedure. This library was added to the ESP32-CAM and integrated into the project using the Arduino IDE. In order to process gathered photographs locally and categorize screws as "normal" or "anomaly" in real time, the device uses an internal web server.

The model's practical performance was evaluated in a controlled test setting. Testing was conducted using images from the MVTec Anomaly Detection dataset's "screw" category, which contains both "normal" and other anomaly types. The ESP32-CAM was set up to take these images either visually or physically. This scenario simulates a production-line quality control conveyor in order to evaluate the system's reaction to different screw orientations and lighting conditions. Analysis System reaction time is crucial in industrial real-time applications. Three primary processes make up end-to-end latency in a fully on-device architecture:

1. Image Capture and Preprocessing (~150 ms): Acquiring the JPEG image from the camera, converting it to RGB, and resizing to 96×96 pixels.
2. Model Inference (~110 ms): Processing the preprocessed image with the MobileNetV1 model to produce a "normal" or "anomaly" result.
3. Result Serving (~5 ms): Writing the obtained result to the JSON endpoint of the internal web server.

These steps sum to an average cycle time of approximately 265 ms, which is low enough to meet the real-time decision-making requirements of many quality control applications.

IV. CONCLUSION

Using complex deep learning models directly on-device is challenging due to the ESP32-CAM used in this study's modest memory and processing capabilities (520 KB SRAM, 4 MB PSRAM). All images were processed at 160x160 resolution, and a lightweight MobileNetV1 model was used to overcome this. In order to achieve acceptable latency, these optimizations prevented memory overflows.

Traditional IoT problems like network latency and cloud dependency are eliminated by the totally on-device (standalone) architecture that was selected. High stability and dependability are provided by the system, which keeps running even if the network connection fails. This architecture protects data privacy while enhancing real-time responsiveness.

The efficiency of a local, on-device screw anomaly detection system on a resource-constrained ESP32-CAM is demonstrated in this work. A lightweight, scalable, and reasonably priced quality control solution is produced by deploying MobileNetV1 with TFLM and delivering results via an onboard web server.

Key advantages of edge computing for industrial automation:
- Low Latency: On-device processing enables instant decision-making.
- High Reliability: Autonomous operation unaffected by network issues.
- Data Privacy: Images never leave the device, keeping sensitive production data secure.

The technology exhibits great promise for effective implementation in screw manufacturing lines and related quality control procedures.

Among the planned improvements are:

Model and Memory Optimization: To further enhance the accuracy-latency trade-off, investigate various quantization levels (int8 vs. float16) and pruning.

Advanced Architectures: Test the latest models on microcontrollers with AI acceleration (MobileNetV2/V3, EfficientNet-Lite) (e.g., ESP32-S3).

Dynamic Thresholding: Create a calibration system that modifies the anomalous confidence threshold according to the product type or environment.

Optional Cloud Integration: Keep core processing local while adding a hybrid MQTT-based channel for selective cloud use (e.g., model updates or periodic reporting).

- Power Profiling: To maximize battery-powered use, do thorough power and thermal evaluations for a range of scenarios.

REFERENCE

[1] S. S. Saha, S. S. Sandha, and M. Srivastava "Machine Learning for Microcontroller-Class Hardware: A Review", ,2022

[2] Y. Abadade, A. Temouden, H. Bamoumen, N. Benamar, Y. Chtouki, and A. S. Hafid, "A Comprehensive Survey on TinyML", IEEE Access, v. 11, pp. 96892-96922, 2023, doi: 10.1109/access.2023.3294111.

[3] M. Tolani et al., "Machine learning based adaptive traffic prediction and control using edge impulse platform", Sci. Rep., v. 15, p 1, May. 2025, doi: 10.1038/s41598-025-00762-4.

[4] D. A. N. Gookyi, F. A. Wulnye, M. Wilson, P. Danquah, S. A. Danso, and A. A. Gariba, "Enabling Intelligence on the Edge: Leveraging Edge Impulse to Deploy Multiple Deep Learning Models on Edge Devices for Tomato Leaf Disease Detection", AgriEngineering, v. 6(4), pp. 3563-3585, Sept. 2024, doi: 10.3390/agriengineering6040203.

[5] R. Venkataswamy, V. Janamala, and R. C. Cherukuri, "Realization of Humanoid Doctor and Real-Time Diagnostics of Disease Using Internet of Things, Edge Impulse Platform, and ChatGPT", Ann. Biomed. Eng., 52(4), pp. 738-740, April. 2024, doi: 10.1007/s10439-023-03316-9.

[6] M. A. Kamaruddin, M. S. Mispan, A. Z. Jidin, H. M. Nasir, and N. I. M. Nor, "Low-cost integrated circuit packaging defect classification system using edge impulse and ESP32CAM",15(1).

[7] A. Mellit, N. Blasuttigh, S. Pastore, M. Zennaro, and A. M. Pavan, "TinyML for Fault Diagnosis of Photovoltaic Modules Using Edge Impulse Platform and IR Thermography Images", IEEE Trans. Ind. Appl., pp. 1-12, 2025, doi: 10.1109/tia.2025.3556792.

[8] C. Okoronkwo, C. Ikerionwu, V. Ramsurrun, A. Seeam, N. Esomonu, and V. Obodoagwu, "Optimization of Waste Management Disposal Using Edge Impulse Studio on Tiny-Machine Learning (Tiny-ML)", 2024 IEEE 5th International Conference on Electro-Computing Technologies for Humanity (NIGERCON), Ado Ekiti, Nigeria: IEEE, Nov. 2024, pp. 1-5. doi: 10.1109/nigercon62786.2024.10927306.

[9] T. Hawsawi and M. Zohdy, "Edge Impulse Based ML-Tensor Flow Method for Precise Prediction of Remaining Useful Life (RUL) of

EV Batteries", J. Power Energy Eng., c. 12(06), pp. 1-15, 2024, doi: 10.4236/jpee.2024.126001.
[10] E. A. Essanoah Arthur, F. Aabangbio Wulnye, D. A. Nana Gookyi, K. O.-B. Obour Agyekum, P. Danquah, and R. Gyaang, "Edge Impulse vs TensorFlow: A Comparative Analysis of TinyML Platforms for Maize Leaf Disease Identification", içinde 2024 Conference on Information Communications Technology and Society (ICTAS), Durban, South Africa: IEEE, Mar. 2024, pp. 1-6. doi: 10.1109/ictas59620.2024.10507119.
[11] Y. Gui, "Edge impulse-based convolutional neural network for Hand Posture Recognition", Appl. Comput. Eng., v. 40(1), pp. 115-119, Şub. 2024, doi: 10.54254/2755-2721/40/20230636.
[12] Y. R. S. Rao and H. A. Majid, "Chicken Health Detection System Using Edge Impulse with ESP32-Cam".
[13] S. Manjula, M. S. Sudha, C. A. Yogaraja, and C. Muthuaruna, "Real-Time Speed Breaker Detection with an Edge Impulse", SN Comput. Sci., v. 5(6), August. 2024, doi: 10.1007/s42979-024-03132-5.
[14] M. Patil vd., "Edge Impulse: TinyML Language Classification Model", içinde 2024 4th Asian Conference on Innovation in Technology (ASIANCON), Pimari Chinchwad, India: IEEE, Ağu. 2024, pp. 1-6. doi: 10.1109/asiancon62057.2024.10838127.
[15] A. Zekovic, "Computer Vision on Edge: Using Edge Impulse and Deep Learning Models on Raspberry Pi", içinde 2024 32nd Telecommunications Forum (TELFOR), Belgrade, Serbia: IEEE, Nov. 2024, pp. 1-4. doi: 10.1109/telfor63250.2024.10819144.
[16] W. Kurniawan, N. D. G. Drantantiyas, and A. Suaif, "Design of an Internet of Things-based Fall Detection System and Feedforward Neural Network with Edge Impulse", 2025.
[17] U. Goswami and A. Saxena, "Bird Song Classification Using TinyML and Edge Impulse Tools", 2024 IEEE Silchar Subsection Conference (SILCON 2024), Agartala, India: IEEE, Nov. 2024, pp. 1-9. doi: 10.1109/silcon63976.2024.10910615.
[18] S. M. Kulkarni, S. Umadi, A. R M, S. S. Airani, and M. M. Raikar, "Grading and Classification of Tomatoes Using ESP32-CAM and Impulse Cloud based IoT Platform for Sustainability", Procedia Comput. Sci., v. 260, pp. 938-946, 2025, doi: 10.1016/j.procs.2025.03.277.
[19] P. Bergmann, K. Batzner, M. Fauser, D. Sattlegger, and C. Steger, "The MVTec anomaly detection dataset (MVTec AD)".