



TINYML-DRIVEN SMART REAL-TIME MULTI-CLASS OBJECT DETECTION ON RESOURCE-CONSTRAINED STANDALONE DEVICE

Satishkumar Kataria¹, Pankaj Kumar Prajapati²

¹Research Scholar, Gujarat Technological University, Chandkheda, Ahmedabad, Gujarat, India

²Associate Professor, Vishwakarma Government Engineering College, Gujarat, India

¹<https://orcid.org/0009-0007-4505-2487>, ²<https://orcid.org/0000-0001-6731-4264>

E-mail: smkataria@gpahmedabad.ac.in, pankaj@ldce.ac.in

ARTICLE INFO

Article History

Received on: April 05, 2026

Reviewed on: May 04, 2026

Accepted on: June 20, 2026

Published on: September 30, 2026

Keywords:

TinyML,
Edge-AI,
Object Detection,
Resource-Constrained Devices,
Standalone Intelligence.

ABSTRACT

Currently, the world is undergoing an AI and ML revolution, and we have seen tremendous growth in AI implementation across sectors over the last decade. Conventional AI-based systems were implemented in a cloud-centric environment, using servers with high processing power, ample storage, and high-speed internet, which consume significant power. This problem has driven the advancement of Edge-AI and TinyML, which have facilitated the implementation of intelligent visual systems on resource-limited embedded devices, enabling standalone device operation. In this paper, a standalone ESP32-CAM device is used to construct a real-time multi-class object detection system based on TinyML. For effective on-device inference, the suggested research strategy focuses on developing and using a quantized deep learning model based on the Edge Impulse architecture. To increase resilience across a range of environmental conditions, a bespoke dataset comprising four classes was produced and pre-processed. Using this method, we have produced an optimized machine learning model that is about 263 KB in size, with an inference latency of 133 ms and real-time performance of 7.5 FPS on the ESP32-CAM board. Under controlled circumstances, the results show an F1-score of 98.02%. In this approach, the end device itself runs an ML model and operates as a standalone device, offering strong data privacy, reduced communication, and very low latency. That's why it balances accuracy, processing power, and real-time operation, making it best suited for small embedded devices.



Copyright ©2026 by authors and Galileo Institute of Technology and Education of the Amazon (ITEGAM). This work is licensed under the Creative Commons Attribution International License (CC BY 4.0).

I. INTRODUCTION

In recent years, we observed tremendous growth in artificial intelligence (AI), along with computer vision applications becoming smart and enabling a variety of smart applications based on object detection, classification, video analysis, etc [1]. Old AI is based on cloud-centric architecture, and this approach has a number of problems that include high bandwidth consumption, high latency, and privacy problems of data [2]. The solution to this problem is suggested by Edge AI and Tiny Machine Learning (TinyML) through the use of on-the-field intelligence to handle the data, which addresses the drawbacks of conventional AI [3]. One of the enablers of Edge AI applications is TinyML. It seeks to implement light-weight ML models on resource-constrained embedded devices without efficiency loss to allow devices to execute tasks locally without an always-on internet connection or offloading to a more powerful processor [4]. Industry 4.0, Smart Agriculture, Smart Surveillance, Smart Healthcare, Environment Monitoring, Smart City, and Smart Healthcare are some of the fields where TinyML is convenient to be used due to its small memory footprint, functionality as a standalone, and low latency [5]. For tiny embedded vision applications, the ESP32-CAM is most suitable because of its power efficiency, built in OV2640 camera and Wi-Fi [6]. Many researchers have worked on this area across different applications, but deploying multi-class object detection with limited hardware resources remains very difficult in this research approach, due to limitations on processing power and memory footprint [7]. Also, multi-class detection performance in independent edge conditions has not been thoroughly assessed, and most experiments have been done on simpler classification tasks or finding single objects [8].

Moreover, a trade-off between model complexity and real-time execution on very low-power hardware platforms, like the ESP32-CAM, is not a well-studied topic [9]. One more significant form of gap is the absence of completely autonomous systems of edge vision that are able to operate without cloud connectivity and have a high quality of detection. Most available systems do not provide sufficient performance to meet hardware constraints, or delegate computation to edge servers [3],[7]. There is a need, therefore, to have a robust, efficient TinyML-enabled solution that delivers real-time and high-accuracy multi-class object recognition at standalone embedded platforms. This paper suggests a Next-Generation Edge AI platform with intelligent, real-time multi-class object recognition with TinyML on the ESP32-CAM platform to address these challenges. The proposed approach develops and optimizes a lightweight deep learning model that is specifically optimized to be used as an embedded system based on the Edge Impulse architecture [10]. The model is trained and evaluated in both controlled and real-world settings using a custom dataset comprising four object classes. The system's ability to function as a standalone unit ensures low latency, minimum power consumption, and data privacy. The main contributions of this work are as follows: The creation of a multi-class object detection model based on TinyML that is optimized for ESP32-CAM; the design of a completely independent real-time edge vision system that is not dependent on the cloud; the achievement of high detection performance with nearly perfect F1-score under controlled conditions; and a comprehensive assessment of latency, memory usage, and real-time inference capability on hardware with limited resources.

II. RELATED WORK

II.1 RELATED WORK

Paul et al. used TensorFlow Lite Micro to build TinyML-based image processing on the ESP32 natively. With real-time inference capabilities, the system achieves a low memory footprint (250–500 KB). According to experimental comparison, embedded deployment is doable, but performance is much lower than that of desktop computers. The method demonstrates how TinyML can be applied practically to embedded vision applications. Nevertheless, it is constrained by limited processing power and low FPS (1-3) [11]. A TinyML-based ESP32-CAM agricultural monitoring system was proposed by Donskoy et al. The system uses little electricity and achieves 90–95% classification accuracy. It shows how machine learning may be effectively used in practical farming applications. The system is designed to operate energy-efficiently in settings with limited resources. Nevertheless, it limits multi-object capacity because it concentrates on classification rather than object detection [12]. Prado et al. developed an ESP32-CAM-based car plate recognition system with LoRa communication. With an average latency of about 3.2 seconds, the system achieves 92.4% recognition accuracy. It also enables real time image capture and wireless transmission to facilitate intelligent surveillance applications. The system is deployable, it consumes less energy and can be done remotely. However, this has a high latency [13], which limits its application in real-time edge AI applications.

Chang et al. developed an ESP32 board-based edge computing system which is a combination of cloud assisted processing and object detection. With embedded limitations the system can generate 1-5 FPS with a latency of approximately 200-400 ms. It demonstrates the way that a hybrid edge-cloud architecture can enhance performance. Nevertheless, the use of cloud connectivity creates latency and overhead in communication. It limits its application to an entirely independent real-time systems [14]. Mittal (2024) has conducted a comprehensive discussion of lightweight object identification models of edge devices. According to the study, most of the TinyML systems achieve an accuracy of 85-95 on trade-offs between latency and model size. It points out the problems of real-time performance and multi-class detection. The report states that limited hardware needs effective architectures. Additionally, it highlights shortcomings in standardized benchmarking [2]. The need for more effective edge intelligence frameworks is highlighted by the fact that, despite tremendous advancements in ESP32 based TinyML systems, current methods are still unable to achieve high accuracy, low latency, and fully standalone multi-class object identification.

II.2 RESEARCH GAPS AND MOTIVATION

The analysis presented above identifies the following research gaps: Limited ESP32-CAM real-time multi-class object identification implementation, TinyML models' trade-off between computational efficiency and accuracy, Absence of completely independent edge vision systems that do not rely on the cloud, Real-time performance indicators (latency, FPS, memory, and F1-score) are not adequately evaluated, and classification activities are prioritized over detection tasks. This paper proposes a NextGen TinyML-based standalone edge AI system that performs real-time multi-class object recognition on an ESP32-CAM with high accuracy and optimal performance to overcome these issues.

III. METHODS AND MATERIALS

III.1 SYSTEM DESIGN FLOW

Four processes make up the System Design Flow for TinyML implementation, as illustrated in Fig. 1: (I) Data collection and pre-processing; (II) ML model design and training; (III) Testing the model accuracy and model quantization; and (IV) Install the model on an embedded device and test it in real time.

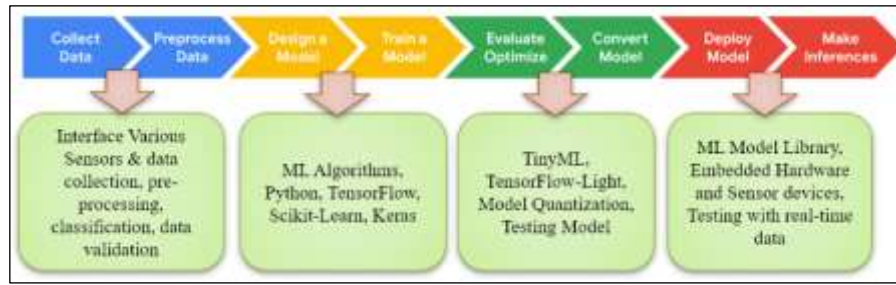


Figure 1: TinyML System Design Flow (Slightly modified version then in. Source: [15].

III.1.1 Data collection and Pre-Processing

This step is divided into two phases: data collection and preprocessing. During the data collection phase, various sensors capture raw signals, images, and audio from the real world. The sampling rate, resolution, duration per sample, device placement, and environmental conditions are all specified [15] [16]. Raw sensor streams are transformed into model-ready inputs with consistent size and distribution during the preprocessing stage; this process should be considered a repeatable pipeline. Cleaning and denoising, segmentation, windowing, resampling, down-sampling [16] [17], normalization, and scaling [17] [18] are the typical procedures carried out. The TinyML model may perform the same pipeline on-device after learning helpful patterns. This stage is particularly crucial in TinyML, as the dataset and preprocessing pipeline must be developed jointly to address sensor noise, memory, computational, and real-time constraints.

III.1.2 ML Model Design and Train The Model

In this phase, we must describe the training setup after choosing (or developing) an ML architecture that satisfies tiny-device restrictions (Flash, RAM, latency) [15] [19].

III.1.3 Testing The Model Accuracy and Model Quantization

Refers to assessing your TinyML model's performance both before and after optimization using both common machine learning metrics and embedded-system metrics (latency, memory, and energy), then utilizing quantization to reduce the model's size for microcontroller deployment while calculating the accuracy trade-off [15] [20]. Using a held-out test set that mimics real-world situations (e.g., various users, noise levels, devices), accuracy testing confirms that the model generalizes to unseen data and satisfies deployment requirements. Accuracy assessment in TinyML articles incorporates on-device realism in addition to ML metrics [21]. Model quantization: To minimize model size, latency, and energy while fitting MCU restrictions, quantization lowers the precision of model weights and activations (e.g., from 32-bit float to 8-bit integer), however accuracy may suffer. In order to demonstrate the trade-off, the accuracy must consistently be posted pre- and post-quantization [22].

III.1.4 Implement Model in Embedded Device and Real Time Testing

The process of deploying the model in embedded devices and real-time testing of these environments (latency, accuracy, power, stability) is parts converting the trained/quantized model into a microcontroller-compatible firmware, flashing it to the target device, and then testing its performance in the real world. This last stage ensures the model's reliability in production, where performance may be impacted by variables including timing jitter, sensor noise, and battery exhaustion [23]. Packing the quantized model (.tflite) into firmware that continuously performs inference in response to timers, interrupts, or sensor events is known as deployment. The procedure uses MCU-optimized embedded machine learning runtimes (no operating system is required) [24].

III.2 SOFTWARE TOOLS

Table 1: TinyML Software Development Tools.

Tool Name	Tool Description
TensorFlow Lite Micro (TFLite Micro)	It is a lightweight, effective, and open-source deep learning framework created by Google that uses only a few kilobytes of memory to implement ML models on edge devices with limited resources, like microcontrollers (MCUs) and digital signal processors (DSPs) [25].
Edge Impulse Studio	A cloud-based end-to-end platform for creating TinyML models is called Edge Impulse. Data collection, labelling, training, optimization, deployment, and testing are all supported in unison. Inference models can be exported as SDKs or libraries to be integrated into edge applications and embedded firmware [26].
LiteRT for MCUs	A C++-based runtime library called LiteRT was created to implement machine learning models on MCU-class devices. In order to enable effective on-device inference, it is tailored for 32-bit ARM Cortex-M processors and supports a variety of platforms, including ESP32, Arduino Nano 33 BLE Sense, STM32, ESP-EYE, and others [27].

STM32Cube.AI	STM32Cube.AI created AI as a tool for STM32 microcontrollers. It offers hardware acceleration through the use of the ST Neural ART NPU and permits the development, optimization, and deployment of AI models into embedded firmware [28].
CMSIS-NN	ARM created the CMSIS-NN library, which offers highly efficient neural network kernels for Cortex-M microcontrollers. When compared to conventional implementations, it greatly increases inference speed by enabling the effective execution of quantized (int8) neural network layers [29].
μTVM	TVM is extended for microcontrollers by μTVM, a machine learning compilation framework. With host-driven execution, model compilation, optimization, and deployment are facilitated. It enables the use of tools like AutoTVM for real-world latency measurement and automated performance optimization [30], [31].
NVIDIA TensorRT	TensorRT is a high-performance SDK for deep learning inference on NVIDIA GPUs. To achieve faster, more effective inference during deployment, it optimizes training models by reorganizing computation graphs and using efficient GPU kernels [32].
ONNX Runtime Mobile	On mobile platforms like Android, iOS, and .NET environments, ONNX Runtime Mobile allows machine learning models to be inferred on-device. For the purpose of running efficient models in limited mobile applications, it offers a lightweight runtime [33].

Source: Authors, (2026).

III.3 HARDWARE TOOLS

Various providers offer numerous platforms. These are a few interesting platforms that have been examined along with their features, connectivity choices, and available sensors. Arduino Nano 33 BLE Sense [34], Apollo3 [35], STM32F Discovery [36], ESP32 [37], ESP-EYE [38], Sony's Spresense [39], GAP8 [40], OpenMV Cam H7 Plus [41], Nordic Semi nRF52840 DK [42], CC1352P Launchpad [43], Pico4ML BLE [44], and Raspberry Pi 5 [45].

IV. SYSTEM IMPLEMENTATION (ESP32-CAM DEPLOYMENT)

Real-time multi-class object detection on a standalone ESP32-CAM device is made possible by the suggested Next-Generation Edge AI-based TinyML system. On an embedded platform with limited resources, the system unifies data collecting, preprocessing, model building, optimization, and deployment via an organized pipeline. The architecture prioritizes low latency, little memory use, and independent operation without reliance on the cloud, all essential components of edge intelligence systems [46], [47].

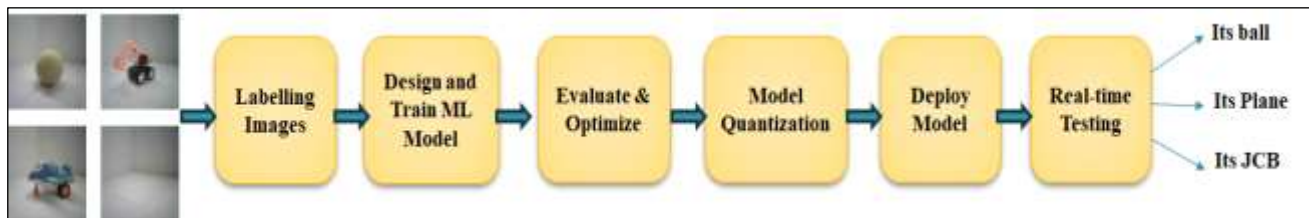


Figure 2: Proposed System Implementation Steps.

Source: Authors, (2026).

IV.1 DATA ACQUISITION

The system begins with the collection of a custom dataset (4 types of objects). The ESP32-CAM and other imaging tools are applied to take pictures in a variety of environmental conditions including variations in backdrop complexity, object orientation, and lighting. It is necessary because this diversity of data collection enhances the robustness and generalization of the model to practical use [7]. That is followed by the Edge Impulse platform, that offers an integrated platform to TinyML model building.

IV.2 DATA ANNOTATION AND PREPROCESSING

The object positions of the detection task of the object are defined as bounding boxes in the generated dataset. In the object detection system, there should be proper labelling where it involves supervised learning. The preprocessing tasks include: Data augmentation (rotation, flipping and scaling); pixel normalization to ensure stable training; and resizing of images to fit in the model input sizes. They are preprocessing techniques, specifically on short data, that enhance the robustness of models and decrease overfitting [48].

IV.3 FEATURE EXTRACTION

Features are extracted by using signal processing blocks that are optimized by the Edge Impulse pipeline. At this step, raw picture data is changed into a significant, abridged feature representations that can be effectively embedded within inference. In the case of TinyML systems, an efficient feature extraction is needed since it results in a low computational cost with a high number of visual features [49]. The lightweight feature extraction techniques are used to ensure compatibility with the limited processing capabilities of microcontrollers such as the ESP32-CAM.

IV.4 MODEL TRAINING

A deep learning model is trained on the labelled dataset, which is a simple lightweight CNN model that can be used to run TinyML. The model is constructed, trained and validated with the help of Edge Impulse framework to implement the training procedure. In order to keep track of the model performance during training, the dataset is divided into the training and validation set. Parameters such as the batch size, learning rate and the number of epochs are adjusted to achieve high detection accuracy without overfitting [50].

IV.5 MODEL OPTIMIZATION

The trained model undergoes several optimization options to enable it to be deployed on ESP32-CAM, such as quantization (e.g. INT8) to minimize the model size and accelerate inference, reduction of memory footprint and optimization of the computational efficiency. These optimizations are important when deep learning models run on devices with limited computing capacity and memory [51].

IV.6 DEPLOYMENT ON ESP32-CAM

With embedded development software such as the Arduino IDE, the optimized model is exported out of the Edge Impulse platform and loaded to the ESP32-CAM. At this stage, the TinyML model is embedded in the firmware, camera settings (resolution, frame size) are customized and memory allocation and optimization is performed. The ESP32-CAM offers increased privacy and reduced overhead of the communication as it operates as a fully independent edge device that does not need any connection to the cloud [10].

IV.7 REAL-TIME INFERENCE AND DETECTION

Once deployed, the ESP32-CAM is used by the system to do real-time inference. It consists of the TinyML model built into the device to manipulate live picture frames. The system is capable of producing the bounding boxes and the class labels, detects the two or more classes of objects at the same time and runs on tight memory and processing requirements. In-device intelligence ensures reliable performance and low latency in the environment where the network coverage is weak or absent [10], [49].

V. EXPERIMENTAL RESULTS AND PERFORMANCE EVALUATION

V.1 EXPERIMENTAL SETUP

Figure 3 shows the Experimental setup of proposed system. The proposed system is divided into four phases *(I) TinyML model creation*: In this step we have used ESP32-CAM board itself to generate dataset. To pre-process dataset, all necessary steps shown in subtopic III.1.1 performed. Finally, for training the model we have used total 1000 grayscale, 96x96 sized images of different class. The model training, testing, evaluation, optimization and quantization steps are performed on Edge Impulse Studio. *(II) Hardware Setup*: In

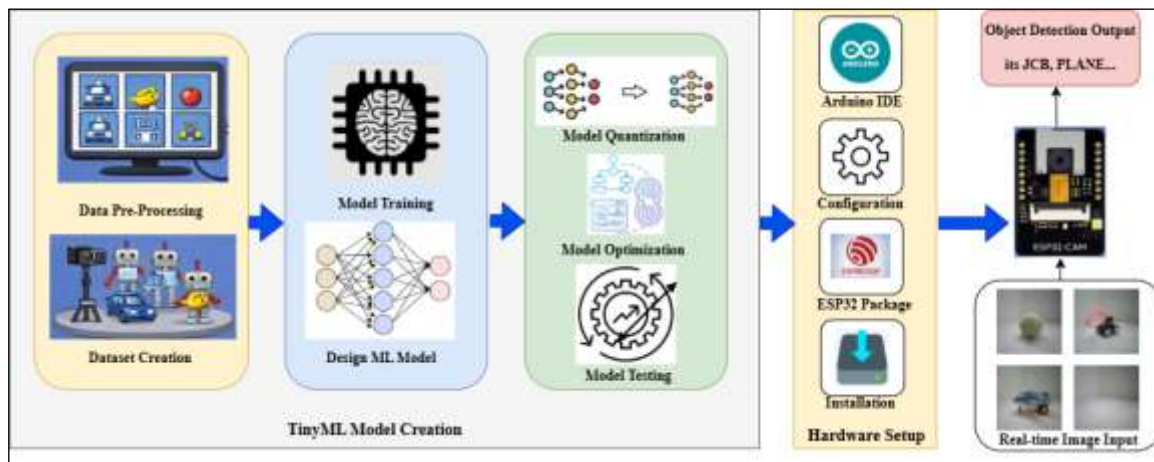


Figure 3: Proposed System Software and Hardware Setup.

Source: Authors, (2026).



Figure 4: Experimental Setup.

Source: Authors, (2026).

This step Arduino IDE needs to prepare with installation of ESP32-CAM board libraries from Espressif and properly configured to use it for ESP32-CAM board. **(III) Compilation and Deployment of ML model:** The TinyML model generated from Edge Impulse Studio is in the library format. Combining the TinyML model library along with Arduino code and then compiled to create firmware. This firmware will run in ESP32-CAM board for real-time operation. **(IV) Real-time Testing:** After deployment, the hardware is ready to work in real-time mode. Now the testing is done with real-time camera and we can evaluate the performance on real-time basis. Figure 4 shows the Experimental setup where real-time testing is done.

V.2 DETECTION RESULTS AND PERFORMANCE EVALUATION

Table 2 is a summary of the performance of the proposed TinyML-based object recognition system, which presents the class-wise and multi-class wise evaluation metrics, such as precision, recall, and F1-score, of the object types. The objects are tested based on single class object (i.e. Plane, JCB, Ball, No-object), Dual class objects (i.e. Plane and Ball, Plane and JCB, Ball and JCB) and multi class objects (i.e. Ball, Plane, JCB all together). The proposed TinyML-based detection system is highly performing across all the categories of items, as observed in the class-wise report presented in Table 2. In the results, we have represented the confusion matrix in table format. Evaluation is done based on single, dual and multi class wise object detections. The parameters of object detection are taken in Table 2 represents TP (True Positive) and FP (False Positive) values. Based on this TP, FP and FN values, Precision, Recall and F1-Score are calculated for different object classes. The final result shows Average Precision percentage 97.89%, Recall percentage 98.17% and F1-Score 98.02%. The results justify the effectiveness of the model in real-time embedded applications showing that the presented system is able to attain high detection rates in all of the classes as well as in multi class object detection. Furthermore, the precision in the Background class is 99, with the recall of 100, which means that this model is very capable of differentiating object and non-object regions. When the results

Table 2: Object Detection Results and Evaluation.

Class Label	TP	FP	FN	Precision (%)	Recall (%)	F1-Score (%)
PLANE	50	1	1	98.04	98.04	98.04
JCB	50	1	2	98.04	96.15	97.09
BALL	50	2	1	96.15	98.04	97.09
PLANE and JCB	75	1	1	98.68	98.68	98.68
PLANE and BALL	75	2	1	97.40	98.68	98.04
JCB and BALL	75	1	1	98.68	98.68	98.68
PLANE, JCB and BALL	100	2	3	98.04	97.09	97.56
BACKGROUND	50	1	0	98.04	100.00	99.01
Average	-	-	-	97.89	98.17	98.02

Source: Authors, (2026).

Are obtained with a very high detection performance, it is imperative to keep in mind that the results were obtained under controlled experimental conditions using a custom dataset. Performance may vary in real world scenarios with more complex environments, and variations in invisible objects, which have different illumination. The Figure 5, 6 and 7 shows real screenshot of Arduino IDE Terminal output showing single-class, dual-class and multi-class object detection with accuracy, coordinates (x, y of object detection) and latency.

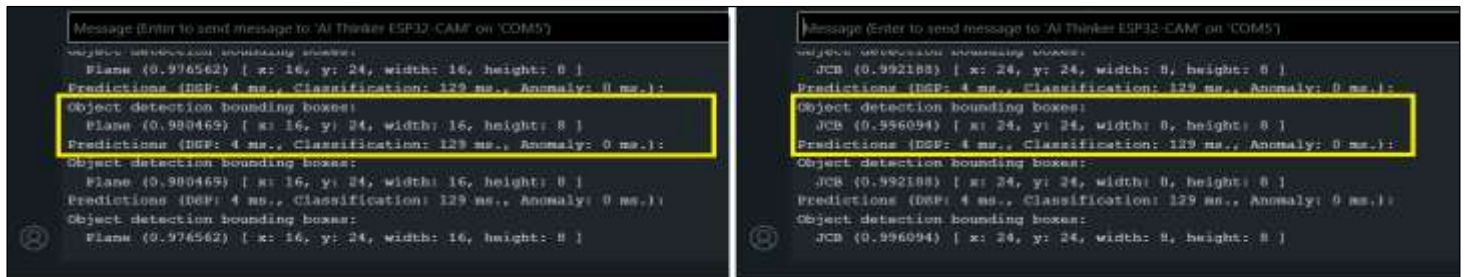


Figure 5: Screenshot of Arduino IDE Terminal output showing single-class object detection with accuracy, coordinate and latency.
Source: Authors, (2026).

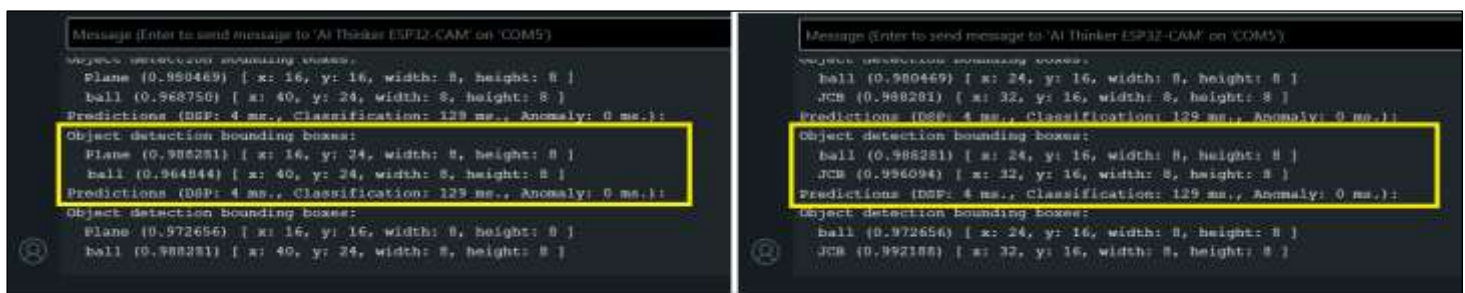


Figure 6: Screenshot of Arduino IDE Terminal output showing dual-class object detection with accuracy, coordinate and latency.
Source: Authors, (2026).

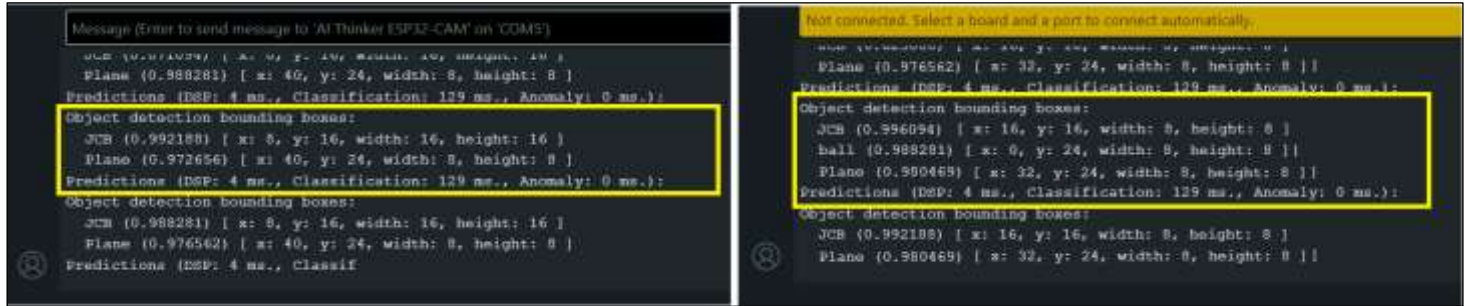


Figure 7: Screenshot of Arduino IDE Terminal output showing multi-class object detection with accuracy, coordinate and latency.
Source: Authors, (2026).

V.3 REAL-TIME PERFORMANCE AND RESULT ANALYSIS

The performance of the proposed TinyML-based object detection system is tested in real-time on ESP32-CAM platform and the performance is presented in Table 3. This system exhibits the ability to execute real time operation using only low-resource microcontroller by taking an average of 7.5 FPS to perform an effective on-device inference. The mean inference time per frame of approximately 133 ms is required to provide a prompt response to changing scenes, and the overall delay of detection which includes image capture and processing is approximately 133 ms. This model has a small size of 263 KB using the methods of quantization, and a runtime memory footprint of 71.3 KB RAM and 843 KB of total firmware FLASH. The above-optimized model will be the most appropriate to implement on the ESP32-CAM board hardware. The model uses 80 mJ of energy per inference and hence it can be used in low-power edge applications.

Table 3: Real-time Performance Parameters.

Parameter	Value
Model Size (KB)	263 KB
RAM Usage (KB)	71.3 KB
Flash Usage (KB)	843 KB
Detection Latency (ms)	133 ms
Energy per Inference	~80 mJ
Detection Stability	High
Standalone Operation	Yes

Source: Authors, (2026).

V.4 COMPARATIVE ANALYSIS

Recent TinyML-based object identification systems installed on ESP32-class devices are compared in Table 3, with an emphasis on important performance measures including frame rate, latency, model size, accuracy, and F1-score. While maintaining subpar detection accuracy with F1-scores usually around 96% [52], [53], [7], [10], [49] current research frequently achieves frame rates in the range of 1–6 FPS with latency between 150–500 ms [11], [50], [14], [12]. The suggested solution, on the other hand, performs better by attaining a higher frame rate of 7.5 FPS, a lower latency of 133 ms, and an enhanced F1-score of roughly 98.02% while keeping a small model size of about 263 KB and operating completely independently. The suggested system offers effective real-time multi-class object identification directly on the ESP32-CAM, in contrast to certain current methods that depend on cloud support or are restricted to classification tasks [14], [12]. By analysing the above results, it is clearly shown that the proposed research is superior in terms of the trade-off between computational efficiency, accuracy, and real-time performance. This will make AI applications suitable for working on resource-constrained embedded devices.

V.5 DISCUSSION

In general, in a resource-constrained ESP32-CAM environment, the suggested TinyML-based system can provide very precise and dependable multi-class object detection. The results show that embedded edge devices can perform almost state-of-the-art without processing in the cloud, which makes the system qualifiable to real-time applications on its own.

VI. APPLICATION AREAS OF TINYML

Figure 8 shows the TinyML Ecosystem where it can be seen that the prominent areas where the need of TinyML is necessary are e-Health Care, Smart Agriculture, Environment Monitoring, Smart city, Industry 4.0 and Vehicular service. In following section, we discuss several application areas of TinyML.



Figure 8: TinyML Ecosystem.

Source: [54].

VI.1 REAL-WORLD APPLICATIONS

The proposed NextGen Edge AI-based TinyML object detection system has immense potentials in deployment in many real-world scenarios. The system is particularly suited in the environments having limited or no access to clouds because it operates independently, consumes low-energy, and has real-time performance [55] [56] [2]. A combination of TinyML and embedded vision also allows intelligent decision-making on the edge and lowers latency and preserves data privacy [53] [4]. Due to this reason, the system is highly adaptable to applications which require local processing and fast responsiveness [14] [49].

VI.2 SMART SURVEILLANCE SYSTEMS

The primary applications of the proposed system include smart surveillance and security monitoring. The heavy use of centralized servers or cloud-based processing by the traditional surveillance systems introduces the issue of latency and privacy [53], [3]. Alternatively, the on-device object detection supported on-the-fly by the proposed ESP32 based system will allow real-time intrusion detection and continuous monitoring [52], [7]. Such solutions are much useful when scalable deployment at low cost is required in the dispersed environment like campuses, warehouses and residential neighbourhoods [14], [57]. Edge-based vision systems also improve the security of data by minimizing the transmission of data to external servers [4], [58].

VI.3 SMART AGRICULTURE AND ENVIRONMENTAL MONITORING

The proposed method can be effectively implemented in precision agriculture where the monitoring of the real-time condition of crops and the environment is a critical aspect [12], [59], [60], [61], [62]. TinyML-enabled edge devices on platforms such as ESP32-CAM will be able to perform on-object recognition on a device to detect pests, monitor crop health, and detect abnormalities in plant growth without necessarily being connected to the cloud. Embedded edge intelligence applications make real-time monitoring and decision-making processes in agricultural settings possible and allow farmers to perform prompt remedial actions, as well as maximize productivity [60], [61], [62], [63], [64], [65] because of their high efficiency, low power requirements, and low infrastructure demands. TinyML, when paired with IoT, can enhance the level of automation and scalability and can be used in smart farming practices such as disease detection, yield prediction, and automated water supply [59], [62], [64]. The result of this implementation is a sustainable development in agriculture with less resource use and operating cost and high efficiency.

VI.4 INDUSTRIAL AND SAFETY APPLICATIONS

Edge systems based on TinyML may infer directly in the device, giving immediate feedback to critical events and creating real-time hazard detection and worker safety deals. In industry IoT where it is integrated with edge AI it can be generally used to predictively maintain devices and detect anomalies. The low energy consumption and latency of TinyML will be suitable in continuous monitoring of environments where safety is paramount [49], [66].

VI.5 IOT AND EDGE INTELLIGENCE INTEGRATION

By incorporating the suggested technology into bigger IoT systems, it is possible to enable distributed edge intelligence [56], [53]. The multiple ESP32-CAM sensors can be used to support such applications as smart home automation, smart city infrastructure, and traffic monitoring [57], [67]. When required, edge-cloud hybrid systems have centralized analytics and local inference combined to enhance additional scalability [2], [68]. This integration does not compromise the benefits of the low-latency edge processing, and it ensures that resources are utilized effectively [53].

VII. KEY CHALLENGES AND FUTURE SCOPE OF RESEARCH

VII.1 KEY CHALLENGES

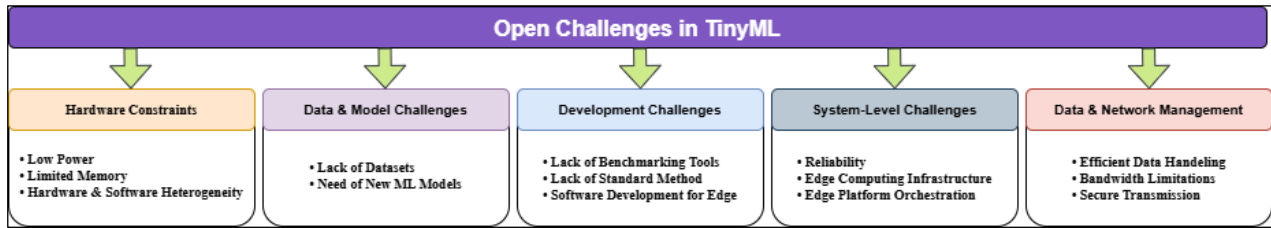


Figure 9: Summary of key Challenges.

Source: Authors, (2026).

(I) Low Power: A limitation on power that edge devices have is one of the main challenges with TinyML-based systems. The TinyML paradigm demands that the models be able to operate with very limited power budgets, and still be acceptable, in accuracy; energy savings must be a key design consideration [69], [70]. The TinyML systems should be optimized well to achieve balance between the computational performance and energy consumption mainly in the inference process. No single method of power management is applicable to all systems due to hardware architecture and preprocessing pipeline differences [70]. Also, when many sensors and peripherals of edge devices are not properly controlled, they can cause inefficient energy consumption [49]. To support reliable and sustainable edge intelligence, energy-efficient TinyML system design remains to be a critical issue. **(II) Limited Memory:** The other issue with TinyML systems is the memory usage limit of edge devices. TinyML is often implemented in microcontrollers with extremely small memory size - often just a few kilobytes of RAM and 1 MB or less of flash storage [4], [70]. Due to a significantly larger memory and processing ability of the system, it is incredibly challenging to implement machine learning models with such a strict memory constraint [70]. These traditional systems often require terabytes or megabytes of memory and this is not feasible within embedded edge devices. Therefore, effective model compression, quantization, and memory-constrained design methods are needed to facilitate the deployment of TinyML to this type of hardware platform [49]. Consequently, the issue of memory constraints remains a major setback in the growth and scalability of TinyML-based edge systems. **(III) Hardware and Software Heterogeneity:** TinyML systems find it extremely challenging to achieve effective learning and deployment methodologies as not all edge platforms have standardized hardware and software systems. The heterogeneity of microcontroller-based systems, toolchains, and runtime environments is a major impediment to machine learning model optimization, portability, and interoperability [71], [3].

Moreover, avoiding the need to develop scalable and reusable solutions to a number of embedded devices is difficult because no existing frameworks and standards are available on TinyML implementation [72]. **(IV) Absence of Benchmarking Tools:** Traditional machine learning benchmark consumes a lot of memory and processing power when making inferences, and is typically designed to run on high-performance computers. These requirements make them unsuitable in the context of TinyML, where devices are constrained to highly restrictive memory and processing capacity requirements [73], [72]. This renders it extremely challenging to apply generic benchmarking designs to the microcontroller-based systems. The consumption of resources, latency, and energy efficiency should also be considered closely when benchmarking TinyML models; these variables are not usually considered by more conventional evaluation techniques, even when adjusted to embedded hardware [72]. To support the heterogeneity of hardware and be able to offer fair performance measurement across TinyML platforms, benchmarking methodologies have to be re-engineered and re-optimized [74]. **(V) Lake of Datasets:** Due to their limited resources and power requirements, data sets that are being used in conventional machine learning systems are often not suitable with TinyML architectures. The spatial and temporal resolution of external sensors data has to be accurately aligned with the feature-extraction requirements of TinyML models [75], [76]. to gain an effective processing. Simplification of data with low-power awareness should also consider the diversity of the datasets, variability, and the amount of noise which greatly influences the performance of the embedded devices model [76]. This is due to the fact that TinyML is not as easy to train and deploy systems because of the absence of standard and high quality datasets. And therefore we require dataset design methods that are suitable to resource-constrained devices [58].

(VI) Absence of Popularly Accepted Methods: Standardized designs of deployment and evaluation Lack of popularly accepted methods Traditional machine learning infrastructures employ numerous learning models, which provide standardized designs to perform both deployment and evaluation For example, because of their efficiency and scalability, lightweight deep neural network models like MobileNet have become essential benchmarks for mobile and edge computing applications [8]. Nevertheless, there are presently no comparable, well-defined, and widely used model architectures in the TinyML ecosystem specifically designed for extremely constrained microcontroller-based settings [7]. Large-scale adoption of TinyML solutions, reproducibility, and benchmarking consistency are all hampered by the lack of widely used techniques. To improve the ecosystem and facilitate more effective, comparable deployments across various edge platforms, standardized TinyML models and frameworks must be developed [77]. **(VII) Edge Computing Infrastructure:** Deploying ML models on edge devices is made extremely difficult by the rapid evolution of edge computing resources, especially because edge infrastructure is dynamic and heterogeneous [78], [79]. The successful implementation of TinyML systems is sometimes hampered by the current edge designs' inadequate support for flexible device setup and smooth integration of increasing hardware capabilities [79]. Moreover, such aspects as mobility and reliable devices continue being significant challenges when the model is implemented, especially in distributed and dynamic contexts [80]. In order to enhance the scalability, flexibility, and deployment efficiency of TinyML systems to modern edge ecosystems, new enablers include network virtualization, containerization technologies, smart handover mechanisms, and isomorphic computing techniques are being studied [78], [80]. **(VIII) Edge Platform Orchestration:** Edge workloads are dynamic and edge devices have limited computing capabilities that are significant challenges to effective resource allocation in existing edge platforms.

The resource allocation and dynamic distribution remain to be the critical issues that directly affect the scalability and performance of the system [81], [82]. These challenges can be solved with data-based optimization control and adaptive resource management algorithms that are able to adjust themselves immediately to the variations in workload [83]. **(IX) Data and Network Management:** In the case of tinyML systems, network and data management must be considered properly as improper treatment of various sensor data may reduce edge networks intelligence and speed significantly [77]. The complexity of the lifecycle of sensor data and data integration between multiple sources makes deployment in real-world environments more complex. A solution to these problems can be effective and self-adaptive edge intelligent system, where data-aware algorithms, fault detection systems and adaptive learning methodologies can be used to solve these problems [76]. **(X) Reliability:** To be used on a large-scale scale in the real world, it is necessary to design reliable TinyML solutions, particularly in edge scenarios where devices are limited in their conditions of operation. Aging of devices, soft errors due to radiation effects and process variations during the manufacture of chip is the key dependability concern that may cause significant system accuracy and performance. Also, edge devices are susceptible to clock circuit, battery, and processing unit faults because they may frequently be deployed in unpleasant or remote locations with little or no human support.

Thus, reliability testing and fault tolerance during TinyML systems implementation are essential prior to the practical applications [55], [72], [84]. **(XI) Software Development:** Security, dynamic device configuration, and flexible deployment limits are some of the issues that must be addressed while developing software for edge AI systems. Scalable and portable application deployment across remote edge environments is made possible by the widespread adoption of containerization technologies and virtualization frameworks. Heterogeneous edge hardware and software interaction, however, continues to be a significant obstacle. Lightweight machine learning models, self-adapting intelligent agents, and sophisticated methods like distributed optimization and swarm intelligence for cooperative learning across edge nodes can all be used to solve this problem [85], [86]. **(XII) ML Modelling Requirement:** The TinyML ecosystem requires the creation of new machine learning models that are appropriate for situations with limited resources. To facilitate quick and flexible decision-making at the edge, techniques including transfer learning, reinforcement learning, online learning, and federated learning can be combined. For real-time performance in an embedded system, a combined design of an ML model with a communication and control mechanism is required. To facilitate such deployment on microcontroller-based systems, model optimization methods such as pruning and quantization are necessary to make models computationally simpler and memory-efficient [55], [87]. During the earlier stages of the paper, we discussed the main challenges that come along with this, such as restricted budgets of resources, the absence of standardized benchmarking strategies, and trade-offs between the accuracy and size of models, etc. challenges in updating deployed models; sensor data is subject to environmental parameters, resulting in diminished model accuracy and performance, etc. All these challenges introduce a scope of future research, particularly to achieve the on-device learning technology, benchmark the technology, support over-the-air upgrades and deployments of the firmware, and enhance model quantization vs. accuracy trade-offs.

VIII. CONCLUSIONS

The given proposal demonstrates an application of an intelligent vision system to a resource-limited hardware with the introduction of NextGen Edge AI-enabled TinyML platform of real-time multi-class object recognition on a self-sufficient ESP32-CAM platform. The suggested approach, lightweight ML models optimization, and on-device inference, manages to strike a balance between the correctness of the models, the latency, and resource consumption. The system has been proved to be suitable to the real-time edge applications through experimental results, and low memory footprint, close to perfection detection, and low power consumption have been achieved. The proposed study compares to the conventional cloud-based solution to offer low latency, low bandwidth usage, and stable functionality in non-network. This publication defines the scaled architecture of the next-generation edge intelligence. In order to enhance further system generalization and robustness, future studies must be dedicated to adaptive learning and creation of a heterogeneous TinyML ecosystem and integration of multimodal sensors.

IX. AUTHOR'S CONTRIBUTION

Conceptualization: Satishkumar Kataria, Pankaj kumar Prajapati.

Methodology: Satishkumar Kataria.

Investigation: Satishkumar Kataria.

Discussion of results: Satishkumar Kataria, Pankaj kumar Prajapati.

Writing – Original Draft: Satishkumar Kataria.

Writing – Review and Editing: Satishkumar Kataria, Pankaj kumar Prajapati.

Resources: Satishkumar Kataria.

Supervision: Pankaj kumar Prajapati.

Approval of the final text: Satishkumar Kataria, Pankaj kumar Prajapati.

X. REFERENCES

- [1] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," Dec. 10, 2015, *arXiv*: arXiv:1512.03385. doi: 10.48550/arXiv.1512.03385.
- [2] M. Satyanarayanan, "The Emergence of Edge Computing," *Computer*, vol. 50, no. 1, pp. 30–39, Jan. 2017, doi: 10.1109/MC.2017.9.
- [3] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge Computing: Vision and Challenges," *IEEE Internet Things J.*, vol. 3, no. 5, pp. 637–646, Oct. 2016, doi: 10.1109/JIOT.2016.2579198.
- [4] P. Warden and D. Situnayake, *TinyML: Machine Learning with TensorFlow Lite on Arduino and Ultra-Low-Power Microcontrollers*, 2nd ed. O'Reilly, 2023.
- [5] J. Chen and X. Ran, "Deep Learning With Edge Computing: A Review," *Proc. IEEE*, vol. 107, no. 8, pp. 1655–1674, Aug. 2019, doi: 10.1109/JPROC.2019.2921977.

- [6] “Espressif Systems,” “ESP32-CAM Datasheet,” 2021. [Online]. Available: https://www.espressif.com/sites/default/files/documentation/esp32-cam_datasheet_en.pdf
- [7] P. Mittal, “A comprehensive survey of deep learning-based lightweight object detection models for edge devices,” *Artif Intell Rev*, vol. 57, no. 9, p. 242, Aug. 2024, doi: 10.1007/s10462-024-10877-1.
- [8] A. G. Howard *et al.*, “MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications,” 2017, *arXiv*. doi: 10.48550/ARXIV.1704.04861.
- [9] A. Aldubaikhi and S. Patel, “Advancements in Small-Object Detection (2023–2025): Approaches, Datasets, Benchmarks, Applications, and Practical Guidance,” *Applied Sciences*, vol. 15, no. 22, p. 11882, Nov. 2025, doi: 10.3390/app152211882.
- [10] “Edge Impulse,” “Building Embedded Machine Learning Applications,” 2024. [Online]. Available: <https://docs.edgeimpulse.com/docs>
- [11] J. Paul, L. Schmid, M. Klaiber, and M. Rössle, “Extraction of Measurement Device Information on an ESP32 Microcontroller: TinyML for Image Processing,” *Procedia Computer Science*, vol. 246, pp. 2002–2011, 2024, doi: 10.1016/j.procs.2024.09.670.
- [12] D. Donskoy, V. Gvindjiliya, and E. Ivliev, “TinyML Classification for Agriculture Objects with ESP32,” *Digital*, vol. 5, no. 4, p. 48, Oct. 2025, doi: 10.3390/digital5040048.
- [13] F. A. F. Prado, W. C. P. Campos, S. A. C. Tacuri, and F. D. C. Alva, “Design of a communication system Images for identification of vehicle plates”.
- [14] Y.-H. Chang, F.-C. Wu, and H.-W. Lin, “Design and Implementation of ESP32-Based Edge Computing for Object Detection,” *Sensors*, vol. 25, no. 6, p. 1656, Mar. 2025, doi: 10.3390/s25061656.
- [15] V. J. Reddi *et al.*, “Widening Access to Applied Machine Learning with TinyML,” Jun. 09, 2021, *arXiv*: arXiv:2106.04008. doi: 10.48550/arXiv.2106.04008.
- [16] R. Samanta, B. Saha, S. K. Ghosh, and R. B. Roy, “Optimizing TinyML: The Impact of Reduced Data Acquisition Rates for Time Series Classification on Microcontrollers,” Sep. 17, 2024, *arXiv*: arXiv:2409.10942. doi: 10.48550/arXiv.2409.10942.
- [17] “TensorFlow Blog.” [Online]. Available: <https://blog.tensorflow.org/2021/05/building-tinymml-application-with-tf-micro-and-sensiml.html>
- [18] T. Langer, M. Widra, and V. Beyer, “TinyML Towards Industry 4.0: Resource-Efficient Process Monitoring of a Milling Machine,” Aug. 22, 2025, *arXiv*: arXiv:2508.16553. doi: 10.48550/arXiv.2508.16553.
- [19] S. Heydari and Q. H. Mahmoud, “Tiny Machine Learning and On-Device Inference: A Survey of Applications, Challenges, and Future Directions,” *Sensors*, vol. 25, no. 10, p. 3191, May 2025, doi: 10.3390/s25103191.
- [20] A. Dey, S. Srivastava, G. Singh, and R. G. Pettit, “Real-Time Performance Benchmarking of TinyML Models in Embedded Systems (PICO: Performance of Inference, CPU, and Operations),” Sep. 05, 2025, *arXiv*: arXiv:2509.04721. doi: 10.48550/arXiv.2509.04721.
- [21] G. Delnevo, S. Mirri, C. Prandi, and P. Manzoni, “An evaluation methodology to determine the actual limitations of a TinyML-based solution,” *Internet of Things*, vol. 22, p. 100729, Jul. 2023, doi: 10.1016/j.iot.2023.100729.
- [22] “LearnOpenCV.” [Online]. Available: <https://learnopencv.com/tensorflow-lite-model-optimization-for-on-device-machine-learning/>
- [23] “hackster.io.” [Online]. Available: <https://www.hackster.io/news/easy-tinymml-on-esp32-and-arduino-a9dbc509f26c>
- [24] “ELOQUENT ARDUINO.” [Online]. Available: <https://eloquentarduino.com/posts/tensorflow-lite-tinymml-esp32>
- [25] “GITHUB.” [Online]. Available: <https://github.com/tensorflow/tflite-micro>
- [26] “EDGE IMPULSE.” [Online]. Available: <https://docs.edgeimpulse.com/studio/projects/deployment>
- [27] “Google AI for Developers.” [Online]. Available: <https://ai.google.dev/edge/litert/microcontrollers/overview>
- [28] “ST Microelectronics.” [Online]. Available: <https://stm32ai.st.com/stm32-cube-ai/>
- [29] G. Armeniakos, G. Mentzos, and D. Soudris, “Accelerating TinyML Inference on Microcontrollers through Approximate Kernels,” Sep. 25, 2024, *arXiv*: arXiv:2409.16815. doi: 10.48550/arXiv.2409.16815.
- [30] “Apache TVM.” [Online]. Available: <https://tvm.apache.org/2020/06/04/tinymml-how-tvm-is-taming-tiny>
- [31] “daobook.github.io.” [Online]. Available: https://daobook.github.io/tvm/docs/arch/microtvm_design.html
- [32] “NVIDIA TensorRT Documentation.” [Online]. Available: <https://docs.nvidia.com/deeplearning/tensorrt/latest/index.html>
- [33] “ONNX RUNTIME.” [Online]. Available: <https://onnxruntime.ai/docs/tutorials/mobile/>
- [34] “Arduino official website.” [Online]. Available: <https://docs.arduino.cc/hardware/nano-33-ble-sense/>
- [35] “Sparkfun Electronics.” [Online]. Available: <https://www.sparkfun.com/sparkfun-edge-development-board-apollo3-blue.html>
- [36] “STMicroelectronics.” [Online]. Available: <https://www.st.com/en/evaluation-tools/stm32f4discovery.html>
- [37] “ESPRESSIF.” [Online]. Available: <https://docs.espressif.com/projects/esp-dev-kits/en/latest/esp32/esp32-devkitc/index.html>
- [38] “ESPRESSIF.” [Online]. Available: <https://www.espressif.com/en/products/devkits/esp-eye/overview>
- [39] “SONY.” [Online]. Available: <https://developer.spsense.sony-semicon.com/>

- [40] E. Flamand *et al.*, “GAP-8: A RISC-V SoC for AI at the Edge of the IoT,” in *2018 IEEE 29th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, Milan: IEEE, Jul. 2018, pp. 1–4. doi: 10.1109/ASAP.2018.8445101.
- [41] “OpenMV.” [Online]. Available: <https://openmv.io/products/openmv-cam-h7-plus?srltid=AfmBOorEwSINqVxm8RV3ze7xfHHYw3ipE43skpIoAW6Mmes06tMdL5pH>
- [42] “NORDIC Semiconductor.” [Online]. Available: <https://www.nordicsemi.com/Products/Development-hardware/nRF52840-DK>
- [43] “TEXAL INSTRUMENT.” [Online]. Available: <https://www.ti.com/tool/LAUNCHXL-CC1352P>
- [44] “Raspberry Pi.” [Online]. Available: <https://forums.raspberrypi.com/viewtopic.php?t=324642&sid=dacfl774122e6bebd9a1d523722e2068>
- [45] “Raspberry Pi.” [Online]. Available: <https://www.raspberrypi.com/products/raspberry-pi-5/>
- [46] M. Satyanarayanan, “The Emergence of Edge Computing,” *Computer*, vol. 50, no. 1, pp. 30–39, Jan. 2017, doi: 10.1109/MC.2017.9.
- [47] E. Li, L. Zeng, Z. Zhou, and X. Chen, “Edge AI: On-Demand Accelerating Deep Neural Network Inference via Edge Computing,” 2019, *arXiv*. doi: 10.48550/ARXIV.1910.05316.
- [48] C. Shorten and T. M. Khoshgoftaar, “A survey on Image Data Augmentation for Deep Learning,” *J Big Data*, vol. 6, no. 1, p. 60, Dec. 2019, doi: 10.1186/s40537-019-0197-0.
- [49] J. Lin, W.-M. Chen, Y. Lin, J. Cohn, C. Gan, and S. Han, “MCUNet: Tiny Deep Learning on IoT Devices,” Nov. 19, 2020, *arXiv*: arXiv:2007.10319. doi: 10.48550/arXiv.2007.10319.
- [50] R. David *et al.*, “TensorFlow Lite Micro: Embedded Machine Learning on TinyML Systems,” Mar. 13, 2021, *arXiv*: arXiv:2010.08678. doi: 10.48550/arXiv.2010.08678.
- [51] S. Han, H. Mao, and W. J. Dally, “Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding,” Feb. 15, 2016, *arXiv*: arXiv:1510.00149. doi: 10.48550/arXiv.1510.00149.
- [52] J. Moosmann, M. Giordano, C. Vogt, and M. Magno, “TinyissimoYOLO: A Quantized, Low-Memory Footprint, TinyML Object Detection Network for Low Power Microcontrollers,” in *2023 IEEE 5th International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, Jun. 2023, pp. 1–5. doi: 10.1109/AICAS57966.2023.10168657.
- [53] M. Merenda, C. Porcaro, and D. Iero, “Edge Machine Learning for AI-Enabled IoT Devices: A Review,” *Sensors*, vol. 20, no. 9, p. 2533, Apr. 2020, doi: 10.3390/s20092533.
- [54] R. Sanchez-Iborra and A. F. Skarmeta, “TinyML-Enabled Frugal Smart Objects: Challenges and Opportunities,” *IEEE Circuits Syst. Mag.*, vol. 20, no. 3, pp. 4–18, 2020, doi: 10.1109/MCAS.2020.3005467.
- [55] S. Heydari and Q. H. Mahmoud, “Tiny Machine Learning and On-Device Inference: A Survey of Applications, Challenges, and Future Directions,” *Sensors*, vol. 25, no. 10, p. 3191, May 2025, doi: 10.3390/s25103191.
- [56] Dr. L. Dutta and S. Bharali, “TinyML Meets IoT: A Comprehensive Survey,” *Internet of Things*, vol. 16, p. 100461, Dec. 2021, doi: 10.1016/j.iot.2021.100461.
- [57] J. Liu, C. Liu, B. Wang, G. Gao, and S. Wang, “Optimized Task Allocation for IoT Application in Mobile-Edge Computing,” *IEEE Internet Things J.*, vol. 9, no. 13, pp. 10370–10381, Jul. 2022, doi: 10.1109/JIOT.2021.3091599.
- [58] N. N. Alajlan and D. M. Ibrahim, “TinyML: Enabling of Inference Deep Learning Models on Ultra-Low-Power IoT Edge Devices for AI Applications,” *Micromachines*, vol. 13, no. 6, p. 851, May 2022, doi: 10.3390/mi13060851.
- [59] A. Kamilaris and F. X. Prenafeta-Boldú, “Deep learning in agriculture: A survey,” *Computers and Electronics in Agriculture*, vol. 147, pp. 70–90, Apr. 2018, doi: 10.1016/j.compag.2018.02.016.
- [60] F. A. Wulnye, E. A. E. Arthur, D. A. N. Gookyi, K. Ahiadormey, J. O. Agyemang, and K. O.-B. O. Agyekum, “TinyML for Smart Agriculture: Comparative Analysis of TinyML Platforms and Practical Deployment for Maize Leaf Disease Identification”.
- [61] V. Tsoukas, A. Gkogkidis, and A. Kakarountas, “A TinyML-based System For Smart Agriculture,” in *Proceedings of the 26th Pan-Hellenic Conference on Informatics*, Athens Greece: ACM, Nov. 2022, pp. 207–212. doi: 10.1145/3575879.3575994.
- [62] M. Nawaz and M. I. K. Babar, “IoT and AI for smart agriculture in resource-constrained environments: challenges, opportunities and solutions,” *Discov Internet Things*, vol. 5, no. 1, p. 24, Mar. 2025, doi: 10.1007/s43926-025-00119-3.
- [63] T. Malche, M. Joshi, G. M. Upadhyay, and P. K. Soni, “Automated tomato leaf disease detection and alert system using Internet of Things and TinyML,” *Discov Internet Things*, vol. 6, no. 1, p. 8, Dec. 2025, doi: 10.1007/s43926-025-00257-8.
- [64] A. M. Hayajneh, S. A. Aldalahmeh, F. Alasali, H. Al-Obiedollah, S. A. Zaidi, and D. McLernon, “Tiny machine learning on the edge: A framework for transfer learning empowered unmanned aerial vehicle assisted smart farming,” *IET Smart Cities*, vol. 6, no. 1, pp. 10–26, Mar. 2024, doi: 10.1049/smc2.12072.
- [65] D. A. N. Gookyi *et al.*, “TinyML for smart agriculture: Comparative analysis of TinyML platforms and practical deployment for maize leaf disease identification,” *Smart Agricultural Technology*, vol. 8, p. 100490, Aug. 2024, doi: 10.1016/j.atech.2024.100490.
- [66] M. Antonini, M. Pincheira, M. Vecchio, and F. Antonelli, “An Adaptable and Unsupervised TinyML Anomaly Detection System for Extreme Industrial Environments,” *Sensors*, vol. 23, no. 4, p. 2344, Feb. 2023, doi: 10.3390/s23042344.
- [67] M. Batty *et al.*, “Smart cities of the future,” *Eur. Phys. J. Spec. Top.*, vol. 214, no. 1, pp. 481–518, Nov. 2012, doi: 10.1140/epjst/e2012-01703-3.

- [68] F. Bove and L. Bedogni, “Smart Split: Leveraging TinyML and Split Computing for Efficient Edge AI,” in *2024 IEEE/ACM Symposium on Edge Computing (SEC)*, Rome, Italy: IEEE, Dec. 2024, pp. 456–460. doi: 10.1109/SEC62691.2024.00052.
- [69] L. Xia, D. Yang, J. Zhang, H. Yang, and J. Chen, “Enhanced Semantic Information Transfer of Multi-Domain Samples: An Adversarial Edge Detection Method Using Few High-Resolution Remote Sensing Images,” *Sensors*, vol. 22, no. 15, p. 5678, Jul. 2022, doi: 10.3390/s22155678.
- [70] J. Chen and X. Ran, “Deep Learning With Edge Computing: A Review,” *Proc. IEEE*, vol. 107, no. 8, pp. 1655–1674, Aug. 2019, doi: 10.1109/JPROC.2019.2921977.
- [71] Dr. L. Dutta and S. Bharali, “TinyML Meets IoT: A Comprehensive Survey,” *Internet of Things*, vol. 16, p. 100461, Dec. 2021, doi: 10.1016/j.iot.2021.100461.
- [72] L. Capogrosso, F. Cunico, D. S. Cheng, F. Fummi, and M. Cristani, “A Machine Learning-oriented Survey on Tiny Machine Learning,” Sep. 26, 2023, *arXiv:2309.11932*. doi: 10.48550/arXiv.2309.11932.
- [73] C. Banbury *et al.*, “MLPerf Tiny Benchmark,” Aug. 24, 2021, *arXiv:2106.07597*. doi: 10.48550/arXiv.2106.07597.
- [74] P. Mattson *et al.*, “MLPerf Training Benchmark,” Mar. 02, 2020, *arXiv:1910.01500*. doi: 10.48550/arXiv.1910.01500.
- [75] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*, vol. 521, no. 7553, pp. 436–444, May 2015, doi: 10.1038/nature14539.
- [76] H. Li, K. Ota, and M. Dong, “Learning IoT in Edge: Deep Learning for the Internet of Things with Edge Computing,” *IEEE Network*, vol. 32, no. 1, pp. 96–101, Jan. 2018, doi: 10.1109/MNET.2018.1700202.
- [77] S. Deng, H. Zhao, W. Fang, J. Yin, S. Dustdar, and A. Y. Zomaya, “Edge Intelligence: The Confluence of Edge Computing and Artificial Intelligence,” *IEEE Internet Things J.*, vol. 7, no. 8, pp. 7457–7469, Aug. 2020, doi: 10.1109/JIOT.2020.2984887.
- [78] W. Yu *et al.*, “A Survey on the Edge Computing for the Internet of Things,” *IEEE Access*, vol. 6, pp. 6900–6919, 2018, doi: 10.1109/ACCESS.2017.2778504.
- [79] M. Chiang and T. Zhang, “Fog and IoT: An Overview of Research Opportunities,” *IEEE Internet Things J.*, vol. 3, no. 6, pp. 854–864, Dec. 2016, doi: 10.1109/JIOT.2016.2584538.
- [80] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, “A Survey on Mobile Edge Computing: The Communication Perspective,” *IEEE Commun. Surv. Tutorials*, vol. 19, no. 4, pp. 2322–2358, 2017, doi: 10.1109/COMST.2017.2745201.
- [81] T. Taleb, K. Samdanis, B. Mada, H. Flinck, S. Dutta, and D. Sabella, “On Multi-Access Edge Computing: A Survey of the Emerging 5G Network Edge Cloud Architecture and Orchestration,” *IEEE Commun. Surv. Tutorials*, vol. 19, no. 3, pp. 1657–1681, 2017, doi: 10.1109/COMST.2017.2705720.
- [82] Y. Mao, J. Zhang, and K. B. Letaief, “Dynamic Computation Offloading for Mobile-Edge Computing With Energy Harvesting Devices,” *IEEE J. Select. Areas Commun.*, vol. 34, no. 12, pp. 3590–3605, Dec. 2016, doi: 10.1109/JSAC.2016.2611964.
- [83] S. Wang, X. Zhang, Y. Zhang, L. Wang, J. Yang, and W. Wang, “A Survey on Mobile Edge Networks: Convergence of Computing, Caching and Communications,” *IEEE Access*, vol. 5, pp. 6757–6779, 2017, doi: 10.1109/ACCESS.2017.2685434.
- [84] M. Terziyska, I. Ilieva, Z. Terziyski, and N. Komitov, “A Review of the Transition from Industry 4.0 to Industry 5.0: Unlocking the Potential of TinyML in Industrial IoT Systems,” *Sci*, vol. 8, no. 1, p. 10, Jan. 2026, doi: 10.3390/sci8010010.
- [85] S. Leroux, P. Simoens, M. Lootus, K. Thakore, and A. Sharma, “TinyMLOps: Operational Challenges for Widespread Edge AI Adoption,” Mar. 27, 2022, *arXiv:2203.10923*. doi: 10.48550/arXiv.2203.10923.
- [86] S. Soro, “Approved for Public Release; Distribution Unlimited. Public Release Case Number 20-2709”.
- [87] V. Rajapakse, I. Karunanayake, and N. Ahmed, “Intelligence at the Extreme Edge: A Survey on Reformable TinyML,” *ACM Comput. Surv.*, vol. 55, no. 13s, pp. 1–30, Dec. 2023, doi: 10.1145/3583683.