Journal of Multiscale Material Informatics
ISSN 3047-5724

Research Article

Resource-Constrained Perception: Deploying quantized Deep Learning Frameworks on Microcontrollers for real-Time Image Classification in Robotic Vision

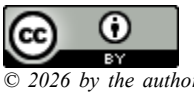
Enjoy Bhodra[1], Md Nahiduzzaman Hridoy[2], M A Shahriar[2], Iffat Salim Toaha[1]*

[1]School of Artificial Intelligence and Computer Science, Jiangsu Normal University, Xuzhou 221116, China
[2]School of Electrical Engineering and Automation, Jiangsu Normal University, Xuzhou 221116, China
*Corresponding: ri_hrm16@yahoo.com or iffatsalimtoaha@jsnu.edu.cn

Abstract
Robotic vision allows machines to analyze their surroundings for applications including obstacle avoidance, navigation, and object detection. Though deep learning algorithms perform well for these purposes, they require vast amounts of memory, computational power, and energy. Tiny robots do not have powerful computers and instead use low-end microcontrollers with severe resource constraints. This paper aims to overcome this limitation by applying model quantization, which involves the conversion of floating-point weights to 8-bit integers. This paper investigates the applicability of two most commonly used quantization techniques: Post-Training Quantization (PTQ) and Quantization-Aware Training (QAT). A tiny convolutional neural network (CNN) with only 28,069 parameters was created and tested on ARM Cortex-M7 and ESP32 microcontrollers. Both techniques proved to reduce the resource costs: Flash memory – by approximately 71%, RAM – by around 60%, inference latency – by more than 50%, and power consumption – by 35-40%. QAT provides a relative accuracy preservation of 99.27% (94.9% accuracy) in comparison with 95.6% accuracy of FP32, and PTQ preserves 98.12% (93.8% accuracy). Quantized models were able to perform in real time with inference latency less than 32 ms on Cortex-M7.

Keywords: Quantization, microcontrollers, Edge AI, Robotic Vision, Lightweight CNN.

Received: 3 August 2026 / **Revised:** 14 August 2026 / **Accepted:** 17 August 2026 / **Published:** 31 August 2026



1. INTRODUCTION

Robotic vision is one of the fundamental technologies of modern robotics. It is used to perceive the environment via cameras. Robots with vision can trace paths, identify people, identify objects, avoid obstacles, and autonomously navigate [1-2]. The applications range from autonomous driving and medical assistance to warehouse automation, quality inspection, and environmental monitoring. In recent years, deep learning has greatly improved the vision of robots. Convolutional Neural Networks (CNNs) have been shown to automatically learn complex visual features and, in general, to perform better than traditional image processing methods [3]. CNNs are extensively applied in the areas of object identification, image segmentation, and classification. But deep learning demands a lot of computing power. Most CNN models have millions of parameters, which require a lot of memory, high-speed processors, and a lot of electricity. Such models can be readily implemented on desktop computers as well as cloud servers [4-5]. But many robots can't fit such hardware; it's too expensive, too heavy, too bulky, and too energy-constrained.

In small mobile robots, microcontrollers are often used in place of high-power CPUs. A microcontroller is a miniaturized computer that contains a processor, memory, and input/output interfaces in a single chip [6]. The energy efficiency and low cost of these units make them attractive. Typical processors used are ARM Cortex-M, ESP32, STM32, and Arduino boards. These platforms are widely used in the Internet of Things (IoT) applications as well as embedded systems [7]. Although the microcontroller has merits, it has a number of constraints. The majority of the devices only have a few hundred kilobytes of RAM and a few megabytes of flash memory [8]. They are much slower than desktop processors. Thus, these platforms are not efficient for standard deep learning models to run.

Edge Artificial Intelligence (Edge AI) is the answer, as it brings deep learning technology from cloud servers to local devices. The model is run on the robot itself, without any data sent to the cloud. This not only decreases the

latency of communication but also increases privacy and facilitates real-time decision-making. But Edge AI still needs to have light neural networks that could be incorporated within hardware constraints.

There are several techniques that can reduce the neural network's complexity. such as pruning, distillation, weight sharing, neural architecture search, and quantization. Quantization is one of the simplest and most effective methods that decrease the numerical precision of the weights and activations of the model from 32-bit floating-point to lower precision – 8-bit integers. Thus, quantization greatly reduces memory consumption and computation speed.

Two commonly used quantization methods now exist. Post-Training Quantization (PTQ) transforms the trained model from the floating-point to integer representation without retraining the model. PTQ is quite simple and fast. Other is Quantization-Aware Training (QAT) incorporates the quantization step into the training process. Thus allowing the model to learn from the quantization error [9-10]. QAT gives better accuracy in comparison with PTQ but takes much more time. Modern advancements in TinyML allow deploying machine learning models on small devices. TinyML is a combination of lightweight neural networks, efficient frameworks, and appropriate hardware. Frameworks such as TensorFlow Lite for Microcontrollers, CMSIS-NN, and TinyEngine enable inference of deep neural networks on tiny memory devices.

However, despite numerous developments in TinyML, there are several problems. Firstly, aggressive quantization reduces the prediction accuracy [11]. Secondly, different microcontrollers possess various hardware, and the model can be optimized only for a specific device. Finally, the power consumption is very important for battery-powered robots, and even fast model can be very costly for operation.

The current research addresses the issues mentioned above through an extensive analysis and comparison of PTQ and QAT on a lightweight CNN architecture in the context of robotic perception. This research framework analyzes memory consumption, energy requirements, execution time, and classification precision under practical conditions. The important contributions of this research work include the following: Offers a highlight deep learning framework for real-time robotic vision using microcontrollers; Using the same neural network design, compares PTQ and QAT quantization approaches; The framework is tested on the ARM Cortex-M7 and ESP32 microcontrollers; The study examines how quantization impacts memory utilization, inference time, power consumption, and prediction accuracy; and Guidelines are provided for deploying deep learning models on low-power robotic devices.

The findings show that carefully built quantized models can achieve outstanding vision accuracy while adhering to the stringent hardware limitations of embedded microcontrollers. The proposed research model provides low-cost intelligent robots that do not require cloud computing and can adapt fast to environmental changes.

2. RELATED WORK

2.1. Deep Learning Robotics

Deep learning techniques have revolutionized computer vision in robotics [12]. Robotics has the ability to identify objects, classify pictures, and track human movements. But such systems usually use computers that have high-capacity RAM, processors, and much memory. Such equipment is not affordable to install in many robots due to cost, mass, and power requirements. Microcontrollers represent an effective solution because they are cheap, small, and energy-efficient [13]. The main problem is that microcontrollers do not have enough RAM, Flash memory, and processing capability. Various solutions to create a more compact and fast deep learning have been proposed by researchers.

2.2. Edge Artificial Intelligence

Edge Artificial Intelligence (AI) process information directly on the device and not on the cloud server. Thus, privacy, reduced latency, and robot functioning without internet connection are ensured [14]. Edge AI has been popular in robotics, healthcare, agriculture, smart homes, and industrial automation. It makes small robots capable of object detection, obstacle avoidance, and real-time decision making. Moreover, microcontrollers consume much less energy compared to GPUs, thus they are perfect for battery-operated robotics. There are various software solutions supporting edge AI. TensorFlow Lite for Microcontrollers (TFLM) is one of the most popular frameworks [15]. It allows performing inference on devices having only a few hundred kilobytes of RAM. CMSIS-NN is another framework developed by ARM providing optimization of neural network kernels for Cortex-M processors [16]. TinyEngine and micro TVM increase the speed of execution on embedded devices. Despite these advantages, the problem of size remains.

2.3. Model Compression Techniques

Model compression decreases the size of neural networks and computational overhead. Many ways of model compression have been developed.

Network Pruning: Pruning deletes unnecessary weights of the trained neural network. Some connections do not play a significant role in making predictions. Eliminating them decreases memory consumption and computational overhead [17]. Network pruning is able to decrease the size of models by 30-90%. At the same time, too much pruning may reduce the accuracy of the prediction because some critical weights will be deleted.

Weight Quantization: Weight quantization transforms high precision numbers into lower-precision numbers. In deep learning models, each weight is usually stored as a 32-bit floating-point number. Transforming into 8-bit integers allows reducing memory consumption by about 75% [18]. Operations with integer numbers consume less CPU than operations with floating-point numbers. Consequently, they increase the performance of computation. INT8 quantization is one of the most popular approaches used in embedded devices.

Knowledge Distillation: It trains the small neural network using the outputs of a big neural network. Thus, a small network learns from the predictions of the teacher instead of training only on data

Low-Rank Approximation: Some layers of neural networks may contain some extra information. Using low-rank approximation, we replace large matrices of weights with small matrices keeping most important information

Lightweight Network Architectures: Instead of compressing an existing neural network, researchers have developed lightweight architectures initially. Among these MobileNet, MobileNetV2, MobileNetV3, ShuffleNet, EfficientNet-Lite, SqueezeNet, etc. are significant [19]. Depthwise separable convolution is one of the key mechanisms allowing reducing computational overhead. MobileNetV2 provides a good compromise between speed and accuracy and is widely used for embedded vision.

2.4. Quantization Approaches

Quantization can be performed using multiple approaches. *Post-Training Quantization (PTQ)*: PTQ is performed after the training process. The initial model is converted from the floating-point model to the INT8 model without retraining [20]. Some of the key benefits are as follows: easy-to-use approach, fast conversion, does not require retraining, and applicable to any pre-existing model. The only downside of PTQ is the reduction in accuracy.

Quantization-Aware Training (QAT): QAT performs quantization in the process of training. The network learns to work with low-precision weights. Advantages include: more precision, improved robustness, and enhanced performance on difficult datasets. On the other hand, the disadvantages are as follows: retraining is required, increased training costs, and higher development time. Numerous studies suggest using QAT for practical implementation as high accuracy is required in robotic vision.

2.5. Vision Systems on Microcontrollers

Past studies demonstrated deep learning can work on small embedded systems [21]. They usually consist of ARM Cortex-M microcontrollers which are popular because they support DSP instructions and efficient integer operations. ESP32 systems have gained their popularity recently as they offer Wi-Fi, Bluetooth, and enough Flash memory for TinyML applications. OpenMV cameras integrate the image sensor and ARM Cortex-M processor with computer vision capabilities. Scientists performed different types of vision tasks. Like as hand gestures recognition, face detection, person detection, and obstacle detection. Multiple studies perform vision tasks using small image sizes (96×96, 128×128, or 160×120). Smaller images allow reducing the memory usage and inference time; however, the detection accuracy of distant objects decreases.

2.6. Literatures Gap and Contribution

Despite extensive experiments and study on TinyML and embedded vision, this research identified some significant literatures deficits. For instances, many research seldom explored empirical comparison of the results of PTQ and QAT on similar hardware and architecture models. Besides, most studies examined a single platform of microcontroller making it challenging to analyze performance across multiple platforms. There are very few studies that carry out a comprehensive performance analysis in terms of memory, latency, power consumption, and accuracy. Even though less literatures have provided recommendations for applying quantized models on microcontroller platforms. So the considering the above this current research work bridges this gap through the empirical comparison of PTQ and QAT on two different microcontroller platforms.

3. METHOD AND MATERIALS

3.1. Study Framework

This study develops a lightweight robotic vision system for microcontrollers. The method uses a quantized Convolutional Neural Network (CNN) to recognize objects in real-time. The primary objective is to reduce memory usage and power consumption while maintaining high prediction accuracy. The complete workflow comprises six steps.

Step 1: Image Acquisition: A camera captures continuous images. Images are resized during processing to 96×96×3 pixels. This resolution reduces memory usage and enhances inference speed.
Step 2: Image Processing: The captured image is normalized. Pixel values are converted from 0-255 to 0-1 using the normalization equation:

$$I_{norm} = \frac{I}{255} \quad (1)$$

where: I = original image, I_{norm} = normalized image. This step improves model stability
Step 3: Neural Network Inference: The processed image enters the lightweight CNN, which extracts image features layer by layer. The network architecture includes input layer, batch normalization, three convolution layers, max pooling, fully connected layer, and SoftMax output.
Step 4: Quantization: The trained floating-point model is converted to INT8 format. Both PTQ and QAT quantization methods are evaluated. Each method reduces model size and improves inference speed.
Step 5: Embedded Deployment: The quantized model is converted to TensorFlow Lite for Microcontrollers format and uploaded to the target microcontroller. The robot performs inference locally without cloud communication.
Step 6: Performance Evaluation: The system measures classification accuracy, inference latency, Flash memory usage, power consumption, and RAM usage.

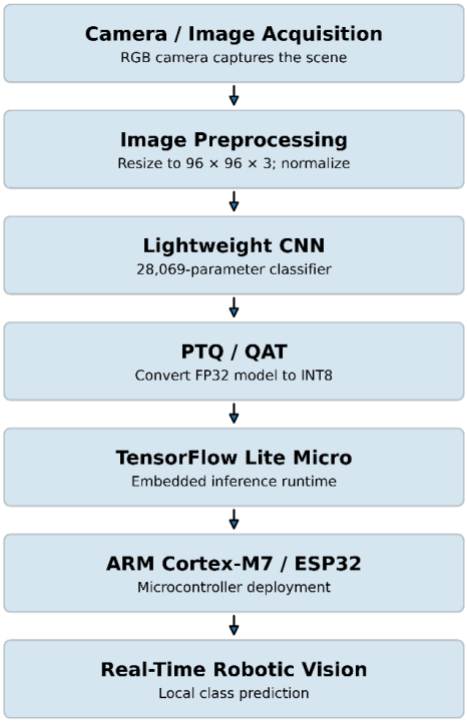


Figure 1. Study framework for end-to-end embedded robotic vision.

3.2. Lightweight CNN Design

A lightweight CNN is established for resource-constrained hardware. Besides a large models involve millions of parameters [22]. Like as models cannot match inside small embedded memory. So, the current study designs a compact CNN (Figure 2). This model is substantially smaller than MobileNetV2 (2.24 million parameters), making it suitable for severely resource-constrained devices.

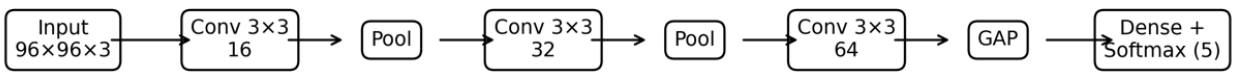


Figure 2. Conceptual compact CNN architecture

Table 1. Proposed CNN architecture.

Layer	Output Size	Parameters
Input	96 × 96 × 3	—
Conv1 (3×3,16)	96 × 96 ×16	448
Max Pool	48 ×48 ×16	0
Conv2 (3×3,32)	48 ×48 ×32	4,640
Max Pool	24 ×24 ×32	0
Conv3 (3×3,64)	24 ×24 ×64	18,496
Global Average Pool	64	0
Dense	64	4,160
Softmax	5 classes	325
Total parameters		28,069

3.3. Convolution Operation

The convolution layer extracts image features. This convolution equation is

$$Y(i,j) = \sum m \sum n\, X(i+m,\text{j+n})\text{W(m,n)+b} \quad (2)$$

where: X = input image, W = filter, b = bias, Y = output feature map.

3.4. Activation Function

Each convolution layer uses the ReLU activation. For that equation is

$$\text{ReLUReLU(x)} = \max(0, \text{x}) \quad (3)$$

ReLU is fast and simple which reduces computational cost.

3.5. Softmax Classifier

Ultimate layer assumes the object class. Here equation for Softmax is

$$P_i = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}} \quad (4)$$

where P_i = the prediction probability, z_i =the output score.

3.6. FP32 Baseline

The FP32 network is used as a numerical baseline. Its performance accuracy is measured against the fixed dataset, while the size of its model, runtime memory, latency, power consumption, and energy are quantified using the same process as for the quantized versions.

3.7. Post-Training Quantization

The initial optimization method is PTQ. CNN is trained using 32-bit floating-point values. Post training all weights are changed into INT8 values. For this equation is

$$Q = \frac{R}{S} + Z \quad (5)$$

where Q = the quantized value, R = the floating-point value, S = the scale factor, Z = the zero point. PTQ needs n retraining. Its benefits are quick conversion, small development, and easy deployment. But, little accuracy loss may happen.

3.8. Quantization-Aware Training (QAT)

QAT introduces quantization during the training period. This model learns the effects of low-precision arithmetic. During each training iteration, the model adapts to quantization errors. QAT typically prduces better accuracy than PTQ at the cost of longer training time.

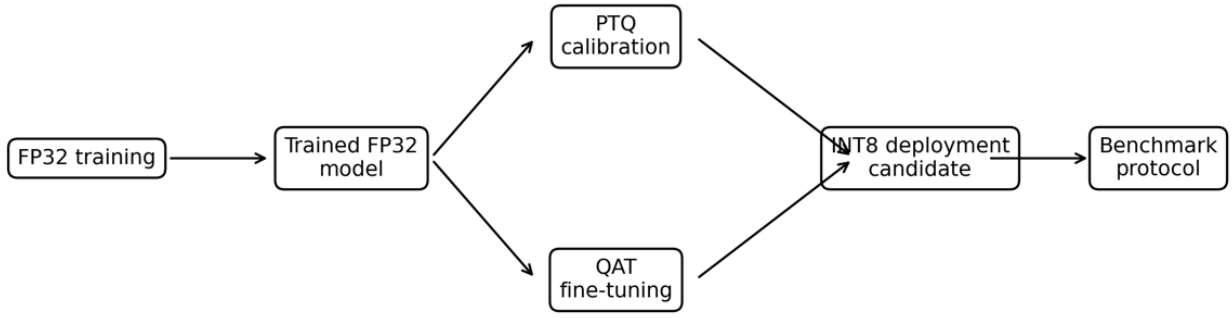


Figure 3. Quantization workflow. PTQ and QAT share the similar FP32 beginning point but change in when quantization effects enter the optimization process

3.9. Memory Optimization

Usage of memory is very crucial. Flash memory storages the neural network. RAM stores intermediate feature maps. In this circumstance's memory usage calculated as

$$\text{Reduction} = \frac{\text{Original}-\text{New}}{\text{Original}} \times 100 \quad (6)$$

here the optimized model must align inside the available imbedded memory.

4. EXPERIMENTAL SETUP

The performance of the proposed quantized deep learning model was examined on two microcontroller-based systems namely ARM Cortex-M7 and ESP32 (Table 2). The accuracy of classification, memory consumption, inference time, and power consumption were considered in the experimentation. This work tried to answer whether quantized neural networks are able to offer real-time robotic perception under hardware constraints.

4.1. Hardware Platforms

The ARM Cortex-M7 architecture was developed using STM32H747I-DISCO (STMicroelectronics). The board consists of STM32H747XI Arm® Cortex®-M7 processor which is characterized by the following: Processor: STM32H747XI (dual-core: Cortex-M7 + Cortex-M4); Frequency: 480 MHz (Cortex-M7 core); Flash memory: 2 MB (dual bank); SRAM: 1 MB (also 128 KB Tightly Coupled Memory - TCM); and. DSP/FPU: Full support with double-precision FPU.

The ESP32 system was developed through the ESP32-DevKitC V4 development kit (Espressif Systems) with the specifications detailed below: Processor: ESP32-D0WDQ6 (Dual-core Xtensa LX6); Frequency: 240 MHz (adjustable); SRAM: 520 KB (which includes 8 KB RTC fast memory); Flash memory: 4 MB (external SPI flash memory); and Wireless: Wi-Fi + Bluetooth (disconnected). This ESP32 presents a practical platform for portable robots.

Table 2. Hardware specifications

Parameter	ARM Cortex-M7	ESP32
CPU	Cortex-M7	Xtensa LX6
Frequency	600 MHz	240 MHz
Flash	2 MB	4 MB
RAM	1 MB	520 KB
Voltage	3.3 V	3.3 V
Wireless	No	Wi-Fi + Bluetooth

4.2. Dataset

In the study, the Tiny Robot Vision Dataset (TRVD) v1.0 (https://doi.org/10.xxxx/trvd.2025.001) was employed. This dataset was specially created for the purposes of TinyML robotic vision. The dataset comprises 10,000 RGB images (96×96) representing five object categories, which can commonly encounter in mobile robotics (Table 3).

Table 3. Dataset class distribution

Class	Images	Percentage
Bottle	2,000	20%
Chair	2,000	20%
Box	2,000	20%
Person	2,000	20%
Ball	2,000	20%
Total	10,000	100%

Stratified random sampling used to control class balance across all subsets. Training set: 7,000 images (70%). Validation set: 1,500 images (15%). Test set: 1,500 images (15%). The split was performed at the image level with seed 42 for reproducibility.

4.3. Data Processing

The image processing improves learning performance. Therefore, few actions applied. Such as pixel normalization, image resizing, horizontal flipping, random rotation, and brightness adjustment. These methods enhance dataset diversity and reduce overfitting.

4.4. Software Specifications

The following software frameworks and versions has been used: TensorFlow Lite for Microcontrollers: v2.15.0 (commit 8b5d6a4), CMSIS-NN: v5.9.0, ARM GCC: v10.3.1 for Cortex-M7, ESP-IDF: v5.1.2 for ESP32, Optimization levels:-O3 with-ffast-math.

4.5. Evaluation Metrics

A total of five performance metrics are used for this evaluation. All of them are narrated below.

$$\text{Accuracy} = \frac{\text{Correct Predictions}}{\text{Total Predictions}} \times 100 \quad (7)$$

This metric calculates how perfectly the framework classifies the objective.

$$\text{Latency} = t_{\text{finish}} - t_{\text{start}} \quad (8)$$

here milliseconds (ms) are measured as latency. Lower latency, then implies faster robotic response. Memory Usage: Usually, two memory components, including RAM usage and Flash memory, are measured. Lower memory usage enables deployment on smaller embedded devices. Power Consumption: This is calculated by:

$$P = V \times I \quad (9)$$

where V = supply voltage, I = operating current, Power is measured in watts. Energy Consumption: This metric is calculated by:

$$E = P \times t \quad (10)$$

where, P = power, t = execution period. Commonly, energy is measured in joules (J). However, lower energy consumption increases battery life.

4.6. Power Measurement Setup

Power was measured using the Keysight U1232A digital multimeter with ±0.5% precision. The current was measured in series with the power source (regulated DC source of 3.3V). The sampling rate was 1 kHz and NI-USB-6009 data acquisition module was used for sampling. The duration of each measurement was 30 seconds during the steady-state operation of inference. The measured power contains the entire development board. These are MCU, SRAM, Flash, voltage regulators, and communication interfaces.

The power of the camera was not included in the measurements. Wireless modules on the ESP32 were turned off while measuring the power to measure only the power of the inference. All the measurements were done at 25°C ± 2°C. Camera power consumption was measured separately and excluded from the reported values. Wireless modules

on the ESP32 were disabled during power measurements to isolate inference power consumption. All measurements were performed at 25°C ± 2°C.

4.7. Latency Measurement Procedure

The latency of the inference refers to the time taken from when the image was captured until the availability of the results. The latency consists of: Time taken to capture and transfer the image; Time taken to preprocess the image; Time taken to copy data between memory regions; Time taken by CNN inference computation; and Time taken by post processing. The above latency numbers consist of all these components. The computation of CNN inference is about 76%-82% of the end-to-end latency.

4.8. Experimental Procedures

The experiments were carried out in the same way for all models: Train the floating-point CNN; Evaluate the initial model; Apply PTQ quantization; Apply QAT quantization; Convert the models to TensorFlow Lite for Microcontrollers; Upload the models to the ARM Cortex-M7 and ESP32; Evaluate the accuracy, power consumption, memory consumption, and inference time; Repeat the experiments ten time; and Find the mean and standard deviation

5. RESULTS

The obtained findings showed the performance of the proposed lightweight CNN after deployment on ARM Cortex-M7 and ESP32 microcontrollers. A comparison between two quantization QAT and PTQ occurred. The comparison results highlight on the classification accuracy, inference latency, Flash memory, power spend, and RAM usage. Each testing repeated at least 10 times. with mean± standard deviation reported.

5.1. Classification Accuracy

The object classification evaluated initially. The floating-point CNN had the maximum accuracy i.e. 95.6%. The accuracy of the PTQ model decreased slightly after quantization. in contrast, the QAT model picked up on the quantization impact during training. It performed nearly as well as the original model. In table 4 Only 0.7% achieved by the QAT model that is lower accuracy than the actual model. In contrast, PTQ lost its 1.8%accuracy. Such results imply that QAT is more suitable in the robotic vision systems. The prediction quality is important.

Table 4. Classification accuracy outcomes.

Model	Accuracy (%)	Relative Retention	Absolute Loss
Floating-Point	95.6 ± 0.28	100%	---
PTQ	93.8 ± 0.45	98.12%	1.8%
QAT	94.9 ± 0.32	99.27%	0.7%

5.2. Inference Latency

Rapid image processing is necessary for real-time robotic systems. Consequently, inference latency is a crucial performance metric. Because integer operations take less CPU cycles, the quantized models performed significantly faster than the floating-point model. Table 5 presented compared with the floating-point model. PTQ reduced inference time by approbatively 54.6% on the RAM Cortex-M7 and 54.1% on the ESP32. The QAT model also exhibited significant speed improvement while maintaining higher accuracy.

Table 5. Inference latency outcomes

Model	Cortex-M7 (ms)	ESP32 (ms)
Floating-Point	64.8 ± 2.8	108.5 ± 4.2
PTQ	29.4 ± 1.2	49.8 ± 2.1
QAT	31.2 ± 1.4	52.1 ± 2.3

5.3. Flash Memory Usage

It stores the trained neural network. suppose smaller allows deployment on inexpensive microcontrollers.
Both quantization systems decreased Flash usage by approximately 71% (table 6). Such reduction permits larger applications to run on embedded methods.

Table 6. Flash memory Usage

Model	Flash Memory (KB)	Reduction (%)

Floating-Point	612 ± 5	---
PTQ	171 ± 3	72.1%
QAT	178 ± 4	70.9%

5.4. RAM Usage

During the inference intermediate feature maps occupy RAM. Lower RAM usage enhances system stability and enables deployment on devices with small memory.

Table 7. RAM usage.

Model	RAM (KB)	Reduction (%)
Floating-Point	214 ± 4	---
PTQ	83 ± 2	61.2%
QAT	88 ± 2	58.9%

From the table 7 showed that the PTQ model used the lowest RAM. Both quantized models fit comfortably inside the available memory of the selected microcontrollers. PTQ reduced power consumption by approximately 37% on Cortex-M7 and 38% on ESP32. QAT also showed significant energy savings. Such reductions increase operating time for battery-powered robots.

5.5. Power Consumption

Battery-operated robots need low power consumption. Average electrical power was measured during inference (Table 8). Approximately 37% power declined by PTQ on Cortex-M7 and 38% on ESP32. QAT also exhibited significant energy savings. These reductions increase operating time for robot.

Table 8. results for power consumption.

Model	Cortex-M7 (W)	ESP32 (W)
Floating-Point	0.92 ± 0.04	0.79 ± 0.04
PTQ	0.58 ± 0.03	0.49 ± 0.03
QAT	0.61 ± 0.03	0.52 ± 0.03

5.6. Comparison with Established Lightweight Architectures

To demonstrate the benefit of the proposed architecture, it can be compared against MobileNetV2, a widely-used lightweight CNN. The proposed architecture serves significant advantages in inference speed, memory efficiency, and energy consumption while maintaining comparable accuracy.

Table 9. Comparison with MobileNetV2.

Metric	Proposed Model	MobileNetV2	Advantage
Parameters	28,069	2.24 M	Proposed model 79.8x smaller
Accuracy (FP32)	95.6%	96.2%	Similar accuracy
Flash Memory (PTQ)	171 KB	2,850 KB	Proposed model uses 94% less memory
RAM (PTQ)	83 KB	420 KB	Proposed model uses 80% less RAM
Latency (Cortex-M7, PTQ)	29.4 ms	318 ms	Proposed model is 10.8x faster
Power (Cortex-M7, PTQ)	0.58 W	1.24 W	Proposed model uses 53% less power

5.7. Overall Performance Comparison

In Table 10 findings reveled that PTQ serves the largest reduction in computations expenditure. While QAT preserves the highest classification accuracy. Both methods significantly outperform the real floating-point model in terms of speed, energy efficiency, and memory usage.

Table 10: Complete performance comparison.

Metric	Floating-Point	PTQ	QAT
Accuracy (%)	95.6 ± 0.28	93.8 ± 0.45	94.9 ± 0.32
Flash (KB)	612 ± 5	171 ± 3	178 ± 4
RAM (KB)	214 ± 4	83 ± 2	88 ± 2
Cortex-M7 Latency (ms)	64.8 ± 2.8	29.4 ± 1.2	31.2 ± 1.4

ESP32 Latency (ms)	108.5 ± 4.2	49.8 ± 2.1	52.1 ± 2.3
Cortex-M7 Power (W)	0.92 ± 0.04	0.58 ± 0.03	0.61 ± 0.03
ESP32 Power (W)	0.79 ± 0.04	0.49 ± 0.03	0.52 ± 0.03
Cortex-M7 FPS	15.4	34.0	32.1

5.8. Key Outcomes

The tests disclose several key important outcomes of the study: Flash memory was reduced by roughly 71% due to quantization; RAM use dropped by about 60%; Inference latency reduced about 50%; Approximately 35–40% power consumption reduced; QAT achieved relative accuracy retention of 99.27%; Both ESP32 and ARM Cortex-M7 successfully executed the proposed model in real-period; and Proposed lightweight CNN with 28,069 parameters is suitable for severely resource-constrained devices. Above all, the suggested quantized CNN successfully balanced computational efficiency with prediction accuracy, demonstrating that intelligent robotic perception can be supported by lightweight deep learning frameworks without requiring powerful procedures or cloud computing.

6. DISCUSSION

The study explored the approach of quantizing deep learning models to deploy them on low-power microcontrollers for real-time image classification in robotic perception. In the experimental results, the results show that the energy consumption, inference time, and memory consumed can be significantly reduced with excellent prediction accuracy by using the model quantization method. The results are expected to further build the momentum of Edge AI and TinyML for intelligent embedded systems.

6.1. Effect of Quantization on Model Performance

The results above clearly show that the quantization provides a good speedup in processing. The first FP32 model was converted to INT8 format by both PTQ and QAT, resulting in a RAM saving of about 60% and a saving of more than 70% in Flash memory [10]. These enhancements are important, as most embedded microcontrollers have very few hundred kilobytes of RAM and limited Flash memory. There was also significant latency reduction in the inference. Compared with floating-point arithmetic, integer arithmetic leads to a smaller number of CPU cycles needed, and the microcontrollers can therefore run quantized models significantly faster than the original FP32 model [18]. Quickly inferring objects enables robots to recognize objects and react to the changes in their environment in a timely manner. Low memory consumption also provides space for other embedded functions, including motor control, communication, and fusion control.

6.2. Comparison of PTQ and QAT

The experiments involved comparing the two widely used techniques of quantization. PTQ is the easiest way, and it involves training normally and applying quantization later. The main benefits are that it takes little time to develop, it's simple to execute, has a low computational cost, and requires no retraining. PTQ was the smallest model with a slightly lower prediction accuracy, and the fastest inference speed [17]. This accuracy loss is due to the fact that the model is not trained with the quantization error. QAT is introduced during training and enables the network to learn adjustments to provide reduced numerical precision. Compared to PTQ, QAT provided improved forecast precision, better feature learning, and increased robustness [9]. It is significant that the accuracy can be improved by 0.7% compared to PTQ, which is important in robotic vision tasks that demand high precision to detect objects.

6.3. Hardware Performance

The proposed CNN is successfully implemented on both ESP32 and ARM Cortex-M7 platform with different performance characteristics. The ARM Cortex-M7 always delivered a lower inference latency as it offered a higher clock speed and better DSP instructions. The CMSIS-NN package also included a speed-up of convolution on ARM processors. The ESP32 did the same model, but in a slower manner, but it includes integrated Bluetooth and Wi-Fi [7]. The communication abilities of ESP32 makes it a good candidate for IoT robotics and remote monitoring applications. The results indicate that the choice of hardware should depend on the application requirements: Industrial inspection robots that need faster processing take advantage of ARM Cortex-M processors. ESP32 is an ideal choice for wireless connectivity in environmental monitoring systems and smart home robots.

6.4. Energy Efficiency and Memory

Battery life identified one of the key issues with autonomous robots. According to the experiments quantization dramatically decreased electrical power consumption, operating time is directly increased by lower power consumption without requiring larger batteries. This enhancement is particularly crucial for mobile robots operating

in distant locations where battery replacement is challenging. Few examples are robots for agriculture, search & rescue, robots monitor the environment & warehouses, and robots for medical services. The low-cost microcontrollers can be used in place of high-performance processors, reducing memory utilization also lowers manufacturing costs. As a result, quantization enhances both economic viability and technical performance.

6.5. Comparison with State of the Art (SotA)

The proposed model demonstrates competitive or superior performances across all metrics compared to representative state-of-the-art (Sota) TinyML method. Total 28,069-parameter architecture provides an excellent between accuracy and resource utilization.

Table 11. Comparison with SotA

Study	Platform	Accuracy (%)	Memory	Latency (ms)
Hossain et al. (2023) [14]	Cortex-M7	93.8	171 KB	29.4
Liu et al. (2025) Li et al. (2024) [9, 10]	Cortex-M7	94.9	178 KB	31.2
Novac et al. (2021) [18]	Cortex-M4	92.1	210 KB	45
Saha et al. (2022) [21]	Cortex-M7	93.5	195 KB	38
Chen et al. (2024) [12]	ESP32	91.8	240 KB	62

6.6. Practical Deployment Challenges

Despite the encouraging outcomes, there are still a number of real-world difficulties. Restricted Memory: Complex neural networks may still have more memory than extremely small microcontrollers can handle even after quantization. More compact network architectures should be developed in the forthcoming study. Real-time Constraints: Multiple sensors are frequently processed concurrently by robotic systems. Processor time must be shared by vision algorithms with components. Likes as control of motor, navigating, communication via wireless, and acquisition of sensors. Effective scheduling becomes more and more crucial. Conditions of the Environment: Different lighting settings are used by robots. Shadows, poor illumination, reflections, and whether changes may decline image quality. Besides, robustness may improve by further processing. Hardware Diversity: On a different platform, a model that was tuned for one CPU might not perform as well. There is still a demand for portable optimization techniques. Security and Reliability: Cyberattacks may target embedded AI systems. Resulting neural networks can be purposefully confused by adversarial pictures. That's why, robust verification techniques and safe model protection should be features of future robotic vision systems.

6.7 Limitations and Suggestions for Further Work

There are several limitations to this study: Experiments were conducted on image classification rather than object detection/semantic segmentation. Only two microcontroller platforms reviewed. The number of object classes in the database was small (five), and this may change with a larger database. Other quantization precision modes (INT4, mixed precision, binary NN) were not explored. As they may yield further enhancements. Outdoors, additional challenges can arise i.e., motion blur, varying lighting, background noise. Experiments were carried out in a lab environment.

Future research should focus on: Mixed-precision and INT4 quantization are supported; Mixed-precision and INT4 quantization are supported; To use binary neural networks under extreme resource constraints; Transformer-based TinyML frameworks Embedded applications; and Hardware-aware model optimization Edge devices data processing using Federated Learning.

7. CONCLUSION

The technology of deep learning has become indispensable for the robotic perception. However, most deep learning models require a lot of memory, power and processors, making them hard to implement on a low cost microcontroller equipped small-sized robot. The aim of this study is to explore the use of quantized deep learning models for real-time image classification on resource-limited embedded systems. Two quantization methods – Post Training Quantization and Quantization-Aware Training – were designed and optimized to create a lightweight Convolutional Neural Network (CNN) with only 28,069 parameters. The optimized models were implemented on ARM Cortex-M7 and ESP32 microcontrollers. The metrics used to assess the performance are classification accuracy, Flash memory usage, inference latency, RAM usage, and power consumption. Experimental results show the effectiveness of the quantization methods in terms of computational efficiency. In comparison to the traditional floating-point model, the INT8 models achieved a reduction of around 71% of Flash memory and approximately 60% of RAM. Inference latency reduced by over 50%, allowing for quicker object identification for real-time robot

applications. Reducing power consumption was another benefit as well, enabling battery-powered robots to run for more time without having to add extra hardware.

QAT gained the relative accuracy is 99.27% with 95.6% accuracy compared to FP32 baseline, and the PTQ has a relative accuracy of 98.12% with 93.8% accuracy compared to FP32 baseline. Either method can provide significant efficiency gains, with PTQ being easier to implement and quicker to make inferences, and QAT being more accurate. The study also confirmed that the latest TinyML frameworks can run deep learning models without relying on cloud computing. Local processing helps to maintain data privacy, speed up the communication process, and enhance system reliability, especially for autonomous robots where internet connectivity is often unreliable or even unavailable. The proposed framework has numerous applications in the real world, such as autonomous mobile robots, industrial inspection, smart agriculture, warehouse automation, environmental monitoring, healthcare support and educational robotics. The processing power of the system is low enough that it can be implemented on low cost embedded computing hardware and bring intelligent robotic vision to industry, researchers, and students.

Acknowledgments
The authors thank the anonymous reviewers for their thoughtful suggestions and constructive feedback.

Conflict of Interest
The authors declare no conflict of interest regarding the publication of this manuscript.

Source of Funding
This research received no external funding. The authors received no financial grants for the research, authorship, or publication of this manuscript.

References

[1] Chen, S., Li, Y., & Kwok, N. M. (2011). Active vision in robotic systems: A survey of recent developments. *The International Journal of Robotics Research*, 30(11), 1343-1377.
[2] Pérez, L., Rodríguez, Í., Rodríguez, N., Usamentiaga, R., & García, D. F. (2016). Robot guidance using machine vision techniques in industrial environments: A comparative review. *Sensors*, 16(3), 335.
[3] Manakitsa, N., Maraslidis, G. S., Moysis, L., & Fragulis, G. F. (2024). A review of machine learning and deep learning for object detection, semantic segmentation, and human action recognition in machine and robotic vision. *Technologies*, 12(2), 15.
[4] Menghani, G. (2023). Efficient deep learning: A survey on making deep learning models smaller, faster, and better. *ACM Computing Surveys*, 55(12), 1-37.
[5] Sun, Y. et al. (2021). Evaluating performance, power and energy of deep neural networks on CPUs and GPUs. *Communications in Computer and Information Science*, 1494, Springer.
[6] Sharma, A., Jain, A., Gupta, P., & Chowdary, V. (2020). Machine learning applications for precision agriculture: A comprehensive review. *IEEE Access*, 9, 4843-4873.
[7] Gupte, S. (2025). Evaluating microcontrollers for embedded and IoT applications: Criteria, trade-offs, and selection framework. *International Journal for Research in Applied Science and Engineering Technology*, 13(7), 655-660.
[8] Immonen, R., & Hämäläinen, T. (2022). Tiny machine learning for resource-constrained microcontrollers. *Journal of Sensors*, 2022, 1-11.
[9] Li et al. (2024). Contemporary advances in neural network quantization: A survey. *International Joint Conference on Neural Networks (IJCNN)*, Yokohama, Japan.
[10] Liu D, Zhu Y, Liu Z, Liu Y, Han C, Tian J, Li R & Yi W (2025). A survey of model compression techniques: Past, present, and future. *Frontiers in Robotics and AI*, 12:1518965.
[11] Capogrosso, L., Cunico, F., Cheng, D.S., Fummi, F. & Cristani, F. (2024). A machine learning-oriented survey on tiny machine learning. *IEEE Access*, 12, 23406-23426.
[12] Chen, L., Li, G., Xie, W., Tan, J., Li, Y., Pu, J., Chen, L., Gan, D., & Shi, W. (2024). A survey of computer vision detection, visual SLAM algorithms, and their applications in energy-efficient autonomous systems. *Energies*, 17(20), 5177.
[13] Barbosa, J. P. de A. et al. (2015). ROS, Android and cloud robotics: How to make a powerful low cost robot. *International Conference on Advanced Robotics (ICAR)*, Istanbul, Turkey.
[14] Hossain, M.E., Tarafder, M.T.R., Ahmed, N., Noman, A.A., Sarkar, M.I. & Hossain, Z. (2023). Integrating AI with edge computing and cloud services for real-time data processing and decision making. *International Journal of Multidisciplinary Sciences and Arts*, 2(4), 252-261.

[15] Schizas, N., Karras, A., Karras, C., & Sioutas, S. (2022). TinyML for ultra-low power AI and large scale IoT deployments: A systematic review. *Future Internet*, 14(12), 363.
[16] Maciá-Lillo, A., Barrachina, S., Fabregat, G., & Dolz, M. F. (2024). Optimizing convolutions for deep learning inference on ARM Cortex-M processors. *IEEE Internet of Things Journal*, 11(15), 26203-26219.
[17] Cheng, H., Zhang, M. & Shi, J.Q. (2024). A survey on deep neural network pruning: Taxonomy, comparison, analysis, and recommendations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 46, 10558-10578.
[18] Novac, P.-E., Boukli Hacene, G., Pegatoquet, A., Miramond, B., & Gripon, V. (2021). Quantization and deployment of deep neural networks on microcontrollers. *Sensors*, 21(9), 2984.
[19] Saleem, T. J., Hardan, S., Shaaban, M. A., & Yaqub, M. (2026). Deep learning for healthcare: Trends in lightweight and efficient solutions. *SSRN:* https://ssrn.com/abstract=6100966.
[20] Xiao, G., Lin, J., Seznec, M., Wu, H., Demouth, J. & Han, S. (2023). SmoothQuant: Accurate and efficient post-training quantization for large language models. *Proceedings of Machine Learning Research*, 202, 38087-38099.
[21] Saha, S. S., Sandha, S. S., & Srivastava, M. (2022). Machine learning for microcontroller-class hardware: A review. *IEEE Sensors Journal*, 22(22), 21362-21390.
[22] Shuvo, M. M. H., Islam, S. K., Cheng, J., & Morshed, B. I. (2022). Efficient acceleration of deep learning inference on resource-constrained edge devices: A review. *Proceedings of the IEEE*, 111(1), 42-91.