Proceedings of CONF-MLA 2026 Symposium: Intelligent Systems and Automation: AI Models, IoT, and Robotic Algorithms
DOI: 10.54254/2755-2721/2026.LD37676

Adaptability of Lightweight AI Models for Embedded Deployment: A Case Study of an Agricultural Monitoring Robot

Jingen Zhou

Cardiff Sixth Form College, Cardiff, UK
jingen.zhou@ccoex.com

Abstract. Many artificial intelligence programs require powerful computing hardware, whereas agricultural robots, drones, and smart cameras rely on small embedded devices with limited computing power, battery capacity, and heat dissipation. Lightweight models reduce parameters and computational costs while maintaining acceptable accuracy, enabling AI deployment on resource-constraint network devices. This paper reviews key lightweighting methods, including compact network architectures, model pruning, model quantization, and knowledge distillation. It compares common platforms such as Jetson, Raspberry Pi, and microcontrollers. Using an agricultural monitoring robot as an example, this study proposes a system combining a camera, a lightweight YOLO model, Jetson, and a microcontroller for real-time crop monitoring. Experiments are designed to evaluate accuracy, speed, power consumption, temperature, and battery life. Rather than pursuing maximum model complexity, this paper focuses on explaining how to select a model that is truly suitable for the actual device. For agricultural robots, effective deployment requires sufficient accuracy, real-time performance, and low energy consumption.

Keywords: Lightweight model, embedded system, YOLO, Jetson, agricultural robot

1. Introduction

Artificial intelligence (AI) has been applied to devices such as facial-recognition cameras, autonomous vehicles, drones, and robots. In the past, many AI tasks required data to be uploaded to serve for processing. Although servers provide strong computing capabilities, this method depends on network speed and stability. If a farm lacks a reliable wireless network, an agricultural robot may not receive recognition results in time. Running AI models directly on devices can reduce network latency, reduce the need to continuously upload images to the Internet, and improve data protection. However, embedded devices have limited computing power, memory, battery capacity, and heat-dissipation capability. Therefore, large models usually cannot be deployed directly and must be simplified and optimized.

Existing studies have proposed several solutions for resource-constrained hardware. MobileNet uses depthwise separable convolutions to reduce computational costs, Efficient Net improves efficiency by jointly scaling network depth, width, and input resolution, and YOLO performs object

© 2026 The Authors. This is an open access article distributed under the terms of the Creative Commons Attribution License 4.0 (https://creativecommons.org/licenses/by/4.0/).

DOI: 10.54254/2755-2721/2026.LD37676

localization and classification within a unified framework, making it suitable for real-time detection [1-3]. In addition, pruning and quantization can reduce model size and computational requirements, while knowledge distillation uses a large model to guide the training of a smaller model [4, 5]. However, many studies mainly compare offline accuracy or single-inference speed and rarely consider power consumption, latency, temperature, and long-term operating stability together.

This paper addresses three questions: what are the common methods for lightweighting AI models, what tasks are suitable for different embedded platforms, and how can lightweight models be deployed in agricultural monitoring robots and evaluated for practical suitablility? Rather than pursuing model size or accuracy alone, this study focuses on balancing accuracy, speed, power consumption, and stability. It aims to provide guidance for model selection in smart agricultural equipment and practical edge-AI deployment.

2. Main methods for lightweight models

2.1. Lightweight network architecture design

The most direct way to make a model lightweight is to reduce the size of the network during the model-design stage. For example, MobileNet, EfficientNet, and the YOLO nano series reduce the number of network layers, channels, or repeated calculations, making model files smaller and inference faster [1-3]. In agricultural robot applications, YOLO nano-scale models are relatively suitable. YOLO can provide both the locations and classes of objects in one image, and its training, export, and deployment tools are relatively complete. Compared with larger YOLO models, nano models may have slightly lower accuracy, but they are more suitable for edge-computing platforms such as Jetson.

2.2. Model pruning

Model pruning is a method that reduces model complexity by removing unimportant parameters from a neural network [4]. A trained model may contain redundant parameters or parameters that have little effect. After these parameters are removed, the model file becomes smaller and the computational load may also decrease. Pruning is generally divided into unstructured pruning and structured pruning. Unstructured pruning removes individual weights and may achieve a high compression ratio, but it does not necessarily improve inference speed directly on ordinary hardware. Structured pruning removes complete channels or convolution kernels and is more likely to provide real acceleration on embedded devices.

It should be noted that excessive pruning may reduce a model's ability to extract features. Farmland images often contain small objects, leaf occlusion, and complex backgrounds, so removing too many parameters may cause missed detections. Therefore, practical applications usually use gradual pruning. Only a small proportion of parameters is removed each time, after which the model is retrained and an independent test set is used to check whether accuracy has decreased significantly.

2.3. Model quantization

Model quantization is a method for reducing the computational precision of a model and is also a common technique in the model compression process [4]. Traditional deep-learning models usually use FP32 calculations, while quantization can convert some calculations to FP16 or INT8. Reducing

DOI: 10.54254/2755-2721/2026.LD37676

data can lower memory usage and may improve computational efficiency, while also reducing graphics-memory and main-memory usage. On Jetson, FP16 is usually relatively easy to use and causes relatively little accuracy loss, so it can be used as the first optimization method. INT8 may be faster, but it requires calibration using representative images from real farmland. If the calibration images include only scenes with good lighting, the model may perform poorly in shadows, strong sunlight, or evening conditions. Table 1 compares the main characteristics of the three computational precisions.

Table 1. Comparison of the characteristics of different quantization precisions

Quantization Method	Data Precision	Main Advantage	Point to Note
FP32	32-bit floating point	Stable accuracy and convenient training	High computational cost and memory usage
FP16	16-bit floating point	Simple deployment and relatively fast speed	Small accuracy changes need to be checked
INT8	8-bit integer	Fast, with potentially lower power consumption	Requires calibration data, accuracy may decrease

2.4. Knowledge distillation

Knowledge distillation is a method that uses a large model to assist the training of a small model [5]. The large model can be regarded as the "teacher model," while the small model can be regarded as the "student model." The student model learns not only the correct labels of images, but also the class probabilities and decision patterns produced by the teacher model. In this way, the small model may achieve better recognition performance without increasing its number of parameters.

The advantage of this method is that the student model does not require additional computation during actual operation. However, the training process is more complex because both a teacher model and a student model must be prepared. For this project, knowledge distillation is a method that can be studied further in the future and is not an essential part of the initial stage.

2.5. Comparison of four lightweight methods and the selection strategy

Table 2. Comparison of four lightweight methods

Method	Description	Advantage	Problem
Small Network	Directly select a smaller model	Easiest way to begin	Accuracy may be slightly lower
Pruning	Remove unimportant parameters or channels	Can reduce model size	Excessive pruning may cause missed detections
Quantization	Convert FP32 to FP16 or INT8	Fast and uses less storage	Accuracy needs to be rechecked
Knowledge Distillation	A large model helps train a small model	The small model may perform better	The training process is more complex

DOI: 10.54254/2755-2721/2026.LD37676

As shown in Table 2, different lightweight methods have different characteristics. During actual deployment, it is not recommended to use all optimization methods at the same time. Instead, a step-by-step optimization strategy should be adopted: first train a baseline YOLO nano model, then export it to ONNX and test FP16. If the speed is still insufficient, consider INT8. Only when the model is still too large or its accuracy is insufficient should pruning or knowledge distillation be attempted. This sequence makes it easier to determine the changes caused by each optimization step and avoids changing too many factors at the same time, which would make it difficult to identify the source of a problem.

3. Common embedded platforms and the AI model deployment process

3.1. Common embedded hardware platforms

The Jetson series features an integrated GPU and is suitable for running image-recognition and object-detection models. The Jetson software platform provides systems, libraries, and tools for edge-AI development [6]. Because it supports CUDA and TensorRT, many YOLO models can run on this platform after conversion. However, compared with general-purpose single-board computers, Jetson devices have higher prices and power consumption, and they usually require a heat sink or a fan. Raspberry Pi has a relatively low cost and many interfaces, making it suitable for learning programming, connecting sensors, and completing simple control tasks. Its official documentation also provides information about support for cameras and AI expansion hardware [7]. It may be sufficient for low-frequency image classification, but continuous object detection is usually slow when it relies only on the CPU. STM32 and ESP32 are microcontrollers. They have low power consumption and good real-time control capabilities, making them more suitable for reading sensors and controlling motors. Taking STM32 as an example, official edge-AI tools can perform model analysis, optimization, and code generation. However, microcontrollers have limited memory and usually cannot run ordinary YOLO models directly [8].

3.2. Model deployment process

A model is usually trained on a computer first. After training is complete, it can be exported to ONNX format. ONNX provides a general model representation that makes conversion between different frameworks and operating environments easier [9]. TensorRT can then be used to build an optimized inference engine for an NVIDIA GPU, with FP32, FP16, or INT8 precision selected according to the hardware [10]. The entire process can be expressed as follows: prepare data - train the model - export to ONNX - convert with TensorRT - test on Jetson. Completing the conversion does not mean that deployment has succeeded. First, the recognition results from the computer model, the ONNX model, and the TensorRT model must be compared using the same group of images to check whether the predicted classes, confidence scores, and object locations are consistent. The camera should then be connected to test FPS and latency in continuous video. Finally, the system must run for a long time to observe whether its inference speed decreases as the temperature rises.

3.3. Dataset preparation and annotation

Agricultural environments change greatly, so a dataset cannot contain only a small number of well-lit ideal images. Training images should cover sunny weather, cloudy weather, shadows, strong light,

DOI: 10.54254/2755-2721/2026.LD37676

different camera angles, different stages of crop growth, and different ground backgrounds. To teach the robot to detect obstacles, the dataset should also include stones, tools, people, other vehicles, and partially occluded objects.

Table 3. Dataset splitting strategy

Data Subset	Recommended Proportion	Purpose
Training Set	About 70%	Train model parameters
Validation Set	About 20%	Select the model and adjust parameters
Test Set	About 10%	Final fair evaluation

After data collection is complete, manual annotation is required. This means drawing boxes around objects and assigning the correct classes. Annotation standards should remain consistent. For example, weeds must not be annotated as individual plants in some images and as a small area in other images. After annotation is complete, the data can be divided into training, validation, and test sets. The training set is used for learning, the validation set is used to adjust the model, and the test set is used only for the final comparison and must not be repeatedly used to select parameters. Table 3 shows the data-splitting strategy used in this project.

3.4. Points to note during training and model conversion

Before training, image sizes should be standardized, and incorrect labels and duplicate images should be checked. During training, overall accuracy is not the only metric that should be examined, the recall of each class should also be observed. For example, if the model can correctly identify crops but often misses obstacles, the system cannot be said to meet the safety requirements of the robot. Model conversion should preferably be carried out after the baseline model has been trained and validated stably. The FP32 version should first be saved as a reference, after which FP16 and INT8 versions can be produced. INT8 calibration images should be selected from the real training environment and should include different lighting conditions. After each conversion, the version name, conversion settings, and test date must be saved to avoid confusion about which model version was used later.

4. Agricultural monitoring robot system design

4.1. Main tasks

The agricultural monitoring robot designed in this paper is mainly intended for farmland inspection tasks. It can move between crops, observe the environment ahead through a camera, and use an artificial intelligence model to identify crops, weeds, abnormal leaves, and obstacles. Recognition results can be saved as records or sent to the control system to help staff decide whether further inspection, weeding, or spraying is required.

The goal of this system is not to replace human operation completely, but to reduce repetitive inspection work. Manual inspection usually takes a considerable amount of time, and staff may become tired after observing for a long period. The robot can repeatedly inspect a fixed route and mark suspicious locations, after which staff can make further judgments and take action.

DOI: 10.54254/2755-2721/2026.LD37676

4.2. System components

The agricultural monitoring robot mainly consists of a visual-data acquisition module, an edge-computing module, a motion-control module, a sensor module, a power module, and a communication module. The main function of each module and its response to a fault are shown in Table 4. Jetson is responsible for running the YOLO model, while the microcontroller is responsible for motor control and safety-related functions. Separating computation from control can reduce the effect of artificial intelligence model failures on the movement safety of the robot.

Table 4. Main modules of the agricultural monitoring robot and fault handling

Module	Main Function	Response to a Fault
RGB Camera	Capture farmland images	Stop high-risk actions and prompt an inspection
Jetson	Run the object-detection model	The microcontroller commands the robot to slow down or stop
Microcontroller	Motor and safety control	The hardware emergency stop cuts off the drive output
Distance Sensor	Detect nearby obstacles	Slow down or stop immediately
Wireless Module	Upload records and receive tasks	Save data offline and continue safe patrol

4.3. Model selection and operating modes

The system first selects a YOLO nano model and trains it using its farmland images collected for the project. The robot has three operating modes. Energy-saving mode is suitable for normal patrol and uses a lower resolution and fewer detections. Standard mode is used for normal monitoring. Fine-inspection mode increases the resolution and performs another check after a suspicious object has been found.

When the battery level is low or the Jetson temperature is too high, the system automatically reduces the detection frequency and movement speed. Actions that may cause damage, such as spraying or cutting, also require human confirmation or confirmation from another sensor and must not be carried out directly based on a single visual-recognition result.

4.4. Coordination between vision and control

The vision system outputs the object class, location, and confidence score, and the control system makes corresponding decisions based on this information. For example, when an obstacle is detected ahead, the robot first reduce its speed and then uses a distance sensor for confirmation. When a possibly abnormal leaf is detected, the robot records its location and switches to fine-inspection mode. If the vision system fails to return a result within the required time, the controller should prevent the robot from continuing to move at high speed.

4.5. Power management and battery-life design

The agricultural robot is powered by a battery that supplies energy to the Jetson, camera, motors, communication module, and sensors. Motor setup and operation on slopes may cause high current

DOI: 10.54254/2755-2721/2026.LD37676

demand, so the battery life of the entire robot cannot be estimated based only on the computing module's power consumption. A more reasonable method is to measure total power consumption separately under three conditions: stationary recognition, low-speed patrol, and movement under load.

To extend operating time, the system can automatically adjust its operating mode according to the task. During normal patrol, it is unnecessary to process every video frame, so the detection frequency can be reduced. When the robot approaches a target area, the detection frequency can be increased. When the robot stops and waits, the GPU frequency can be reduced or unnecessary modules can be turned off.

4.6. Data updates and model maintenance

When operating on different farms, the robot may encounter situations that are not included in the original training data. The system can save low-confidence images, images corrected by users, and images containing obvious recognition errors. These images can then be manually viewed and added to the training dataset. Compared with randomly saving all images, this approach reduces storage requirements and makes it easier to identify situations in which the model performs poorly.

Before farmers' data is used, appropriate consent should be obtained, and users should be informed about what data will be stored, how long it will be retained, and how it will be used. A new model should not be deployed to all robots simultaneously. It can first be tested on device and then gradually deployed to other devices. The previous model should also be retained so that the system can return to it if the new version causes significant problems.

4.7. Safety design

Because AI models can make incorrect predictions, safety should not depend entirely on the AI system. The robot should include an emergency-stop button, distance sensors, speed limits, and an independent low-level controller. If the vision system stops responding, the microcontroller should slow down or stop the robot. If communication is interrupted, the robot should enter a predefined safe mode rather than continue an uncertain task.

For important actions, two-stage confirmation can be used to reduce the risk of incorrect decisions. For example, after the vision model detects an obstacle, the robot first reduces its speed and then uses the distance sensor for confirmation. When an area requiring spraying is identified, the robot first records its location and performs the action only after human confirmation. Although this approach reduces the level of automation, it is more appropriate for early-stage prototype testing.

5. Experimental design and performance evaluation

5.1. Experimental objective

The purpose of the experiment is to compare different models and quantization methods and identify the solution most suitable for the agricultural robot. All models should be evaluated using the same test images, the same video, and the same Jetson device. Camera resolution, environmental temperature, and power settings should also remain as consistent as possible so that the comparison is fair.

DOI: 10.54254/2755-2721/2026.LD37676

5.2. Main performance metrics

The experiment compares the FP32, FP16, and INT8 versions of the YOLO nano model and evaluates detection accuracy, recall, FPS, single-inference latency, power consumption, temperature, and continuous-operation stability. Detection accuracy shows whether the model can correctly recognize targets, while recall reflects the likelihood of missing targets. FPS and latency show whether the system can respond in time, whereas power consumption and temperature directly affect battery life and operational stability. Table 5 presents the recommended experimental targets and the reasons for setting them.

Table 5. Experimental performance metrics and recommended targets

Metric	Recommended Target	Reason
Detection Speed	About 15 FPS or higher	Meet the real-time needs of low-speed inspection
Recall for Important Targets	Aim for above 90%	Reduce missed obstacles
Single-Inference Latency	Aim for below 100 ms	Shorten the time from detecting a target to outputting a result
Continuous Operation	At least 30 minutes	Observe whether temperature and speed remain stable
Total System Power Consumption	As low as possible after meeting performance requirements	Extend battery operating time

The values in Table 5 are project targets rather than actual measured results. After the formal experiment is completed, they must be replaced with actual measured data, and expected values must not be written as experimental conclusions.

5.3. Experimental steps

The experimental procedure mainly includes the following steps:

(1) Prepare and divide the training, validation, and test images, and check for incorrect labels.

(2) Train the FP32 baseline YOLO nano model on a computer and save the training settings.

(3) Export the model to ONNX and then convert it to FP16 and INT8 versions.

(4) Test with the same video on the same Jetson, keeping the resolution and camera position fixed.

(5) Record accuracy, recall, FPS, latency, power consumption, temperature, and graphics-memory usage separately.

(6) Run continuously for at least 30 minutes and observe whether FPS decreases as the temperature rises.

(7) Compare the results and select the solution that meets the minimum accuracy and speed requirements while having relatively low power consumption.

DOI: 10.54254/2755-2721/2026.LD37676

5.4. Method for selecting results

A model cannot be selected based a single metric. A model with high accuracy but very low speed is unsuitable for a real-time robot, while a fast model that frequently misses obstacles is also inappropriate. Therefore, the minimum requirements for detection accuracy, recall of important classes, and detection speed should be established first. Power consumption and temperature can then be compared among models that meet these requirements. FP16 can be used as the initial optimization method, but whether INT8 provides better overall performance should be determined based on the actual results.

6. Problems in practical applications and directions for improvement

Agricultural environments are highly variable. Different seasons, weather conditions, crop types, and camera positions can affect image quality, meaning that a model trained on one farm may not perform equally well on another. A practical solution is to continuously collect real data, especially images from incorrect or low-confidence detections, and regularly update the training data and model. In addition, Jetson produces heat when running object detection. If heat dissipation is poor, the device may automatically reduce its operating frequency, causing FPS to decrease. A robot will also encounter dust, mud and water, vibration, and high temperatures in farmland. Therefore, the enclosure must provide appropriate ventilation, dust and water protection, and secure mounting. Even a small change in camera position may affect recognition performance.

In addition to technical problems, the cost, maintenance requirements, terrain, and ease of operation should also be considered when evaluating practical deployment. A high-performance computing module can increase speed, but it also increases cost and power consumption. For a small farm, it may be more practical to reduce the number of functions and retain only inspection and recording. Larger farms may require more powerful computing module and automatic path planning. Economic value should therefore be evaluated not only by the initial hardware cost, but also by battery replacement, maintenance, model updates, and operator training. The robot is more likely to be suitable for wider deployment if it can reduce repeated inspection time, identify problems earlier, and reduce losses.

In the future, agricultural robots could use lower power NPUs and incorporate depth cameras, thermal-imaging cameras, soil sensors, and more accurate positioning modules. Multiple sensors can reduce incorrect judgments caused by relying only on a camera, but they also increase cost and software complexity. Data sharing and model updates among multiple robots also have development potential. However, data sharing must have the farmer's consent, and specific location and production information must be protected. A model update must also be tested on a small scale first and must not be put into actual use simply because its offline accuracy has improved.

7. Conclusion

This paper analyzed the adaptability of lightweight AI models to embedded devices, using an agricultural monitoring robot as a case study. The analysis shows that lightweight networks, model pruning, model quantization, and knowledge distillation can reduce computational, storage, and energy requirements, but may also lead to lower accuracy, missed detections, or a more complex training process. Therefore, model lightweighting should not be evaluated only by model-file size. Accuracy, recall for important classes, FPS, latency, power consumption, temperature, and continuous-operation stability should be compared together on the actual device.

DOI: 10.54254/2755-2721/2026.LD37676

For the system design, this paper uses a camera, Jetson, a YOLO nano model, and a microcontroller to form an agricultural monitoring robot. Jetson performs image recognition, while the microcontroller handles motor control and independent safety functions. The model can be trained on a computer and deployed to Jetson through ONNX and TensorRT. Energy-saving, standard, and fine-inspection can further adjust detection frequency according to task difficulty, battery level, and device temperature, thereby balancing real-time performance and battery life.

This paper also has clear limitations. The values currently presented in the experiment tables are planned targets rather than measured results, and long-term testing in real farmland has not yet been completed. The dataset is also limited in terms of size, crop types, and environmental conditions, so the proposed system cannot yet be considered reliable across different farms. Future experiments should compare FP32, FP16, and INT8 versions on the same device and test continuous operation for at least 30 minutes. Conditions such as shadows, strong light, occlusion, mud and water, and camera vibration should also be added to examine model recall and stability under environmental changes.

Overall, the model most suitable for an agricultural robot is not necessarily the largest, fastest, or most accurate in a single test. It should provide sufficiently stable recognition under limited hardware resources, respond within an acceptable time, maintain reasonable power consumption, and allow the robot to enter a safe state when failures occur. Future research can further compare different NPU platforms, add depth cameras and soil sensors, and use data from multiple farms to verify model generalization. This paper demonstrates that practical edge-AI deployment should be guided by overall system requirements rather than by an isolated accuracy metric.

References

[1] Howard, A. G., Zhu, M., Chen, B., et al. (2017). MobileNets: Efficient convolutional neural networks for mobile vision applications. arXiv:1704.04861.
[2] Tan, M., & Le, Q. V. (2019). EfficientNet: Rethinking model scaling for convolutional neural networks. In *Proceedings of the 36th International Conference on Machine Learning* (pp. 6105–6114).
[3] Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. In *2016 IEEE Conference on Computer Vision and Pattern Recognition* (pp. 779–788).
[4] Han, S., Mao, H., & Dally, W. J. (2016). Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding. In *International Conference on Learning Representations*.
[5] Hinton, G., Vinyals, O., & Dean, J. (2015). Distilling the knowledge in a neural network. arXiv:1503.02531.
[6] NVIDIA. (n.d.). Jetson software documentation. *NVIDIA*. https://docs.nvidia.com/jetson/
[7] Raspberry Pi Ltd. (n.d.). Raspberry Pi documentation. *Raspberry Pi*. https://www.raspberrypi.com/documentation/
[8] STMicroelectronics. (n.d.). STM32Cube AI Studio documentation. *STMicroelectronics*. https://wiki.st.com/stm32mcu/wiki/AI:STM32Cube_AI_Studio_documentation
[9] ONNX. (n.d.). ONNX documentation. *ONNX*. https://onnx.ai/onnx/
[10] NVIDIA. (n.d.). TensorRT documentation. *NVIDIA*. https://docs.nvidia.com/deeplearning/tensorrt/latest/