



REAL-TIME PEST DETECTION USING DEEP LEARNING ON EDGE DEVICES FOR PRECISION FARMING

Sarala Patchala¹, Sumalatha Pilli², G Usandra Babu³, Shobana Gorintla⁴, Garga Srilakshmi⁵, Swathi Voddi⁶, M Praveen Kumar⁷, Inakoti Ramesh Raja⁸, Chavvakula Janaki Devi⁹

^{1,2}Associate Professor, Department of Electronics and Communication Engineering, KKR & KSR Institute of Technology and Science, Guntur, India.

³Associate Professor, Department of Electronics and Communication Engineering, Aditya University, Surampalem, Kakinada, India.

⁴Assistant Professor, Department of Computer Science and Engineering, NRI Institute of Technology, Pothavarapadu, Agiripalli mandal, Eluru Dist, Andhra Pradesh, India.

⁵Associate Professor, Department of Electronics and Communication Engineering, Aditya University, Surampalem, Kakinada, India.

⁶Assistant Professor, Department of ECE, Koneru Lakshmaiah Education Foundation, Vaddeswaram, AP, India.

⁷Assistant Professor, Department of CSE, R V R J C College of Engineering, Guntur, India.

^{8,9}Associate Professor, Department of Electronics and Communication Engineering, Aditya University, Surampalem, India.

¹<http://orcid.org/0000-0002-5184-0814>, ²<https://orcid.org/0009-0006-7155-0933>, ³<http://orcid.org/0000-0002-8859-2041>

⁴<https://orcid.org/0000-0002-9357-6142>, ⁵<http://orcid.org/0009-0005-5254-7766>, ⁶<http://orcid.org/0000-0002-5269-1873>

⁷<http://orcid.org/0000-0002-6098-748X>, ⁸<http://orcid.org/0009-0006-6806-2605>, ⁹<http://orcid.org/0009-0004-2916-0363>

Email: *saralajntuk@gmail.com, psumaece@gmail.com, usandrababu.g@gmail.com, drgshobana@gmail.com, srilakshmi1853@gmail.com, swathiklu23@gmail.com, praveenkumarmuvva@gmail.com, inakoti.rameshrama@gmail.com, janaki.chavvakula04@gmail.com

ARTICLE INFO

Article History

Received: February 5, 2026

Reviewed: July 31, 2026

Accepted: August 31, 2026

Published: September 30, 2026

Keywords:

Precision,
Deep Learning,
Farming,
Pest Detection

ABSTRACT

Precision agriculture needs better pest monitoring. Current traps send many images to the cloud. This uses much energy and needs human review. The paper presents a smart trap that runs deep learning on the node. The trap uses a Raspberry Pi and an Intel Neural Compute Stick. It captures images inside pheromone traps. It runs neural networks to detect codling moths. Three models were trained and compared: LeNet-5, VGG16 and MobileNetV2. The system selects models that balance accuracy and power use. The trap only sends small alerts not full images. This reduces data transfer and power needs. The device uses a solar harvester and a battery. Energy harvesting lets the trap run for long periods. The system works without human intervention. Training used 4400 images and augmentation steps. The models reached high accuracy in tests. VGG16 showed the highest precision. LeNet gave the best energy trade-off on Raspberry Pi3. The detection pipeline uses sliding windows and ROI extraction. Non-maximum suppression filters overlapping detections. Preprocessing steps include color correction and edge extraction. Training used pruning to reduce model size and keep accuracy. Models were optimized with node merging and batch normalization. The design supports low bitrate radios like LoRa. This enables wide area deployment across orchards. The approach helps reduce pesticide use and save resources. It speeds pest detection and farmer response. It lowers labor and inspection costs. The work combines embedded hardware, edge AI and power management. It moves pest monitoring toward automated, energy-neutral systems.



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).

1. INTRODUCTION

Agriculture uses large amounts of water and contributes to greenhouse gas emissions [1]. Farmers rely on pesticides to protect crops, but this harms pollinators and the environment. There is a clear need to reduce chemical use and move toward smarter farming. Pest monitoring is central to this objective [2]. Traditional pheromone traps capture insects, but the current process requires human experts to inspect images and decide on actions. This is slow, costly and dependent on transmitting large image files to the cloud [3].

Such data transfer consumes energy and bandwidth, which is not ideal for rural orchards. Edge-based pest detection is proposed as a solution. Edge AI processes images directly on the device [4]. Only alerts or detection results are transmitted, not full images. This reduces energy use, lowers costs and allows wider deployment. The system needs to be low power, affordable, accurate and able to run without human support for long periods [5]. To achieve this, a smart trap is created that combines hardware, software and power management. The trap uses a Raspberry Pi with a Neural Compute Stick to process images inside pheromone traps [6]. It runs deep neural networks to detect codling moths. Accuracy and power requirements are balanced to select the best model for embedded use. A solar harvester and battery enable long-term unattended operation [7]. The trap performs inference on the node, reducing data transmission to a few bytes per alert. The detection pipeline involves preprocessing, sliding windows, ROI extraction and filtering [8]. Optimizations like pruning and batch normalization reduce model size while keeping accuracy. Energy tests confirm that inference consumes the most power, but Raspberry Pi3 running LeNet offers the best trade-off. The system is designed to check traps twice a day, matching farmer needs and reducing pesticide waste [9]. Solar energy supports sustainability, even in modest light conditions [10]. Field results show the device operates for months without intervention. Farmers face increasing pressure to produce more food with fewer resources [11]. Climate change, soil degradation and biodiversity loss increase this pressure. Smart agriculture uses data, sensors and AI to optimize every step in crop management [12]. Pest monitoring is one of the most suitable applications for these systems because pests cause sudden and large yield losses. Early detection is critical, but human monitoring is slow and cannot scale [13]. By replacing human observation with automated traps, farmers gain faster warnings and act precisely when needed. This reduces the overuse of pesticides, which benefits both the farm economy and the environment [14]. Sending large images from many traps to the cloud is impractical. Edge AI solves this by limiting communication to lightweight alerts [15]. In this way, the solution runs even on low-bitrate networks like LoRa. This makes it highly scalable to cover large orchards. The key contributions of this paper are:

- To design a self-sustained smart IoT trap capable of running DNN inference on the node while using energy harvesting for extended lifetime.
- To analyze and deploy models at the edge by training and comparing LeNet-5, VGG16 and MobileNetV2 in terms of accuracy and power consumption, highlighting trade-offs for embedded deployment.
- To assess sustainability by measuring energy per task, analyzing battery lifetime and recharge time and demonstrating indefinite operation in real field tests.

Another important part of the introduction is sustainability. Running hundreds of traps would not be feasible if frequent battery changes were required [16]. By using energy harvesting, the devices recharge themselves and keep running for years. This reduces maintenance costs and allows farmers to deploy and forget the system [17]. The introduction makes a strong case that edge-based pest detection is not only technically feasible but socially and economically valuable. It is a clear step toward future farming systems that are more efficient, environmentally friendly and resilient.

II. BACKGROUND WORK

The field has evolved from manual inspection to machine learning and now to deep learning on the edge. Early pest monitoring relied on pheromone traps. Images were captured, but human experts had to count insects [18], [19]. This method was slow, costly and required heavy human effort. Later, researchers began using image analysis. Some works used lab-prepared insect specimens to create datasets [20], [21]. These images were neat but did not represent real trap conditions. More recent works used images from real field traps, which better matched noisy environments [22], [23]. Feature-based approaches were once common. Researchers extracted shapes, textures or wing patterns to classify species [24], [25]. Classifiers like SVMs, k-nearest neighbors and shallow neural networks were then applied [26], [27]. These worked for small datasets but failed in complex real-world images. Deep learning improved accuracy. CNNs removed the need for manual feature design. Studies showed CNN-based pipelines outperform older methods [28], [29]. PestNet is an example of a deep network designed for multi-class pest detection [23]. Other works tested detectors like Faster R-CNN for localization in trap images [30], [31]. These methods allowed both detection and counting. Another important theme is data augmentation. Since datasets are small, augmentation and semi-automatic labeling were used to improve training [32]. Energy and hardware issues appear in recent studies. Some explored low-power cameras, edge accelerators and battery-driven traps [33], [34]. These works show that accuracy alone is not enough. Sustainable, long-term deployment is vital. The reviewed papers underline a gap: few combine deep learning, energy harvesting and real field operation in one system. The current work addresses that gap by designing a complete edge-based solution.

III. SYSTEM ARCHITECTURE

The system architecture describes the end-to-end operation of the smart trap. It is designed to operate in orchards and similar fields with continuous pest monitoring. The device captures images inside a pheromone cup and processes them directly at the edge. Instead of sending large image files to the cloud, it only transmits short alerts using a LoRa radio. The entire system is powered by a small solar panel and a rechargeable battery. An ultra-low power microcontroller unit (MCU) supervises energy use and wakes up the main processing board only when needed. A dedicated neural accelerator is used to make deep learning inference faster. This arrangement provides a balance between accuracy, latency and energy efficiency which is vital for long-term field deployment. The architecture is divided into four functional blocks: sensing, processing, transmission and power. The sensing block is responsible for image acquisition. The Figure 1 shows a workflow for a detection pipeline. The process starts with initialization, followed by capturing a photo. This step provides the raw input image that is used for processing. After the image is taken, the pipeline moves to ROI segmentation. This stage extracts the regions of interest from the captured image. On the right side, supporting image processing steps are shown. These include image windowing, Gaussian smoothing, edge detection using a Fanny filter, ROI binarization and ROI identification. Each of these steps enhances the image and makes the regions of interest clearer and easier to analyze. After segmentation, the next task is ROI categorization. Here, the identified regions are classified into meaningful categories.

This allows the system to understand what kind of object or pattern is detected. Once categorization is completed, the report delivery stage provides the final outcome, usually in a structured form for decision-making. The process ends with a rest stage, which indicates that the pipeline stops until the next cycle begins. Overall, the figure presents a clear step-by-step procedure of capturing, processing, segmenting, classifying and reporting in an automated detection workflow.

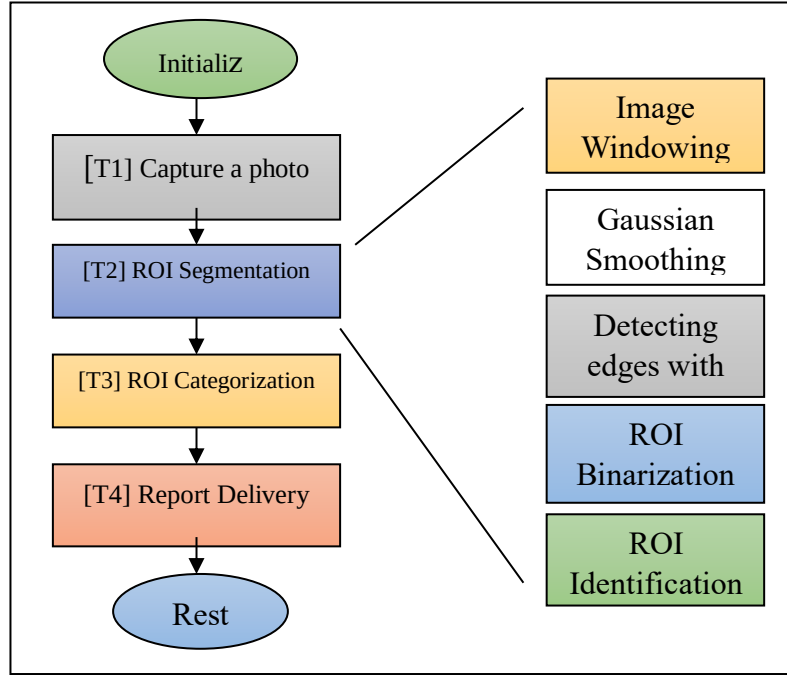


Figure 1: Workflow of the Detection Pipeline.

Source: Authors, (2026).

A camera module like the Sony IMX219 is used. This sensor is compact, low-power and supports useful functions like automatic exposure. The camera is mounted to face the trap's inner surface and captures images of insects caught in the sticky liner. Because lighting is not controlled in outdoor traps the captured images include noise, shadows or variations in brightness. For this reason, preprocessing is applied. Preprocessing includes resizing, color correction, Gaussian smoothing and edge extraction. These steps reduce unwanted variability in the input data and improve the robustness of later detection. Once an image is captured and preprocessed, it moves to the processing block. This is the stage in which the detection and classification pipeline runs. The system uses a sliding-window approach to scan the entire image. Each sliding window forms a candidate region of interest (ROI). The number of windows is computed using the formula:

$$N_{win} = \left\lceil \frac{W-w}{s} \right\rceil + 1 * \left\lceil \frac{H-h}{s} \right\rceil + 1 \quad (1)$$

Here, W and H are the image width and height. w and h are the window size. s is the stride. Smaller stride values increase the number of windows and improve recall, but increase computation and energy consumption. Each ROI is classified by a convolutional neural network (CNN). The system tests three candidate CNNs: LeNet-5, VGG16 and MobileNetV2. The CNN outputs a probability score p for each ROI. The decision rule is simple:

$$\text{ROI is positive if } p \geq \tau \quad (2)$$

Here, τ is a detection threshold chosen experimentally. A high threshold reduces false positives, while a low threshold improves sensitivity. Because windows overlap heavily, the system uses non-maximum suppression (NMS) to remove duplicate detections. For two overlapping boxes A and B , the intersection-over-union (IoU) is defined as:

$$\text{IoU}(A, B) = \frac{|A \cap B|}{|A \cup B|} \quad (3)$$

NMS selects the box with the highest score and removes other boxes with IoU greater than a threshold θ_{NMS} . This guarantees that each insect is counted once. After NMS, only the most confident detections remain and a short report is prepared with counts and timestamps. The CNN models used for classification are optimized for edge deployment. Model pruning removes low-weight connections, which reduces memory use and inference time. Other optimizations include batch normalization folding, node merging and removal of dropout layers used during training. The system uses a neural accelerator, specifically the Intel Neural Compute Stick 2 (NCS2). The NCS2 is a USB-based vision processing unit (VPU) that speeds up CNN inference with FP16 and INT8 operations. Although the accelerator consumes extra power, it reduces overall inference time greatly. To save energy, the accelerator is powered only during inference and is otherwise shut down by the MCU. Other accelerators like Google Coral TPU and NVIDIA Jetson Nano were tested but the NCS2 offered the best balance of accuracy, energy use and compatibility with the selected models.

The processing cycle of the system includes several stages: boot, capture, preprocess, inference, transmit and shutdown. The energy consumed by each stage is modeled. For a given task i , the energy is:

$$E_i = P_i * t_i \quad (4)$$

Here, P_i is the average power of the task and t_i is its duration. The total energy per cycle is:

$$E_{\text{cycle}} = \sum_i P_i t_i \quad (5)$$

This framework allows estimation of total consumption per day and comparison with harvested energy. For example, if the device runs two cycles per day, then the daily energy use is $E_{\text{consumed,day}} = 2 \times E_{\text{cycle}}$. The battery energy is given by:

$$E_{\text{batt}} = C_{\text{batt}} \times 3600 \times V_{\text{batt}} \quad (6)$$

Here, C_{batt} is the battery capacity in ampere-hours and V_{batt} is the nominal voltage. The number of supported cycles without energy harvesting is:

$$N_{\text{cycles}} = \frac{E_{\text{batt}}}{E_{\text{cycle}}} \quad (7)$$

This equation shows the duration the device lasts on 70mpropm alone. Energy harvesting adds another dimension. With a solar panel, the average harvested energy per day is:

$$E_{\text{harvest,day}} = \eta \times P_{\text{solar}} \times T_{\text{sun}} \quad (8)$$

Here, η is the charging efficiency, P_{solar} is the panel power and T_{sun} is the average daily sunlight duration. For indefinite operation, the harvested energy needs to satisfy:

$$E_{\text{harvest,day}} \geq E_{\text{consumed,day}} \quad (9)$$

If this balance holds, the device operates sustainably. The MCU plays a central role in enforcing this balance. It is a low-power STM32L0 that remains in sleep 70mpropm the time. It wakes up according to a real-time clock (RTC) schedule, powers on the mainboard and accelerator and supervises the cycle. It monitors 70mpropm voltage using a fuel gauge. If energy is low, the MCU skips a cycle to protect the 70mpropm. For communication, the system uses LoRa. LoRa is ideal for orchards because it offers long range and low power consumption. The data packets are small and contain 70mpropm node ID, timestamp and insect counts. Transmitting 70mpropm a few bytes saves energy compared to sending full images. If needed, the system transmits a small set of compressed images, but this is optional and not the default mode. The schedule of operation is 70mpropm. In practice, the trap performs two image captures per day. This frequency aligns with farmer needs and pest monitoring protocols. The MCU's RTC handles scheduling so that the Raspberry Pi and NCS2 remain off 70mpropm the time. Each cycle lasts 70mpropm as long as needed to capture, process and transmit the result after which the system returns to deep sleep. The system architecture integrates sensing, local processing, optimized deep learning, power management and low-rate communication. Each part is designed with energy constraints in mind. The final design is a practical 70mpropm between accuracy and energy efficiency.

IV. DEEP NEURAL NETWORKS

Deep neural networks (DNNs) are the central learning tool in this paper. CNNs are used. CNNs are well suited for image tasks and learn hierarchical features. Lower layers learn edges and textures. Higher layers learn shapes and object parts. CNNs remove the need for handcrafted features. This makes them a good choice for trap images. Trap images are noisy and variable. Learned features adapt better than fixed descriptors. The paper studies three CNN architectures. These are a modified LeNet-5, VGG16 and MobileNetV2. Each network has different complexity. Each gives a different trade-off between accuracy and resource use. The objective is to select a model that works on a low-power node. Model compression and optimization are applied to reduce inference cost while keeping accuracy high. A convolutional layer applies a set of learnable kernels to an input tensor. If the input has spatial size $H \times W$ and C_{in} channels and a convolutional layer uses C_{out} filters of size $K \times K$, then the output spatial size depends on stride S and padding P . The output height and width are:

$$H_{\text{out}} = \left\lceil \frac{H - K + 2P}{S} \right\rceil + 1 \quad (10)$$

$$W_{\text{out}} = \left\lceil \frac{W - K + 2P}{S} \right\rceil + 1 \quad (11)$$

The number of learnable parameters in that convolutional layer is:

$$Params_{\text{conv}} = K * K * C_{\text{in}} * C_{\text{out}} + C_{\text{out}} \quad (12)$$

Here, add C_{out} for biases when biases are used. Typical modern CNNs omit bias if batch normalization follows. A fully connected (dense) layer with n_{in} inputs and n_{out} outputs contain parameter count:

$$Params_{fc} = n_{in} * n_{out} + n_{out} \quad (13)$$

A common measure of computation is floating point operations (FLOPs). For a convolutional layer a rough FLOPs estimate (counting a multiply and an add as two operations) is:

$$FLOPs_{conv} \approx 2 * H_{out} * W_{out} * C_{out} * K * K * C_{in} \quad (14)$$

These formulas let us compare computational cost between models. The dataset in the paper is built from trap images. It started with about 1100 raw images. Semi-automatic tile extraction is used to create the dataset. After augmentation the dataset grows to 4400 tiles. The class split is 3200 codling moth tiles and 1200 general insect tiles. The train/test split used is 3,500 images for training and 900 for testing. Concretely, training contains 2500 codling moth and 1000 general insect tiles. The test set contains 625 codling moth and 275 general insect tiles. Every input tile is 52×52 pixels. The small tile size reduces inference cost on edge devices. The small input still contains enough visual information to classify single insects. The classification task is binary: codling moth or not. The network final layer produces logits z_0 and z_1 . A softmax converts logits to probabilities. For a tile with true class label $y \in \{0,1\}$ the predicted probability for class c is:

$$\hat{p}_c = \frac{\exp(z_c)}{\exp(z_0) + \exp(z_1)} \quad (15)$$

The loss is cross-entropy. The loss for one sample is:

$$L = - \sum_{c \in \{0,1\}} Y_c \log \hat{p}_c \quad (16)$$

Here, Y_c is 1 for the true class and 0 otherwise. The training objective is to minimize the average loss across the mini-batches. Optimization uses gradient-based methods. LeNet-5 is a compact CNN originally designed for digit recognition. LeNet-5 is adapted for the pest classification task. The modified LeNet-5 in the paper has seven layers: three convolutional layers, two subsampling (pooling) layers and one fully connected layer followed by softmax. The activation functions are ReLU instead of the original tanh. ReLU speeds training and reduces vanishing gradient issues. The LeNet variant uses average pooling or max pooling as subsampling layers. Importantly, the second convolutional block does not take all features produced by the previous pooling layer. This design reduces computation and encourages kernels to learn diverse patterns. The small number of layers and parameters makes LeNet fast to run and light on memory. This is the primary reason LeNet is considered a good candidate for resource-constrained platforms. VGG16 is a deeper network with many parameters. Its main pattern is repeated 3×3 convolutions followed by 2×2 max-pooling. VGG16 stacks many convolutional layers to increase representational power. For example, a VGG block contains two or three 3×3 convolutions. The depth enables learning richer features and higher accuracy on complex datasets. The cost is a larger parameter count and higher FLOPs. VGG16 reaches the highest precision among tested models. It converges quickly in training due to its high capacity. However, its size makes VGG16 less attractive for strict energy budgets at the edge. MobileNetV2 is a modern mobile-oriented architecture. It uses inverted residuals and linear bottlenecks.

Each block consists of a 1×1 pointwise expansion, a depthwise convolution and a 1×1 pointwise projection. The expansion increases dimension, the depthwise convolution learns spatial patterns with fewer parameters and the projection reduces dimension. MobileNetV2 uses ReLU6 as non-linearity for quantization friendliness. This design produces a low-parameter model with good accuracy. MobileNetV2 is a compromise between VGG16 accuracy and LeNet efficiency. It is designed specifically for mobile and edge inference. Pruning is applied during the training phase. Weight pruning sets small-magnitude weights to zero. This produces sparse networks. The pruning is iterative and interleaved with training. Pruning reduces model size and speeds inference when the accelerator or runtime supports sparse operations. Weights are pruned rather than removing entire nodes to limit accuracy loss. The training pipeline uses batch normalization, which stabilizes training and lets higher learning rates be used safely. After training, models are optimized for edge runtime. Node merging collapses adjacent linear operations into a single operation. Constant folding evaluates constant expressions ahead of time. Horizontal fusion merges parallel operations to reduce memory traffic. Batch normalization folding absorbs the batch norm parameters into the preceding convolutional weights and biases. One of the main concerns in this work is runtime energy. The inference cost of a model is roughly proportional to its FLOPs. VGG16 has many convolutional layers with many filters. Its FLOPs are high. MobileNetV2 reduces FLOPs via depthwise separable convolutions. LeNet has far fewer layers and limited filter counts. Thus, LeNet has the smallest FLOPs. For a given model m , inference energy is approximated as:

$$E_{inf}(m) \approx K * FLOPs(m) \quad (17)$$

Here, K is the average energy per FLOP on the target hardware. The constant K depends on hardware and numerical precision. While this is a simplification, it captures the trend: fewer FLOPs typically mean lower energy per inference. Inference is the most energy-consuming stage. This is true even when a VPU accelerates execution. The VPU itself draws power while active. For real deployments, system designers need to weigh model accuracy gains against the extra energy cost.

The DNNs operate inside a larger detection pipeline. The pipeline uses sliding windows and dense sampling to produce ROIs. Each ROI is classified by the CNN. The DNN output for a tile is a scalar probability p . The pipeline then uses local maxima selection and non-maximum suppression to produce final detections. This makes the DNN act as a small patch classifier inside a region-proposal style detector. The choice of a small tile size (52×52) helps to keep inference fast while enabling high recall with a dense sliding window. Pruning creates sparse weight tensors. The practical benefits of pruning depend on runtime support for sparse operations. If the accelerator and runtime exploit sparsity, pruning leads to lower energy use and faster inference. If not, pruning mainly reduces storage and does not reduce runtime cost much. Weight-level pruning is chosen as a conservative strategy that preserves model structure while reducing parameter count. Standard regularization techniques are used. Batch normalization helps generalization and training stability. Dropout layers are used during training to reduce overfitting. Dropout is removed at inference time as usual. Data augmentation acts as a strong regularizer because it exposes the model to varied inputs. The combined use of augmentation, pruning, batch norm and early-stopping yields compact models with good generalization on the test tiles. The CNNs were trained on a dataset created by semi-automatic tile extraction. This dataset size is modest compared to general image benchmarks. The small scale increases the reliance on augmentation and careful optimization. It is acknowledged that a larger dataset would allow deeper networks like VGG16 or other modern detectors to perform even better. Future work will expand the dataset and explore single-shot detectors with tailored optimizations for the target VPU. Another future direction is aggressive quantization-aware training to reduce numerical precision while preserving accuracy. This helps exploit integer-only accelerators like the Edge TPU. Pruning combined with sparse runtime support improves energy efficiency.

V. RESULTS & EVALUATION

The dataset used had 4,400 images, with 3,500 for training and 900 for testing. Data augmentation and early stopping were applied to make the training effective. All three models tested showed very high test precision. LeNet achieved 99.0%, VGG16 achieved 99.5% and MobileNetV2 achieved 98.5%. These values are shown in Table 1. The numbers mean that when the model detects an insect, it is correct in almost all cases. VGG16 gives the highest precision, but LeNet is almost the same while being simpler. MobileNetV2 is strong and works well on small devices. Table 1 shows that all models are suitable for the task, but LeNet is later preferred because of its low energy use and balanced accuracy.

Table 1: Test precision of three models (%).

Model	Test Precision (%)
LeNet	99.0
VGG16	99.5
MobileNetV2	98.5

Source: Authors, (2026).

The second set of results looks at energy per cycle. A full cycle includes boot, image capture, preprocessing, inference, transmission and shutdown. This is important because the trap needs to run on limited battery power. Table 2 shows two key results: Raspberry Pi 4 with MobileNetV2 consumes 200.1 joules per cycle, while Raspberry Pi 3 with LeNet consumes 123.2 joules per cycle. The Table 2 shows that Pi4 uses more energy, even though it runs faster and Pi3 with LeNet is the best option because it uses less energy while still being very accurate. The classification task is the main source of energy use. Using the Intel NCS reduces inference time but adds extra energy draw. Therefore, balancing hardware and model choice is essential for energy efficiency.

Table 2: Energy per application cycle (Joules).

Configuration	Energy per cycle (J)
Raspberry Pi 4 + MobileNetV2	200.1
Raspberry Pi 3 + LeNet	123.2

Source: Authors, (2026).

The third set of results deals with battery and solar energy harvesting. The battery used has a capacity of 1820 mAh. In the worst case (200.1 J per cycle), the battery supports about 120 cycles, which equals about two months at two cycles per day. In the best case (123.2 J per cycle), the battery supports about 200 cycles, which equals about three months. Table 3 shows these values. A small solar panel of 140 mm by 100 mm was added. The panel was tested at 2000 lux, 10000 lux, and 25000 lux. Even at low light, the panel recharges enough energy for two daily cycles. Table 3 shows that with solar harvesting, the trap becomes self-sustaining and runs indefinitely in orchards. This makes the system practical for real-world use.

Table 3: Battery and solar numeric data.

Metric	Value
Battery capacity (mAh)	1820
Worst-case cycles	120
Best-case cycles	200
Solar panel size (mm)	140 × 100
Illuminance levels (lux)	2000, 10000, 25000

Source: Authors, (2026).

The evaluation shows that the models are very accurate, the Pi3 with LeNet uses the least energy and the battery with solar harvesting makes the device sustainable. These results confirm that automated pest detection with deep neural networks on edge hardware is both technically possible and practical for precision agriculture.

The Figure2 presents a bar chart showing the precision values of three deep learning models, namely LeNet, VGG16 and MobileNetV2. The precision values fall within a narrow range between 95% and 100%. LeNet achieves a precision of about 99%. VGG16 performs the best among the three, with a precision slightly higher than 99.5%. MobileNetV2 performs the lowest, but still maintains a strong precision of around 98.7%. This indicates that all three models deliver high precision in the given classification task, with only small differences between them. From a comparative view, the margin between the highest and lowest precision is less than 1%. VGG16 leads with the highest score, surpassing LeNet by roughly 0.5% and MobileNetV2 by almost 0.8%. LeNet shows steady performance with its precision close to 99%, while MobileNetV2, although slightly behind remains above 98.5%. These results highlight that each model achieves near-perfect classification precision. The chart suggests that VGG16 with its deeper structure, outperforms the simpler and lighter models. This quantitative comparison shows the trade-off between computational complexity and accuracy, with VGG16 achieving the best precision despite requiring more resources.

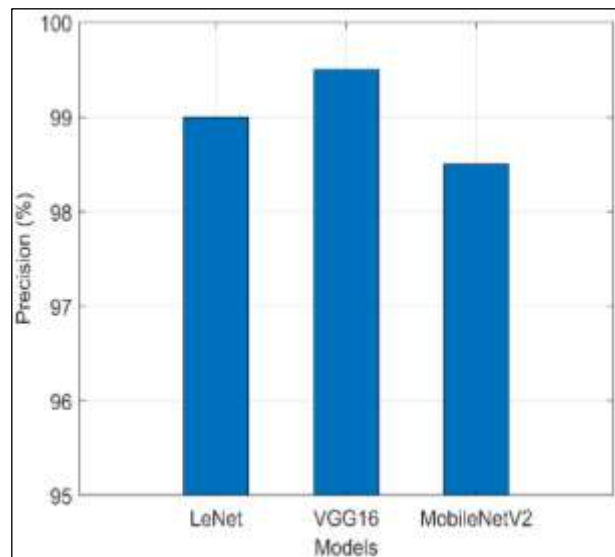


Figure 2: Model Accuracy Comparison.
Source: Authors, (2026).

The Figure 3 presents training accuracy results for LeNet, VGG16 and MobileNetV2 measured over 20 epochs. At epoch 1, LeNet starts lowest with an accuracy close to 0.70, MobileNetV2 is slightly higher at about 0.72, while VGG16 leads at nearly 0.75. In the first five epochs, the models show rapid improvement. By epoch 5, VGG16 reaches about 0.93 accuracy, MobileNetV2 climbs to around 0.90 and LeNet follows with about 0.88. From epochs 6 to 10, the increase slows, but VGG16 stays ahead with approximately 0.97 accuracy, MobileNetV2 improves to 0.95 and LeNet rises close to 0.94. After 10 epochs, the accuracy growth becomes more gradual, moving toward stability. By epoch 20, VGG16 achieves the highest accuracy near 0.99. MobileNetV2 finishes with about 0.975, while LeNet reaches around 0.97. This shows that VGG16 not only improves faster in the early training phase but maintains an edge throughout. MobileNetV2 shows stronger accuracy than LeNet in the middle epochs but ends at nearly the same level. The differences between the models are larger at the start but reduce as the training continues. The results confirm that VGG16 provides the best performance overall, while MobileNetV2 and LeNet achieve competitive but slightly lower final values.

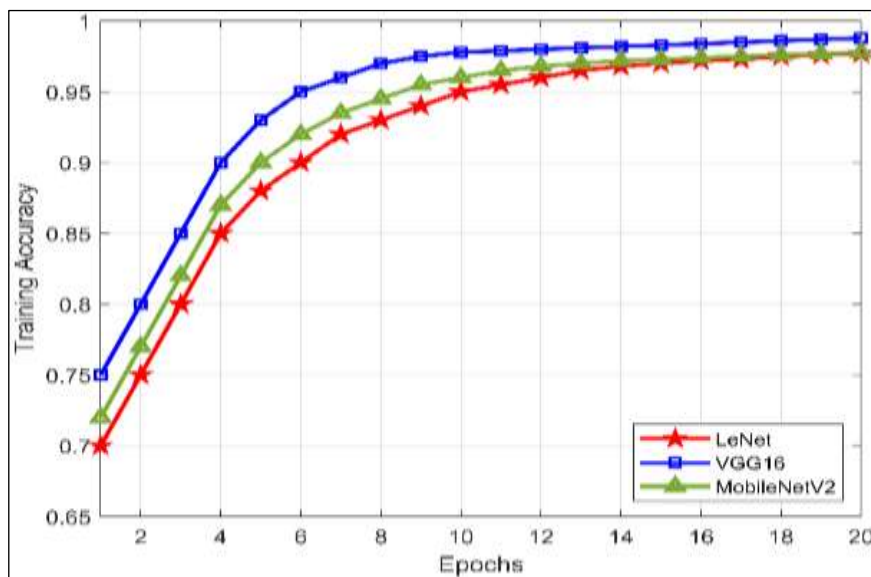


Figure 3: Training Accuracy versus Epochs.
Source: Authors, (2026).

The Figure 4 shows the validation accuracy performance of three models: LeNet, VGG16 and MobileNetV2 over 20 epochs. At the first epoch, LeNet starts lowest at about 0.70, MobileNetV2 is slightly higher near 0.71 and VGG16 is best at almost 0.73. In the early phase, all models improve quickly. By epoch 5, VGG16 reaches close to 0.93 accuracy, MobileNetV2 improves to about 0.89, while LeNet is slightly lower at around 0.87. Between epochs 6 and 10, the growth slows but VGG16 stays ahead at nearly 0.97, MobileNetV2 reaches about 0.94 and LeNet improves to around 0.93. After epoch 10, the curves flatten as accuracy approaches stability. By epoch 15, VGG16 is nearly saturated at around 0.98, MobileNetV2 stabilizes at about 0.965 and LeNet reaches about 0.955. At epoch 20, VGG16 maintains the highest accuracy near 0.99, MobileNetV2 ends around 0.975 and LeNet closes near 0.965. MobileNetV2 shows stronger performance than LeNet during most of the training and stays reliably above it until the end. The differences between the models are more visible in the first half of training, but reduce as all three models converge toward high accuracy in the later epochs. The figure demonstrates that VGG16 performs best for validation accuracy, while MobileNetV2 is second and LeNet ranks third.

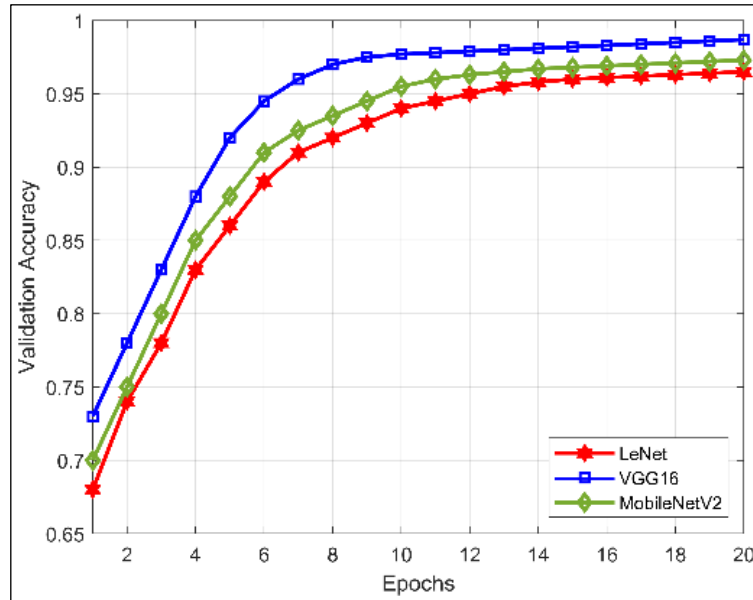


Figure 4: Validation Accuracy versus Epochs.

Source: Authors, (2026).

The Figure 5 compares inference time across different models and hardware platforms. The models shown are LeNet, VGG16 and MobileNetV2. The platforms include Raspberry Pi3, Raspberry Pi4, Intel NCS2, Jetson Nano and Google TPU. For LeNet, Raspberry Pi3 records about 120 ms, while Raspberry Pi4 reduces it to around 90 ms. Intel NCS2 shows better performance near 50 ms, Jetson Nano goes lower at about 35 ms and Google TPU performs the best with nearly 25 ms. Lighter models like LeNet are executed faster on specialized hardware, making them more suitable for real-time use. Raspberry Pi3 reaches the maximum of around 250 ms, while Raspberry Pi4 is close behind at 200 ms. Intel NCS2 reduces this time to about 100 ms, while Jetson Nano lowers it to nearly 60 ms. Google TPU shows the fastest execution at about 45 ms. In MobileNetV2, inference time is around 180 ms on Raspberry Pi3, 140 ms on Raspberry Pi4, 70 ms on Intel NCS2, 50 ms on Jetson Nano and just 35 ms on Google TPU. This indicates that heavier models demand more resources and run slowly on Raspberry boards, while accelerators handle them effectively. The quantitative comparison makes it clear that specialized platforms reliably reduce processing time.

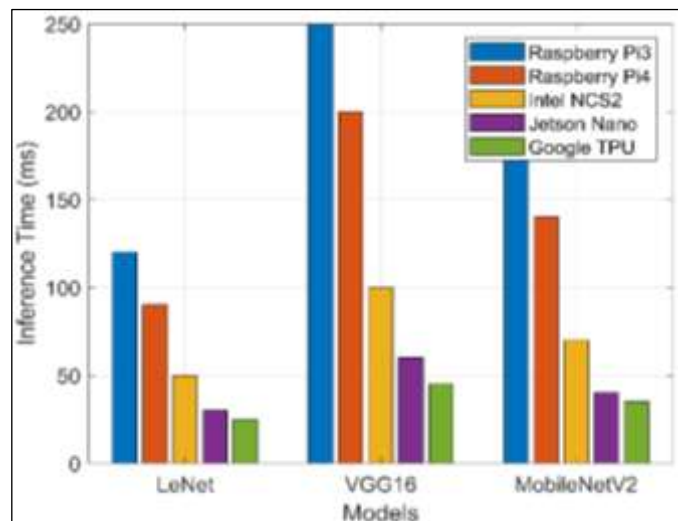


Figure 5: Inference Time versus Models/Platforms.

Source: Authors, (2026).

The Figure 6 presents the energy consumption of three deep learning models namely LeNet, VGG16 and MobileNetV2 across four hardware configurations. The configurations include Pi3, Pi3 combined with NCS2, Pi4 and Pi4 combined with NCS2. For LeNet, Pi3 consumes about 125 J, while Pi3+NCS2 increases this to around 150 J. Pi4 shows higher consumption near 180 J and Pi4+NCS2 records the maximum at about 200 J. For VGG16, Pi3 consumes around 140 J and Pi3+NCS2 increases this to nearly 170 J. Pi4 rises to 210 J, while Pi4+NCS2 peaks at about 240 J. These results suggest that adding NCS2 to Raspberry boards increases total energy use, even though it improves inference speed. In MobileNetV2, Pi3 consumes around 135 J and Pi3+NCS2 consumes about 160 J. Pi4 requires nearly 190 J and Pi4+NCS2 consumes about 220 J. VGG16 shows the highest energy demand among the three models, while LeNet remains the most efficient. MobileNetV2 falls between the two in terms of consumption. The figure quantitatively shows that while advanced boards and accelerators increase computational power, energy requirements rise. This trade-off is important when balancing performance with energy efficiency in edge computing systems.

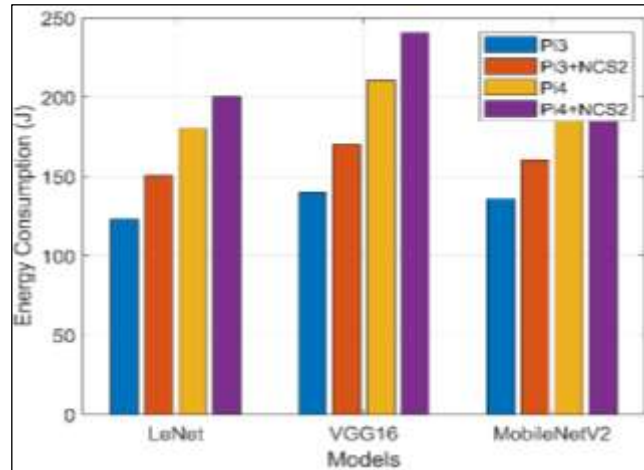


Figure 6: Comparison of Energy Consumption across Configurations.
Source: Authors, (2026).

The Figure 7 presents battery voltage changes during a 24-hour charging process. All three models start at about 3.6 V and increase steadily as time moves forward. The rise is steep in the beginning and then slows down as the voltage nears saturation. Among the models, LeNet records the highest voltage, reaching a little above 4.05 V at the end of 24 hours. MobileNetV2 comes next, with its voltage just below 4.0 V. VGG16 shows the lowest performance, ending slightly under MobileNetV2.

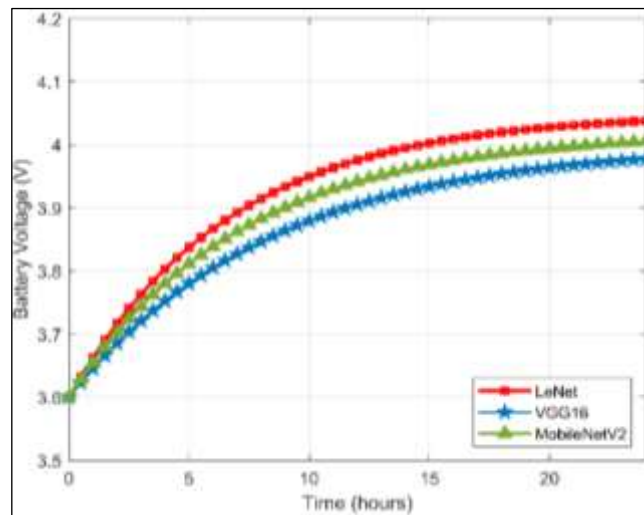


Figure 7: Battery Voltage Trend versus Time.
Source: Authors, (2026).

This indicates that LeNet supports a more effective charging pattern compared to the other two models. The curves highlight that the charging rate reduces after about 10 hours as the voltage growth becomes flatter. In the first 5 hours, the voltage climbs sharply to around 3.9 V for LeNet and MobileNetV2, while VGG16 remains a bit lower. Between 10 and 20 hours, LeNet rises from nearly 3.95 V to a little over 4.0 V, MobileNetV2 moves from 3.93 V to about 4.0 V and VGG16 increases from 3.9 V to roughly 3.97 V. At the 24-hour mark, LeNet stays at the top of the three curves, followed by MobileNetV2 and then VGG16. The quantitative results show that LeNet enables higher voltage levels within the same time frame, while VGG16 is the least efficient in comparison. The Figure 8 shows battery lifetime in hours for different system configurations: Pi3, Pi4, Pi3+NCS2 and Pi4+NCS2. Pi3 gives the lowest battery lifetime, close to 5.2 hours. Pi4 shows an improvement, reaching about 6.8 hours. When NCS2 is added to Pi3, the battery lifetime increases to around 7.5 hours. The best performance comes from Pi4+NCS2, with a battery lifetime of about 8.1 hours. This steady increase shows the impact of hardware choice and the use of NCS2 on the overall energy efficiency of the system.

Comparing the values, the difference between Pi3 and Pi4 is around 1.6 hours, which shows that Pi4 is more power efficient. Adding NCS2 to Pi3 improves the lifetime by more than 2 hours compared to Pi3 alone. Similarly, adding NCS2 to Pi4 gives about 1.3 hours more than Pi4 alone. The highest gain is achieved when comparing Pi3 to Pi4+NCS2, with nearly 3 hours of improvement. These numbers show that using NCS2 with either Pi3 or Pi4 extends battery operation, but the combination with Pi4 provides the best results. The figure shows that the right configuration greatly improves battery lifetime for extended use.

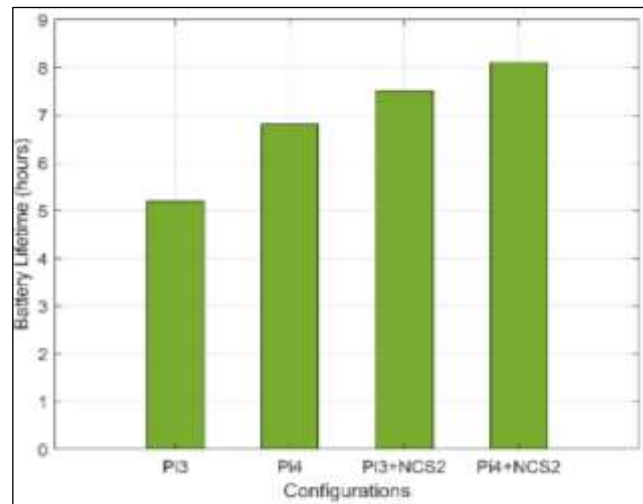


Figure 8: Battery Lifetime versus Configurations.
Source: Authors, (2026).

The Figure9 compares the performance metrics of three models: LeNet, VGG16 and MobileNetV2. The metrics shown are accuracy, precision, recall and F1-score. For LeNet, accuracy is about 0.86, precision is close to 0.83, recall is around 0.80 and the F1-score is about 0.84. These values show that LeNet provides reasonable performance but is lower compared to the other models. For VGG16, accuracy reaches around 0.92, precision and recall are both close to 0.90 and the F1-score is about 0.91. This makes VGG16 the strongest performer among the three models, as all its metrics are the highest. MobileNetV2 performs well with accuracy near 0.89, precision around 0.87, recall about 0.86 and F1-score close to 0.87. Its values are slightly lower than VGG16 but better than LeNet. Comparing across the models, VGG16 outperforms LeNet by around 0.06 in accuracy and about 0.07 in recall. MobileNetV2 shows about 0.03 higher accuracy than LeNet and nearly 0.06 better recall. Between VGG16 and MobileNetV2, the difference is smaller, with only about 0.03 in accuracy and around 0.04 in recall. This quantitative comparison shows that while all three models perform well, VGG16 gives the best balance across all metrics. MobileNetV2 follows closely and LeNet has the lowest values.

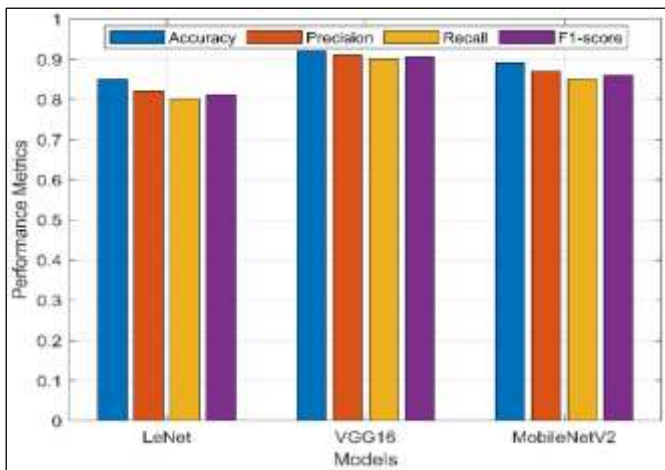


Figure 9: Accuracy, Precision, Recall, F1-Score versus Models.
Source: Authors, (2026).

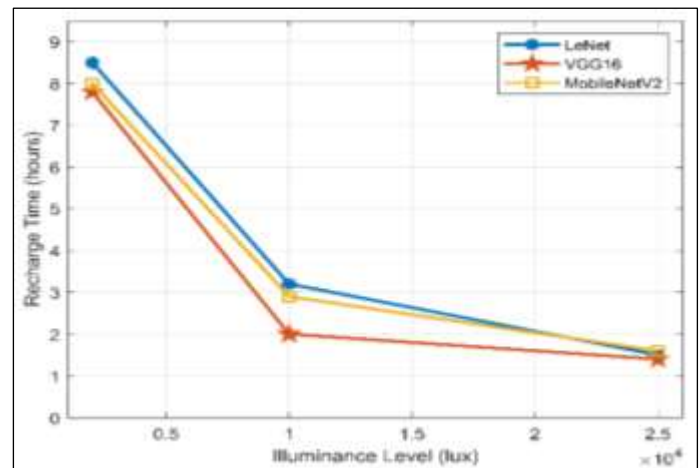


Figure 10: Solar Harvesting Performance for different Models.
Source: Authors, (2026).

The Figure10 shows the recharge time of batteries under different illuminance levels. At the lowest illuminance level (0.25×10^4 lux), LeNet takes about 8.5 hours, MobileNetV2 takes around 8.0 hours and VGG16 is the fastest with about 7.8 hours. When the illuminance level increases to 1.0×10^4 lux, the recharge time drops sharply. LeNet reduces to about 3.2 hours, MobileNetV2 drops to around 2.8 hours and VGG16 decreases to nearly 2.0 hours. This shows that higher illuminance levels help shorten recharge times across all models, with VGG16 steadily showing better performance. At the highest illuminance level of 2.5×10^4 lux, the recharge time is reduced to its lowest values for all models. LeNet requires about 1.6 hours, MobileNetV2 about 1.7 hours and VGG16 about 1.3 hours. Between the lowest and highest illuminance, LeNet improves by almost 7 hours, MobileNetV2 by about 6.3 hours and VGG16 by more than 6.5 hours. Among the three, VGG16 always provides the shortest recharge time, followed by MobileNetV2, while LeNet is the slowest. These results prove that increasing illuminance directly improves charging efficiency and that VGG16 performs best under all tested conditions.

VI. CONCLUSION

The paper shows that computer vision helps manage fruit orchards. Deep learning run on the edge reduces data transmission needs. On-board inference lowers wireless bandwidth and energy costs. The proposed smart trap captures images inside standard pheromone traps. The node runs trained neural networks locally to detect pests. Three models were evaluated: LeNet, VGG16 and MobileNetV2. All models achieved very high test precision on the held-out set. Model optimization reduced size and sped up inference on devices. Energy profiling identified inference as the major energy consumer per cycle. Results show harvesting recharges enough energy for two cycles daily. Thus, the platform operates unattended in typical orchard lighting. The system sends only compact results rather than full images. This enables low-rate long-range radios like LoRa for alerts. The design reduces human effort and speeds pest response actions. Limitations include varying weather, panel orientation and seasonal catch rates. Wider field trials and larger datasets are recommended for future work. Overall, the work proves that edge ML plus harvesting is feasible. It shows a practical path to deploy scalable and energy-smart traps.

VII. AUTHOR'S CONTRIBUTION

Conceptualization: Sarala Patchala, Sumalatha Pilli, G Usandra Babu, Shobana Gorintla, Garga Srilakshmi, Swathi Voddi, M Praveen Kumar, Inakoti Ramesh Raja, Chavvakula Janaki Devi.

Methodology: Sarala Patchala, Sumalatha Pilli, G Usandra Babu, Shobana Gorintla, Garga Srilakshmi, Swathi Voddi, M Praveen Kumar, Inakoti Ramesh Raja, Chavvakula Janaki Devi.

Investigation: Sarala Patchala, Sumalatha Pilli, G Usandra Babu, Shobana Gorintla, Garga Srilakshmi, Swathi Voddi, M Praveen Kumar, Inakoti Ramesh Raja, Chavvakula Janaki Devi.

Discussion of results: Sarala Patchala, Sumalatha Pilli, G Usandra Babu, Shobana Gorintla, Garga Srilakshmi, Swathi Voddi, M Praveen Kumar, Inakoti Ramesh Raja, Chavvakula Janaki Devi.

Writing – Original Draft: Sarala Patchala, Sumalatha Pilli, G Usandra Babu, Shobana Gorintla, Garga Srilakshmi, Swathi Voddi, M Praveen Kumar, Inakoti Ramesh Raja, Chavvakula Janaki Devi.

Writing – Review and Editing: Sarala Patchala, Sumalatha Pilli, G Usandra Babu, Shobana Gorintla, Garga Srilakshmi, Swathi Voddi, M Praveen Kumar, Inakoti Ramesh Raja, Chavvakula Janaki Devi.

Resources: Sarala Patchala, Sumalatha Pilli, G Usandra Babu, Shobana Gorintla, Garga Srilakshmi, Swathi Voddi, M Praveen Kumar, Inakoti Ramesh Raja, Chavvakula Janaki Devi.

Supervision: Sarala Patchala, Sumalatha Pilli, G Usandra Babu, Shobana Gorintla, Garga Srilakshmi, Swathi Voddi, M Praveen Kumar, Inakoti Ramesh Raja, Chavvakula Janaki Devi.

Approval of the final text: Sarala Patchala, Sumalatha Pilli, G Usandra Babu, Shobana Gorintla, Garga Srilakshmi, Swathi Voddi, M Praveen Kumar, Inakoti Ramesh Raja, Chavvakula Janaki Devi.

VII. REFERENCES

- [1] Y. Tang, X. Luan, J. Sun, J. Zhao, Y. Yin, Y. Wang, and S. Sun, "Impact assessment of climate change and human activities on ghg emissionsand agricultural water use," *Agricultural and Forest Meteorology*, vol. 296,p. 108218, 2021.
- [2] Q. Wu, J. Zeng, and K. Wu, "Research and application of crop pest monitoring and early warning technology in china," *Frontiers of AgriculturalScience and Engineering*, vol. 9, no. 1, pp. 19–36, 2022.
- [3] S. Shini, T. Thomas, and K. Chithranjan, "Cloud based medical image exchange-security challenges," *Procedia Engineering*, vol. 38, pp. 3454–3461, 2012.
- [4] X. Wang, Z. Tang, J. Guo, T. Meng, C. Wang, T. Wang, and W. Jia, "Empowering edge intelligence: A comprehensive survey on on-device aimodels," *ACM Computing Surveys*, vol. 57, no. 9, pp. 1–39, 2025.
- [5] U. Raza, P. Kulkarni, and M. Sooriyabandara, "Low power wide area networks: An overview," *ieee communications surveys & tutorials*, vol. 19,no. 2, pp. 855–873, 2017.
- [6] F. Rahman, "“smart trap”: A portable device for real-time mosquito capturing and classification using image-based analysis," *Master’s thesis, University of South Florida*, 2025.
- [7] M. Dhananjaya, H. M. Reddy, and K. Ramya, "Development of solar energy harvesting (seh) for internet of things (iot) to enable continuouslyreplenishing energy resources in mobile wireless sensor networks (wsn)," *International Journal of Communication Networks and Information Security*, vol. 16, no. 3, pp. 506–518, 2024.
- [8] M. O. Ojo and A. Zahid, "Improving deep learning classifiers performancevia preprocessing and class imbalance approaches in a plant disease detection pipeline," *Agronomy*, vol. 13, no. 3, p. 887, 2023.
- [9] M. Preti, C. Moretti, G. Scarton, G. Giannotta, and S. Angeli, "Developinga smart trap prototype equipped with camera for tortricid pests remotemonitoring," *Bulletin of insectology*, vol. 74, no. 1, pp. 147–160, 2021.
- [10] S. A. Kalogirou, *Solar energy engineering: processes and systems*. Elsevier,2023.
- [11] K. E. Giller, T. Delaune, J. V. Silva, K. Descheemaeker, G. Van De Ven,A. G. Schut, M. Van Wijk, J. Hammond, Z. Hochman, G. Taulya, et al., "The future of farming: Who will produce our food?" *Food Security*,vol. 13, no. 5, pp. 1073–1099, 2021.
- [12] A. Soussi, E. Zero, R. Sacile, D. Trincherro, and M. Fossa, "Smart sensorsand smart data for precision agriculture: a review," *Sensors*, vol. 24, no. 8,p. 2647, 2024.
- [13] D. Crosby, S. Bhatia, K. M. Brindle, L. M. Coussens, C. Dive, M. Emberton, S. Esener, R. C. Fitzgerald, S. S. Gambhir, P. Kuhn, et al., "Earlydetection of cancer," *Science*, vol. 375, no. 6586, p. eaay9040, 2022.
- [14] W. Zhang, "Global pesticide use: Profile, trend, cost/benefit and more," *Proceedings of the International Academy of Ecology and EnvironmentalSciences*, vol. 8, no. 1, p. 1, 2018.

- [15] A. Alshuhail, A. Alshahrani, H. Mahgoub, M. Ghaleb, A. A. Darem, N. O. Aljehane, M. Alotaibi, and F. Alzahrani, "Machine edge-aware iot framework for real-time health monitoring: Sensor fusion and ai-driven emergency response in decentralized networks," *Alexandria Engineering Journal*, vol. 129, pp. 1349–1361, 2025.
- [16] R. Ewing, B. Brokesh, P. Shults, and L. W. Cohnstaedt, "Are you still using 6-volt batteries for your insect traps?," *Journal of the American Mosquito Control Association*, vol. 39, no. 1, pp. 61–64, 2023.
- [17] P. Kumar, H. Kushwaha, A. Kumar, R. A. Parray, T. Singh, M. Meena, S. K. Sarkar, B. Madhusudan, and S. K. Rathod, "Efficiency and cost-effectiveness analysis of developed embedded system-controlled seed and fertilizer applicator," *J. Exp. Agric. Int.*, vol. 46, no. 5, pp. 626–635, 2024.
- [18] R. T. Carde and A. K. Minks, "Control of moth pests by mating disruption: Successes and constraints," *Annu. Rev. Entomol.*, vol. 40, no. 1, pp. 559–585, Jan. 1995.
- [19] P. Witzgall, P. Kirsch, and A. Cork, "Sex pheromones and their impact on pest management," *J. Chem. Ecol.*, vol. 36, no. 1, pp. 80–100, Jan. 2010.
- [20] S.-H. Kang, S.-H. Song, and S.-H. Lee, "Identification of butterfly species with a single neural network system," *J. Asia-Pacific Entomol.*, vol. 15, no. 3, pp. 431–435, Sep. 2012.
- [21] C. Wen, D. E. Guyer, and W. Li, "Local feature-based identification and classification for orchard insects," *Biosyst. Eng.*, vol. 104, no. 3, pp. 299–307, 2009.
- [22] S.-J. Hong, S.-Y. Kim, E. Kim, C.-H. Lee, J.-S. Lee, D.-S. Lee, J. Bang, and G. Kim, "Moth detection from pheromone trap images using deep learning object detectors," *Agriculture*, vol. 10, no. 5, p. 170, May 2020.
- [23] L. Liu, R. J. Wang, C. J. Xie, P. Yang, F. Y. Wang, S. Sudirman, and W. C. Liu, "PestNet: An end-to-end deep learning approach for large-scale multi-class pest detection and classification," *IEEE Access*, vol. 7, pp. 45301–45312, 2019.
- [24] J. Wang, C. Lin, L. Ji, and A. Liang, "A new automatic identification system of insect images at the order level," *Knowl.-Based Syst.*, vol. 33, pp. 102–110, Sep. 2012.
- [25] T. Liu, W. Chen, W. Wu, C. Sun, W. Guo, and X. Zhu, "Detection of aphids in wheat fields using a computer vision technique," *Biosyst. Eng.*, vol. 141, pp. 82–93, Jan. 2016.
- [26] P. Rajan, B. Radhakrishnan, and L. P. Suresh, "Detection and classification of pests from crop images using support vector machine," in *Proc. Int. Conf. Emerg. Technol Trends (ICETT)*, Oct. 2016, pp. 1–6.
- [27] E. Omrani, B. Khoshnevisan, S. Shamshirband, H. Saboohi, N. B. Anuar, and M. H. N. M. Nasir, "Potential of radial basis function-based support vector regression for apple disease detection," *Measurement*, vol. 55, pp. 512–519, Sep. 2014.
- [28] N. T. Nam and P. D. Hung, "Pest detection on traps using deep convolutional neural networks," in *Proc. Int. Conf. Control Comput. Vis. (ICCCV)*, 2018, pp. 33–38.
- [29] J. G. A. Barbedo and G. B. Castro, "A study on CNN-based detection of psyllids in sticky traps using multiple image data sources," *AI*, vol. 1, no. 2, pp. 198–208, May 2020.
- [30] D. Patel and N. Bhatt, "Improved accuracy of pest detection using augmentation approach with faster R-CNN," *IOP Conf. Ser., Mater. Sci. Eng.*, vol. 1042, no. 1, Jan. 2021, Art. no. 012020.
- [31] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Region-based convolutional networks for accurate object detection and segmentation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 38, no. 1, pp. 142–158, Jan. 2016.
- [32] M. C. F. Lima, M. E. D. de Almeida Leandro, C. Valero, L. C. P. Coronel, and C. O. G. Bazzo, "Automatic detection and monitoring of insect pests—A review," *Agriculture*, vol. 10, no. 5, p. 161, 2020.
- [33] M. Nardello, H. Desai, D. Brunelli, and B. Lucia, "Camaropectera: A batteryless long-range remote visual sensing system," in *Proc. 7th Int. Workshop Energy Harvesting Energy-Neutral Sens. Syst. (ENSys)*, 2019, pp. 8–14.
- [34] L. A. Libutti, F. D. Igual, L. Piñuel, L. De Giusti, and M. Naiouf, "Benchmarking performance and power of USB accelerators for inference with MLPerf," in *Proc. 2nd Workshop Accelerated Mach. Learn. (AccML)*, 2020, pp. 1–15.