agronomy

MDPI

*Article*

# Rapid Automatic Cacao Pod Borer Detection Using Edge Computing on Low-End Mobile Devices

**Eros Allan Somo Hacinas [1], Lorenzo Sangco Querol [1], Kris Lord T. Santos [2], Evian Bless Matira [2], Rhodina C. Castillo [3], Mercedes Arcelo [4], Divina Amalin [2] and Dan Jeric Arcega Rustia [5,*]**

1 Department of Computer Technology, De La Salle University, Manila 0922, Philippines; eros_hacinas@dlsu.edu.ph (E.A.S.H.); renzo_querol@dlsu.edu.ph (L.S.Q.)
2 Department of Biology, De La Salle University, Manila 0922, Philippines; kris.santos@dlsu.edu.ph (K.L.T.S.); matiraevianbless@gmail.com (E.B.M.); divina.amalin@dlsu.edu.ph (D.A.)
3 College of Agriculture, Sultan Kudarat State University, Tacurong 9800, Philippines; rhodinacastillo@sksu.edu.ph
4 Bureau of Plant Industry—Davao, Davao City 1004, Philippines; mercy_arcelo@yahoo.com
5 Wageningen Plant Research, Wageningen University & Research, 6708 PB Wageningen, The Netherlands
* Correspondence: dan.rustia@wur.nl

**Abstract:** The cacao pod borer (CPB) (*Conopomorpha cramerella*) is an invasive insect that causes significant economic loss for cacao farmers. One of the most efficient ways to reduce CPB damage is to continuously monitor its presence. Currently, most automated technologies for continuous insect pest monitoring rely on an internet connection and a power source. However, most cacao plantations are remotely located and have limited access to internet and power sources; therefore, a simpler and readily available tool is necessary to enable continuous monitoring. This research proposes a mobile application developed for rapid and on-site counting of CPBs on sticky paper traps. A CPB counting algorithm was developed and optimized to enable on-device computations despite memory constraints and limited capacity of low-end mobile phones. The proposed algorithm has an $F_1$-score of 0.88, with no significant difference from expert counts ($R^2$ = 0.97, $p$-value = 0.55, $\alpha$ = 0.05). The mobile application can be used to provide the required information for pest control methods on-demand and is also accessible for low-income farms. This is one of the first few works on enabling on-device processing for insect pest monitoring.

**Keywords:** mobile computing; mobile application; insect pest; sticky trap; deep learning



**Citation:** Hacinas, E.A.S.; Querol, L.S.; Santos, K.L.T.; Matira, E.B.; Castillo, R.C.; Arcelo, M.; Amalin, D.; Rustia, D.J.A. Rapid Automatic Cacao Pod Borer Detection Using Edge Computing on Low-End Mobile Devices. *Agronomy* **2024**, *14*, 502. https://doi.org/10.3390/agronomy14030502

Academic Editor: Luca Ruiu

Received: 30 January 2024
Revised: 19 February 2024
Accepted: 27 February 2024
Published: 29 February 2024



## 1. Introduction

The cacao pod borer (CPB), *Conopomorpha cramerella* (Snellen) (Lepidoptera: Gracillariidae), is a major insect pest of cacao (*Theobroma cacao* L.) in the Philippines and other Southeast Asian countries [1,2]. Females lay eggs on the surface of pods. Newly hatched larvae then bore through the pod to feed on the pulp and the placenta surrounding the beans. Feeding is characterized by tunnels and scars on tissues, which result in premature fruit ripening and clumping of beans making them difficult or even impossible to extract [3]. Economic thresholds have been established based on the percentage of pod infestations that were related to yield loss [1]. Almost no loss was observed with infestations up to 60%, but losses increase rapidly with higher infestation. Monitoring adult populations using pheromone traps may also provide some basis for setting thresholds; however, data must be based on entire cropping periods. Previous studies [1,2] indicated that application of insecticides during the low crop period kept populations of CPB below economic damage levels during the subsequent peak season. On the other hand, spraying during peak crop season had little effect on infestations. This suggests that the best return from pod spraying is likely to come from applications during the early stage of a rising crop, soon after the low-crop period. In the Philippines, the low-crop period is around May–July and peak

season starts in August. Based on our previous data on seasonal abundance of CPB in the Philippines [3], the average male CPB trap catch during the early stages of a rising crop period was around 30, which was the basis of the density thresholds used in this study. At low density (<10 CPB), no control is needed. At medium density (10–30 CPB) and high density (>30 CPB), appropriate control measures need to be employed before the peak population density.

In severe infestations, pods become completely unusable and yield losses can range from 30 to 50% [4]. Due to the detrimental impact of CPB on cacao production, farmers resort to various management strategies. Its management has relied heavily on the use of synthetic insecticides [5]; however, concerns of its adverse effects to human health and the environment directed efforts to develop alternative methods as part of an integrated pest management (IPM) system [6]. An alternative solution is the use of sex pheromones to trap adult male CPBs. The sex pheromone components of CPB were identified in 1986 [7] and have been successfully used in monitoring the population of CPB in Southeast Asian countries, including the Philippines [1]. Monitoring, one of the pillars of IPM, ensures a guided decision-making process [8]. Currently, monitoring by trapping involves the use of delta traps with removable sticky paper traps and sex pheromone-impregnated lures [9]. After the trapping period, sticky paper traps are collected to manually count the trapped CPBs. This may lead to inaccuracies since manual counting is often error-prone. More errors arise when a large number of insects are collected or when the farm manager lacks the technical expertise to recognize and differentiate CPBs from other insect species. An automated method that can accurately identify and count CPB in a short amount of time is therefore necessary for routine pest monitoring.

There are two digital approaches in monitoring insect pests: automated and semi-automated. A fully automated approach entails the use of wireless sensor nodes, which are equipped with cameras that capture sticky paper trap images [10]. The sticky paper trap images are analyzed locally or through cloud computing. This allows close to real-time monitoring of insect pest presence. However, its main bottleneck is its cost, which makes it difficult for low to middle income farmers to afford. On the other hand, semi-automated insect pest monitoring systems collect sticky paper trap images using mobile phones in controlled or uncontrolled environments [11]. The main advantage of this method is reduced equipment cost, which makes it more viable for use of most farmers. But currently, there is still limited work on developing mobile applications for insect pest counting.

Most mobile applications related to insect pest detection perform computations on the cloud, which requires a user to upload an image to a cloud server to receive image analysis results. Unfortunately, this approach necessitates a user's mobile phone to have an internet connection, which is not always feasible, especially for remote farming areas such as cacao plantations. Ref. [12] developed a mobile application that can detect insect pests from close up images of insects on leaves. They trained a Faster-RCNN object detection model then processed images in a cloud server with GPU support. Similarly, ref. [13] detected scale pests by employing multiple object detector models then used a cloud platform for computing. Ref. [14] tested model-centric, data-centric, and deployment-centric strategies to train object detector models for insect pest detection in viticulture. They presented optimization approaches so that each model will work on different mobile device models. The above-mentioned works reveal that mobile computing is a niche topic and several approaches have been taken to develop a convenient and reliable mobile application for insect pest detection. However, it also shows that more work must be carried out to achieve better performance in terms of speed and accuracy.

This work presents novel approaches for enabling mobile computing on low-end mobile devices. This work has three objectives: (1) Develop an insect pest detection algorithm that can locally run on low-end mobile devices without cloud server support; (2) Propose novel approaches for optimizing an edge-based insect pest detection algorithm; and (3) Design and develop a mobile application that can rapidly count cacao pod borers on

sticky paper traps. This work demonstrates the potential of mobile computing in integrated pest management, especially for managing remote farming environments.

## 2. Materials and Methods

### *2.1. Imaging Devices*

Three mobile phones were used for image collection, as shown in Table 1. The three mobile phones were selected since they were found to be commonly owned by Filipino phone users due to their affordability and usability [15]. Out of the three, Realme C30 was used to extensively test the developed mobile application. The testing was focused on two mobile phone specifications that affect detection performance and application compatibility: camera resolution and RAM size; Realme C30 has the poorest of these two mobile phone specifications.

**Table 1.** Mobile phones used for capturing sticky paper trap images.

| Device | Camera Resolution (px) | CPU | RAM | USD | PHP |
|---|---|---|---|---|---|
| Cherry Mobile Flare S8 Deluxe (Cherry Mobile, Manila, Philippines) | 3456 × 4608 | Octa-Core A55 @1.6 GHz | 4 GB | 90 | 5000 |
| Huawei Nova Y7 (Huawei, Shenzhen, China) | 4000 × 3000 | Octa-Core A53 @2.2 GHz | 4 GB | 155 | 8500 |
| Realme C3 (Realme, Guangdong, China) | 3264 × 2448 | Octa-Core A75 @1.8 GHz | 2 GB | 90 | 5000 |

### *2.2. Dataset Collection*

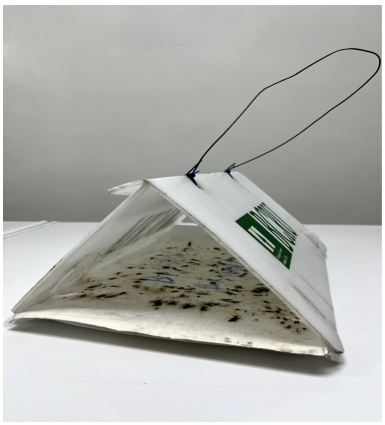
The sticky paper traps were collected from two different sites in Mindanao, Philippines: Davao City, Calinan and Cotabato, Makilala, as depicted in Figure 1. Figure 1b,c shows a sticky paper trap housed inside the delta trap. These delta traps were hung 1 m above cacao trees (Figure 1a,d) to attract CPBs using a sex pheromone-impregnated lure [1]. The pheromone components are produced by female CPBs; hence, only male CPBs will be caught by the traps. The sticky paper traps inside the delta traps were collected and replaced monthly from the field by farm managers. The traps were then assessed by experts to observe the level of CPB infestation over time and to evaluate the effectiveness of the lure. Two sizes of sticky paper traps were used: a short grid trap with 17 cm width and height, and a longer gridless trap with 18 cm width and 28.5 cm height. These traps had varying sizes and presence of grids since the trap design was also optimized during the collection. A plastic cover was put on top of the sticky side of each sticky paper trap; this prevents the sticking of papers to one another. Recommended working distances were determined by attaching the mobile phones to a phone stand. It was found that the optimal height for capturing complete images of a short sticky paper trap was 181 mm, and up to 231 mm for longer traps. Meanwhile, phone to stand distance of the phone stand arm was optimal at 150 mm for both sizes of paper traps, as illustrated in Figure 2. The working distances were determined for recommending an image acquisition setup for the users of the presented mobile application. The sticky paper trap images were taken under varying indoor and outdoor lighting conditions.

All datasets presented in Table 2 were manually annotated by experts using Roboflow image annotation platform [16]. Data collected from all sites between 2017 and 2020 were used for training and validation, while the more recent ones from 2022 to 2023 were used for testing. This temporal split between datasets ensures that the model is able to detect unseen data and simulate their performance during deployment. Furthermore, the CPB dataset was split based on density: low, medium, and high, which were based on economic thresholds determined from previous research. The ratios of the low, medium, and high CPBs on the training and validation have approximately the same proportion to ensure a balanced dataset. Images from training and validation were captured using all phones from Table 1, while the testing dataset is composed of images from Realme C30.

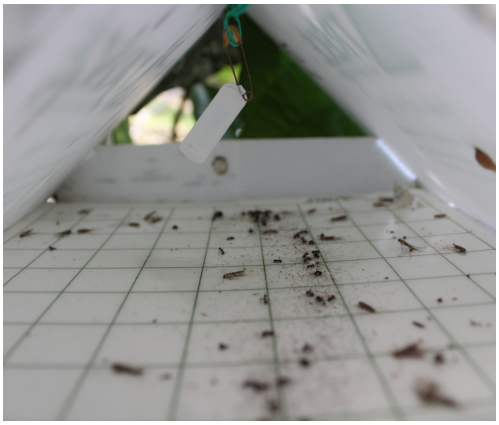
**Table 2.** Dataset statistical summary, where n is the number of CPBs in the sticky paper trap.

| Dataset | Collection Date | Number of Images Based on CPB Density | | |
|---|---|---|---|---|
| | | Low ($n < 10$) | Medium ($10 \leq n < 30$) | High ($n \geq 30$) |
| Training | 2017–2021 | 226 | 107 | 56 |
| Validation | 2017–2021 | 60 | 25 | 14 |
| Testing | 2022–2023 | 56 | 25 | 31 |

(**a**) Cacao tree



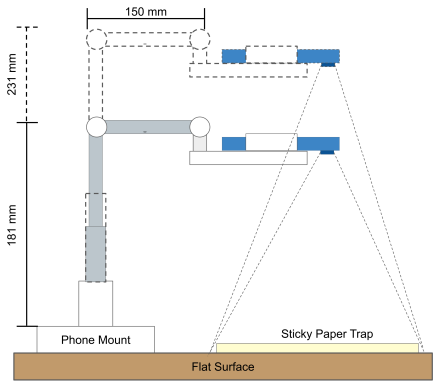
(**b**) Delta trap



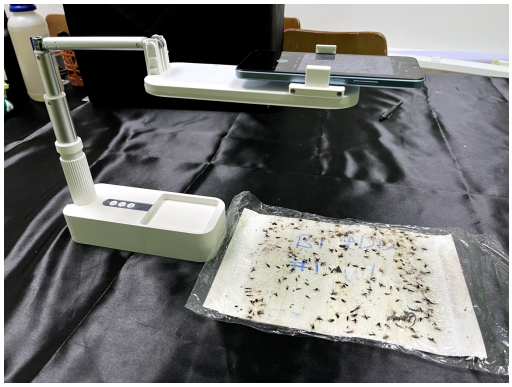
(**c**) Sticky paper trap inside the delta trap

(**d**) Delta trap hung at a cacao tree

**Figure 1.** Data collection methods in the field showing the cacao tree and delta traps housing both sticky paper trap and pheromone lure.




(**a**) Schematic diagram



(**b**) Actual setup

**Figure 2.** *Cont.*


(**c**) Sample image

**Figure 2.** Sticky paper trap image acquisition setup.

### *2.3. Mobile CPB Detection Algorithm*

An automatic CPB detection algorithm was developed for mobile devices, as shown in Figure 3. The algorithm was developed using Python 3.7, PyTorch 1.13.0, ultralytics 8.0.0, OpenCV 4.6.0, and Open Neural Network Exchange (ONNX) Runtime (ORT) 1.15.0.

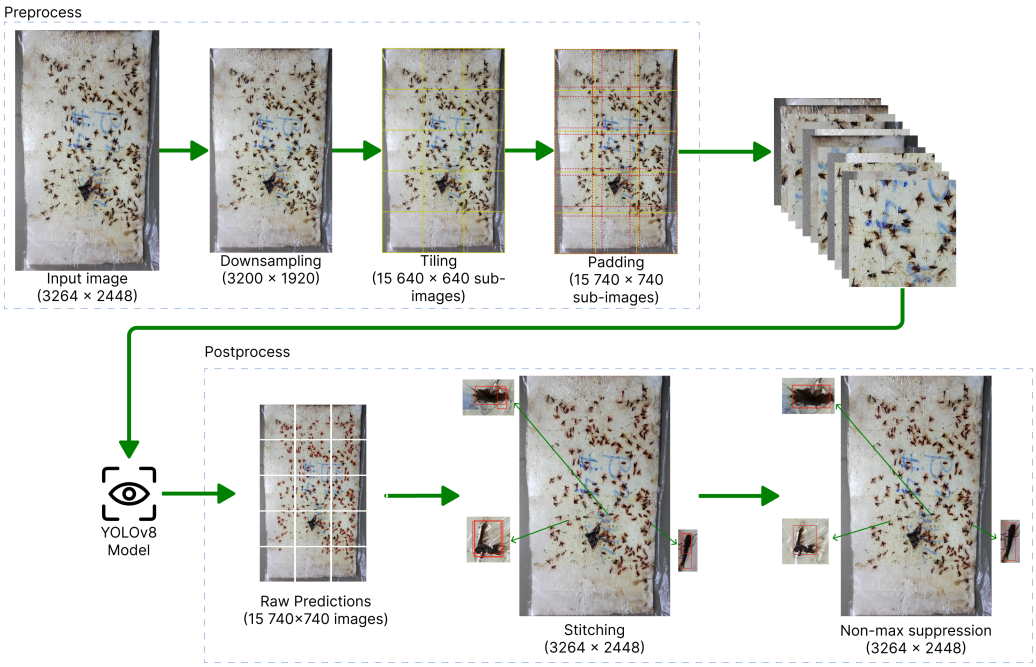



**Figure 3.** Automated CPB detection algorithm.

First, downsampling is applied to the original image of 3264 × 2448 px (relative to the Realme C30) to the nearest resolution divisible by 640 px, resulting in a pre-tiling shape of 3200 × 1920 px. This step is carried out in order to make the original resolution a divisible

size according to the model's input of 640 × 640 px. Tiling is applied to minimize the downsampling factor and improve the overall performance to detect small objects [17,18]. This produces 15 tiled images, with a size of 640 × 640 px each. Finally, each tiled image is padded with 100px on each side, resulting in a final 740 × 740 px resolution. This reinforces the model's learning ability by having overlapping tiles in which an object might be more partially visible in one tile than the other; this gives it better context on how each object is spatially related.

The object detector model used in this work is You Only Look Once version 8 (YOLOv8). YOLOv8 is a one-stage model that performs a single pass on input images [19], making it lightweight in contrast with two-stage models such as the RCNN family [20,21]. The fundamental components of YOLOv8 include the following: backbone, bottleneck, and detection head [22]. The backbone is a modified Cross Stage Partial (CSP) Network Darknet used for extracting high-dimensional features from input images. The bottleneck receives these features and learns the relation between low-level and high-level features at various scales through a pyramid feature network (FPN). This is especially useful in making the model robust to varying object scales due to camera distance [23,24]. The best representative features of the target object are passed to the YOLO head made of dense and convolutional layers to predict coordinates and classes.

The raw output of the YOLOv8 model has coordinates relative to the padded input images (740 × 740 px). Stitching is used to compile all model detections and translate it relative to the original image resolution. The excess bounding boxes due to tiling are filtered using non-max suppression (NMS). NMS works by removing bounding boxes that have an intersection-over-union (IoU), as shown in Equation (1), values above a pre-defined threshold and keeps only the one with the highest confidence. The pre-defined threshold was determined through model-centric optimizations discussed in a later section.

$$\text{IoU} = \frac{\text{area}(B_1 \cap B_2 \cap \ldots B_n)}{\text{area}(B_1 \cup B_2 \cup \ldots B_n)} \quad (1)$$

### *2.4. Algorithm Optimization*

The optimization pipeline in Figure 4 shows the different variables optimized to maximize algorithm performance. The description of each optimization step in Figure 4 is shown in the succeeding sub-sections.

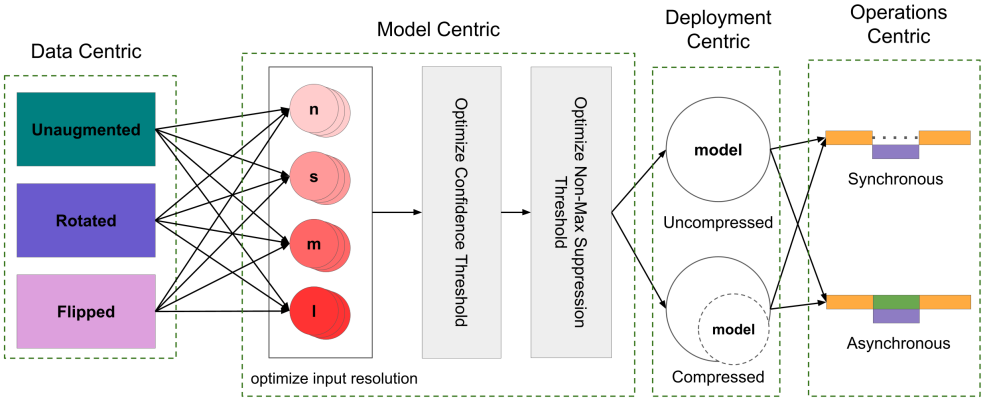



**Figure 4.** Optimization pipeline.

#### 2.4.1. Data-Centric Optimization

Initially, the model was optimized based on data by controlling the number of training images using different augmentation operations. Image augmentation is a technique that improves model invariance and model generalizability by increasing dataset size based on specific image features [25]. All augmentations that were applied to the original images were based on geometric features, such as by translating object orientation, to give

the model extra learnable features while preserving the image's color space. This type of augmentation is preferred over color-based, as it provides an overall better model invariance during training [26]. Firstly, the training images were rotated 90°, 180°, and 270° to change each object's angular orientation. Secondly, flipping was applied to mirror the image horizontally, vertically, and both. Applying these geometrical operations generates two new datasets that were used to see how the lack or presence of these additional features (rotated and flipped objects) affect model performance, as shown in Table 3. Meanwhile, the testing dataset was retained and were independent of the augmentations applied on the training and validation.

**Table 3.** Augmented dataset statistical summary.

| Augmentation Method | Training | Validation | Testing |
|---|---|---|---|
| None | 389 | 99 | 112 |
| Rotation: 90°, 180°, 270° | 1556 | 396 | 112 |
| Flip: Horizontal, Vertical, Origin | 1556 | 396 | 112 |

#### 2.4.2. Model-Centric Optimization

Model-centric optimization aims to optimize different network structure and model hyperparameters. First, four YOLOv8 network structure variants based on increasing learnable parameters were tested: nano (n) (3.2 million), small (s) (11.2 million), medium (m) (25.9 million), and large (l) (43.7 million). Each variant was tested using different input resolutions: 640 × 640 px (default pre-trained resolution) and 320 × 320 px. The former serves as the baseline against which the latter is compared in terms of accuracy and execution speed in mobile devices; it also aims to test the plausibility of reducing input image size for better runtime on low-end phones without sacrificing performance.

The trained models were evaluated using an independent dataset and evaluated through a range of confidence threshold ($conf_{thres}$) values: 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, and NMS threshold ($NMS_{thres}$) values: 0.2, 0.3, 0.4, 0.5, 0.6. The $conf_{thres}$ was used to decide the minimum value of how confident the model was in its detections to be classified as a true positive. Meanwhile, $NMS_{thres}$ was used to filter redundant detections due to tiling.

#### 2.4.3. Deployment-Centric Optimization

Major bottlenecks in deploying deep learning models are limited mobile phone memory and compatibility. Deployment-centric optimization focuses on converting the trained model to a compatible format for mobile devices, while minimizing performance degradation. In so doing, compression was applied by reducing the floating-point precision of the model weights; this is accomplished by converting the uncompressed (PyTorch) model to Open Neural Network Exchange (ONNX) runtime (ORT) format by replacing selected mathematical operations in the former model to custom ones. This operation also reduces the memory requirement and power consumption when deployed on mobile phones [27]. An uncompressed PyTorch model served as a point of comparison against two of its derivatives: retained floating-point 32 precision (fp32) and halving of it to floating-point 16 precision (fp16), with both types of models converted to ONNX format. This method ensures that there is minimal decrease in performance compared to INT quantization, while decreasing the computational load [28]. Furthermore, it improves the mobile application's compatibility for low-end mobile phones.

#### 2.4.4. Operations-Centric Optimization

Operations-centric optimization aims to boost computation speed by using different types of processing operations. Two operations were tested: synchronous and asynchronous. Figure 5 shows which processing operations were applied for each step of the algorithm. Three steps of the algorithm were processed asynchronously: tiling, padding, and model inference. Tiling and padding operations were processed asynchronously by pre-

computing the start and end coordinates of each tile. Algorithm 1 shows the pseudo-code on how the asynchronous processing of tiling and padding was improved for memory-constrained devices. For example, the initial height (H0) and width (W0) were set to Realme C30's raw resolution. Image data between these coordinates were then obtained and passed on to the next step. Encoding from image data to tensors was kept synchronous to make it run on low-end phones with limited memory, since it requires allocation of random-access memory (RAM) proportional to the number and shape of tiles. The pre-computed start and end coordinates of each tile were re-used for stitching and NMS to translate all detections relative to the original image resolution. This saves time by avoiding the cost of recomputing these values. Lastly, model inference was also made asynchronous, as each input was independent from other tiles.

**Algorithm 1:** Asynchronous preprocessing, detection, and post-processing in the mobile application

```
Inputs    : H0 : original height of the image
            W0 : original width of the image
            tileSize : size for tiling the image
            overlapSize : size of intersection between tiles for each side
Outputs   : finalOutputs : final detected objects after NMS
Functions : AsyncProcessTile(originX, originY, padSize) : asynchronously extract tile at
            (originX, originY) then add padding
            AsyncInferenceModel(tile) : performs asynchronous model inference on tile
            StitchDetections(outputs, originX, originY) : stitches detections based on their
            origin
            PerformNMS(allOutputs) : performs Non-Maximum Suppression on all outputs
H0 ← 3264; W0 ← 2448; tileSize ← 640; overlapSize ← 50;
H ← H0 − (H0%tileSize); W ← W0 − (W0%tileSize);
rows ← []; cols ← [];
for i ← 0 to H/tileSize do
    rows.append(i × tileSize);
for i ← 0 to W/tileSize do
    cols.append(i × tileSize);
padSize ← 2 × overlapSize + tileSize; tilesPromises ← [];
foreach originX in rows do
    foreach originY in cols do
        tilesPromises.append(AsyncProcessTile(originX, originY, padSize));
tilesArray ← tilesPromises when finished executing asynchronously;
outputsPromises ← [];
foreach tile in tilesArray do
    outputsPromises.append(AsyncInferenceModel(tile));
outputs ← outputsPromises when finished executing asynchronously;
allOutputs ← [];
for index ← 0 to length of outputs do
    originX ← rows [index/length of cols]; originY ← cols [index%length of cols];
    stitchedOutput ← StitchDetections(outputs [index ], originX, originY);
    allOutputs.append(stitchedOutput);
finalOutputs ← PerformNMS(allOutputs);
```

### 2.5. Algorithm Evaluation

The performance of each trained model was evaluated using four metrics: precision, recall, $F_1$-score, and MAPE. Precision Equation (2) checks the model's ability to predict how many CPBs are correctly detected, compared to the number of detected CPBs in each image [29]. Meanwhile, recall Equation (3) measures how many actual positives were correctly identified by the model [29]. Precision and recall are measured from 0 to 1.0, where values closer to 1.0 indicate better performance.

$$\text{Precision} = \frac{TP}{\text{number of detected CPB objects}} \tag{2}$$

$$\text{Recall} = \frac{TP}{\text{actual number of CPB images}} \tag{3}$$

where

$TP$ = true positive detections

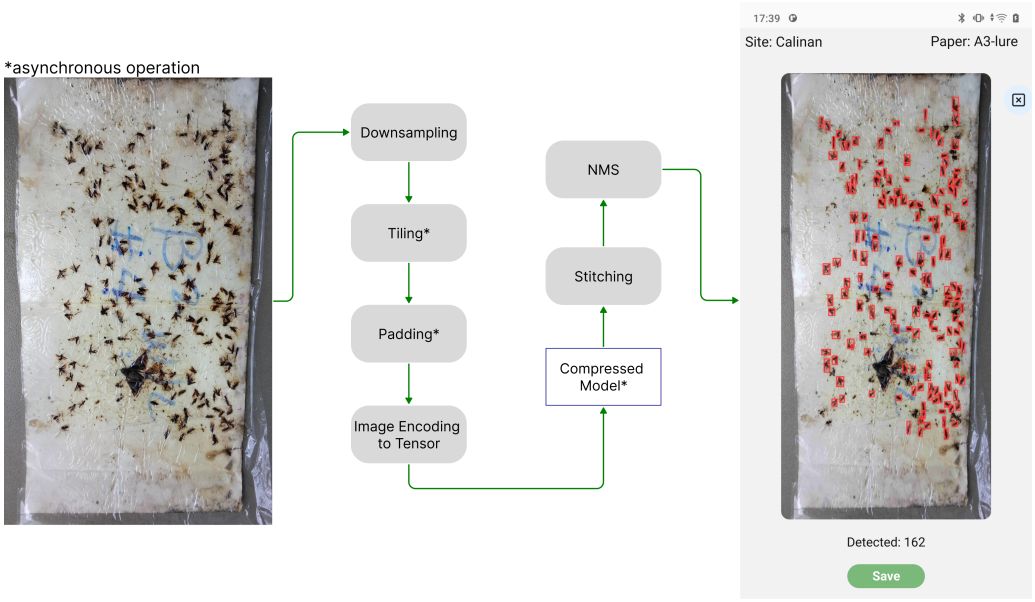



**Figure 5.** Mobile application flowchart. Each process marked with a * was performed asynchronously.

$F_1$-score Equation (4) incorporates both precision and recall, forming a single metric through harmonic mean [30]. This metric penalizes the model if it has high false positives or false negatives, making it a more robust measure of performance.

$$F_1 \text{ score} = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}} \tag{4}$$

Mean absolute percentage error (MAPE) [31] in Equation (5) is the performance loss due to model compression by evaluating the difference in the number of detected CPBs between the compressed (16-bit) and uncompressed (32-bit) model.

$$\text{MAPE} = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{A_i - P_i}{A_i}\right| \times 100\% \tag{5}$$

where

$A_i$ = number of predictions from a 32-bit uncompressed model
$P_i$ = number of predictions from a 16-bit compressed model

Both $F_1$-score and MAPE provide a holistic evaluation in obtaining the best uncompressed model's configuration (dataset-augmentation, model variant, input size, $\text{conf}_{\text{thres}}$, and $\text{NMS}_{\text{thres}}$) when detecting CPBs and how compression affects it.

Linear regression was used to assess the correlation between the predicted and manual CPB counts, determining the viability of the algorithm as an alternative counting method. A two-tailed $t$-test ($\alpha$ = 0.05) was used to evaluate whether the predicted CPB count significantly differs from the manual CPB count. These statistical analyses were performed using Python libraries: SciPy [32] and Pandas [33].

### 2.6. Mobile Application Development and Usage

The optimized algorithm was deployed in a mobile application. Screenshots of the mobile application are shown in Figure 6. The algorithm processes images of sticky paper traps captured by farmers and experts, identifying the number and location of CPBs present in each image. This information allows users to assess the severity of infestation. Users then input additional data such as sticky paper ID, site information, and collection date, which are stored in a database along with the extracted CPB count. Both components operate on-device, which offers cost savings by eliminating the need for a centralized database and internet connectivity, which is most especially beneficial in remote areas. Furthermore, the mobile application aims to reduce the manpower and time required for data processing by automating these tasks. Rapid access to information about insect severity at specific sites provides an advantage by enabling prompt responses to CPB severity without special hardware requirements other than a compatible phone. This can help in reducing potential crop losses and facilitating decisions based on quantitative data. The mobile application was developed using TypeScript v5.2.2, React Native v0.72.3, Expo v6.3.7, and SQLite v3.36.0 to ensure cross-platform compatibility.

The mobile application can be installed on a mobile phone by request. While the minimum specifications of the mobile phones are shown in Table 1. The steps to use the application are as follows: first, the user clicks the "camera" button. Second, the phone is positioned from the sticky paper trap at the recommended distance in Figure 2. Third, the user clicks the capture button and enters the necessary details related to it (Site, Paper, Pheromone, and Date). Lastly, Algorithm 1 is performed and the results are saved in the local database for viewing, as shown in Figure 6.

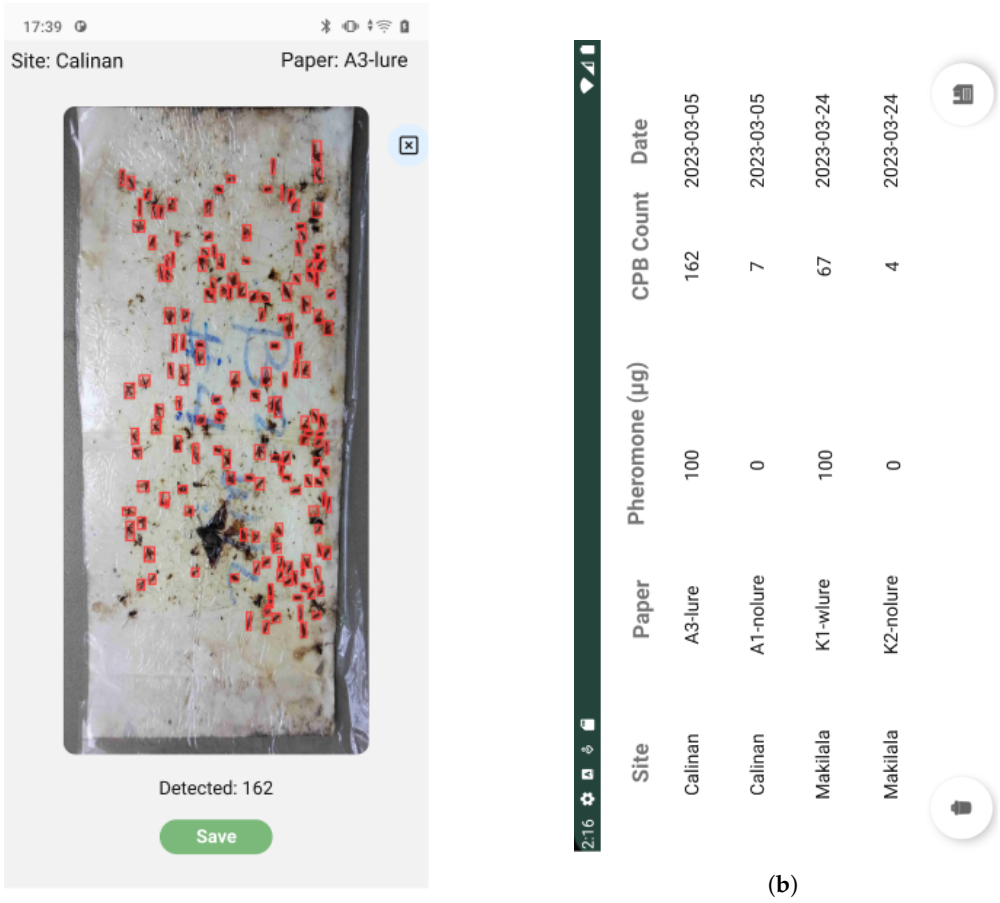



**Figure 6.** Mobile application user interface for automated CPB counting and database. (**a**) Sample mobile application algorithm output, (**b**) Database output.

## 3. Results and Discussion

### *3.1. Algorithm Optimization Results*

#### 3.1.1. Data-Centric Optimization

The $F_1$-scores obtained after applying augmentations on 640 × 640 px tiled images showed improvements in model performance. It was observed that the model trained with rotation augmentation had consistently better performance ($F_1$-score: 0.81) than when flipping ($F_1$-score: 0.80) or no-augmentation ($F_1$-score: 0.79) was applied. Similar $F_1$-scores were measured when 320 × 320 px tiled images were used, in all augmentation cases. This means that reducing the input image size did not have a detrimental effect on model performance. Based on the results obtained, the models trained with rotation augmentation were used for testing in the next optimization steps.

#### 3.1.2. Model-Centric Optimization

Different model hyperparameters including input resolution, network size, $conf_{thres}$, and $NMS_{thres}$, were iteratively tested, as shown in Table 4.

**Table 4.** Best model-centric optimization results.

| Model Size | Input Size (px) | $conf_{thres}$ | $NMS_{thres}$ | $F_1$-Score |
|---|---|---|---|---|
| Nano | 320 × 320 | 0.2 | 0.2 | 0.86 |
| | 640 × 640 | 0.3 | 0.2 | 0.86 |
| Small | 320 × 320 | 0.2 | 0.2 | 0.88 |
| | 640 × 640 | 0.2 | 0.2 | 0.88 |
| Medium | 320 × 320 | 0.2 | 0.2 | 0.88 |
| | 640 × 640 | 0.2 | 0.2 | 0.87 |
| Large | 320 × 320 | 0.2 | 0.2 | 0.88 |
| | 640 × 640 | 0.2 | 0.2 | 0.89 |

The trained nano YOLOv8 model showed that increasing the $NMS_{thres}$ and $conf_{thres}$ resulted in worse $F_1$-scores for both input resolutions. Therefore, a value of 0.2 was used for both thresholds, while using input resolutions of 320 × 320 px and 640 × 640 px had the same best $F_1$-scores of 0.86. The results showed that most detected objects, using the nano YOLOv8 model, have low confidence scores. Therefore, the nano YOLOv8 models do not have enough learnable parameters to capture the more complex relationship of the features in the dataset. On the other hand, the small YOLOv8 model achieved a better $F_1$-score of 0.88 using the minimum $NMS_{thres}$ and $conf_{thres}$, indicating an improvement in detection performance due to having more learnable parameters regardless of input size. For medium models, a higher input size leads to a small decrease in $F_1$-score. The model performance of the large YOLOv8 model converged at an $F_1$-score of 0.89, indicating that the extra features provided by rotation were sufficiently captured starting with the small model.

Based on the testing results, it was concluded that the small YOLOv8 model with 320 × 320 px input size contains the optimal amount of learnable parameters, which leads to faster processing time due to less resource requirement, and was therefore used for model deployment with $conf_{thres}$ of 0.2 and $NMS_{thres}$ of 0.2. It was found that increasing the model size does not provide much improvement in performance, which converged at an $F_1$-score of 0.89. Furthermore, the majority of the best $F_1$-scores were obtained at $conf_{thres}$ and $NMS_{thres}$ of 0.2 in all input resolutions. This suggested that most detections have low confidence scores and bounding boxes with an IoU of at least 0.2 were removed. This was attributed to tiling and padding due to redundant bounding boxes in the overlapping regions. Another reason why these two variables do not have much influence on the model's performance was due to feature drift, in which the images between the training and testing dataset vary caused by temporal difference. This can be mitigated by testing for color-based augmentations and using a more recent dataset for training.

#### 3.1.3. Deployment-Centric Optimization

The results of model conversion for cross-platform deployment are shown in Table 5. Using an uncompressed fp32 model's number of predictions as baseline, Table 5 shows that compression increases model size from 22.5 MB to 44.7 MB at fp32 precision. This was due to the large metadata embedded in the model to make it cross-platform compatible for edge devices. Such metadata include the following: graph structure, node connections, and layer information in the model that ORT uses to dynamically execute and optimize based on the edge-device on which it is deployed [34]. Furthermore, the metadata were also used to interact with the execution environment by knowing the device's memory and processing power constraints.

**Table 5.** MAPE-based performance loss due to model compression.

| Model | Precision | Compressed | Size | MAPE |
|---|---|---|---|---|
| YOLOv8-s | fp32 | No | 22.5 MB | Baseline |
| YOLOv8-s | fp32 | Yes | 44.7 MB | 3.48% |
| **YOLOv8-s** | **fp16** | **Yes** | **22.6 MB** | **3.31%** |

Both compressed models achieved a low MAPE of less than 5%, indicating that their predictions closely follow the uncompressed one. Surprisingly, decreasing the precision to fp16 led to less performance loss due to compression, even if it was a small difference. This was attributed to multiple variables, such as model robustness, in which it was able to effectively learn the objects' features; regularization effect, as precision reduction can be used as a regularization method in some cases, therefore having better generalizability; and randomness, in which reduction in precision was trivial. Given these results, the compressed model with reduced precision (fp16) was used since it has a smaller storage size and closest detection performance relative to the baseline model.

#### 3.1.4. Operations-Centric Optimization

Table 6 shows the results of testing different input sizes and processing operations using the Realme C30 mobile phone.

**Table 6.** Algorithm performance metrics as tested on a Realme C30 mobile phone.

| Model | Input Size (px) | Operation | Avg. Computation Time |
|---|---|---|---|
| **YOLOv8-s** | **320 × 320** | **Asynchronous** | **33.5 s (*7.24 × faster)*** |
| YOLOv8-s | 640 × 640 | Asynchronous | 113.70 s *(2.15 × faster)* |
| YOLOv8-s | 320 × 320 | Synchronous | 158.2 s *(1.54 × faster)* |
| YOLOv8-s | 640 × 640 | Synchronous | 243.8 s *(baseline)* |

Using the small YOLOv8 model with 640 × 640 px input size and synchronous operations as baseline, Table 6 shows that reducing the image input size to 320 × 320 px alone improves the execution speed to 158.2 s. Meanwhile, solely using asynchronous operations decreased the execution speed to 113.7 s. Lastly, an execution speed of 33.5 s was achieved by utilizing both input size downscaling and asynchronous operations. It was concluded that the use of 320 × 320 px as input size on the small YOLOv8 model both maximizes the execution speed and detection performance in low-end mobile phones. This was because the detection performance of 320 × 320 px and 640 × 640 px models did not differ much while the former provided 1.54 × to 7.24× speed-up on execution. The results proved that the optimization efforts made it possible to execute the algorithm in a low-end mobile phone.

### *3.2. Qualitative Algorithm Evaluation*

A sample output from the best configuration of the small YOLOv8 model is shown in Figure 7. The algorithm was able to detect most CPBs in the sticky paper trap even

if they were composed of both fresh and decayed ones. Furthermore, it was also able to avoid classifying other insects as a CPB even if they look similar (True Negative). However, the algorithm missed CPBs that overlap each other, while falsely detected non-CPBs with similar outlines from those that are. Both false positive and false negative detections suggested that the model heavily relied on the learned outlines of CPBs. Despite that, the algorithm still remained robust against these types of errors especially in high density sticky papers. A low $NMS_{thres}$ also solved the redundant detections due to tiling from our previous research [35].

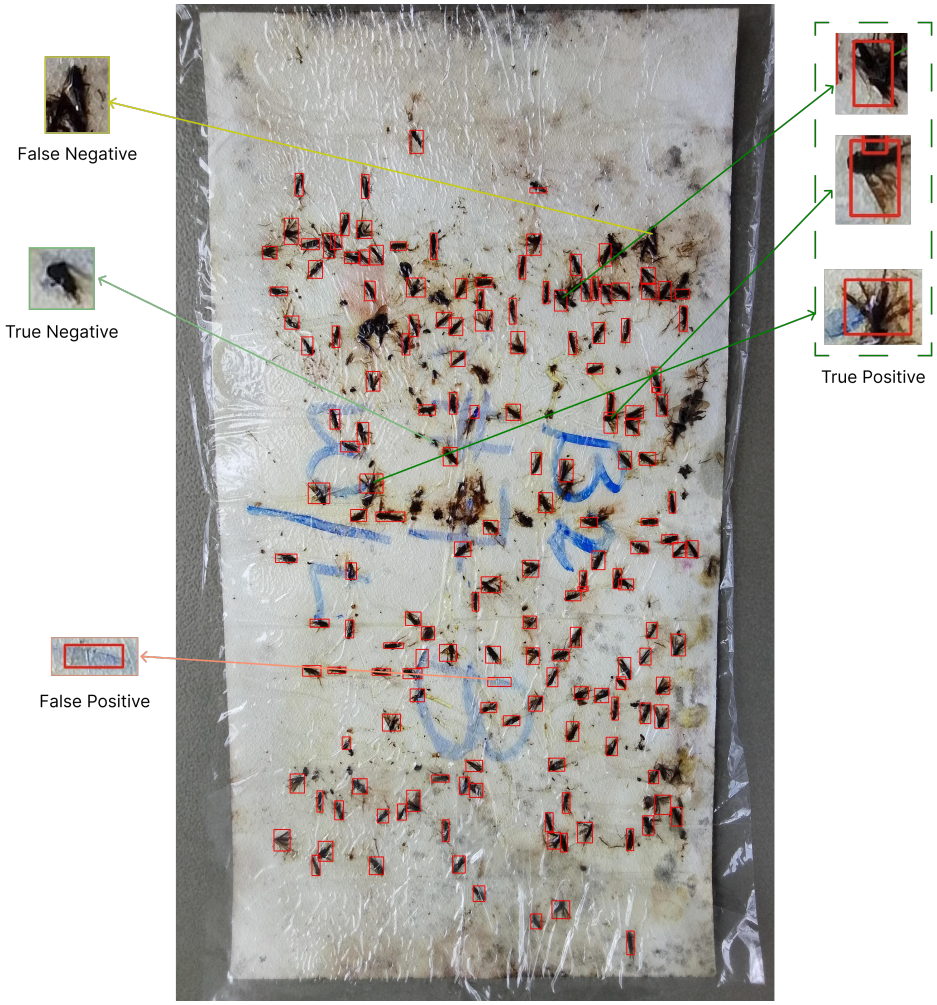
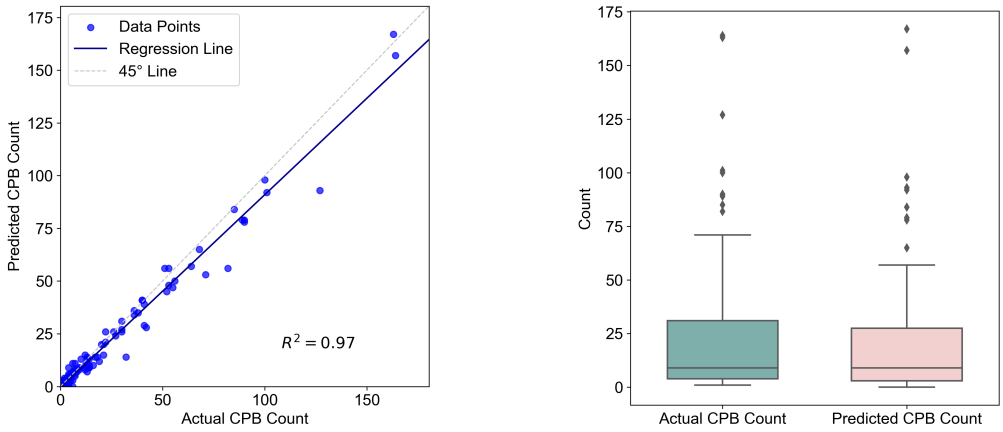



**Figure 7.** Sample optimized algorithm results.

### *3.3. CPB Count Comparison*

Figure 8a shows that the best model has an $R^2$ of 0.97 with the manually annotated test dataset. It was presented that the algorithm had a strong correlation with sticky paper traps containing up to 60 CPBs. Meanwhile, it starts to slightly underestimate the CPB count for sticky paper traps with more than 60 CPBs. This was most likely due to CPBs that overlap each other. Nevertheless, the two-tailed t-test showed that the model's count has no significant difference with the actual count (t-statistic = 0.60, $p$-value = 0.55, $\alpha$ = 0.05, N = 112, degrees of freedom = 110). Figure 8b further validated these findings, showing that the actual count and predicted count had comparable median, variability, and spread. Finally, it is also more important for the model to detect CPBs from sticky paper traps with low and medium densities since it was more important for implementing IPM strategies.




(**a**) (**b**)

**Figure 8.** Statistical analysis of actual CPB count vs predicted CPB count. (**a**) Correlation between actual CPB count vs. predicted CPB count, (**b**) Box plot for actual CPB count vs. predicted CPB count.

### *3.4. Field Use and Cost-Benefit Analysis*

Figure 9 illustrates the integration of the mobile application into the overall process of data collection and insect control.

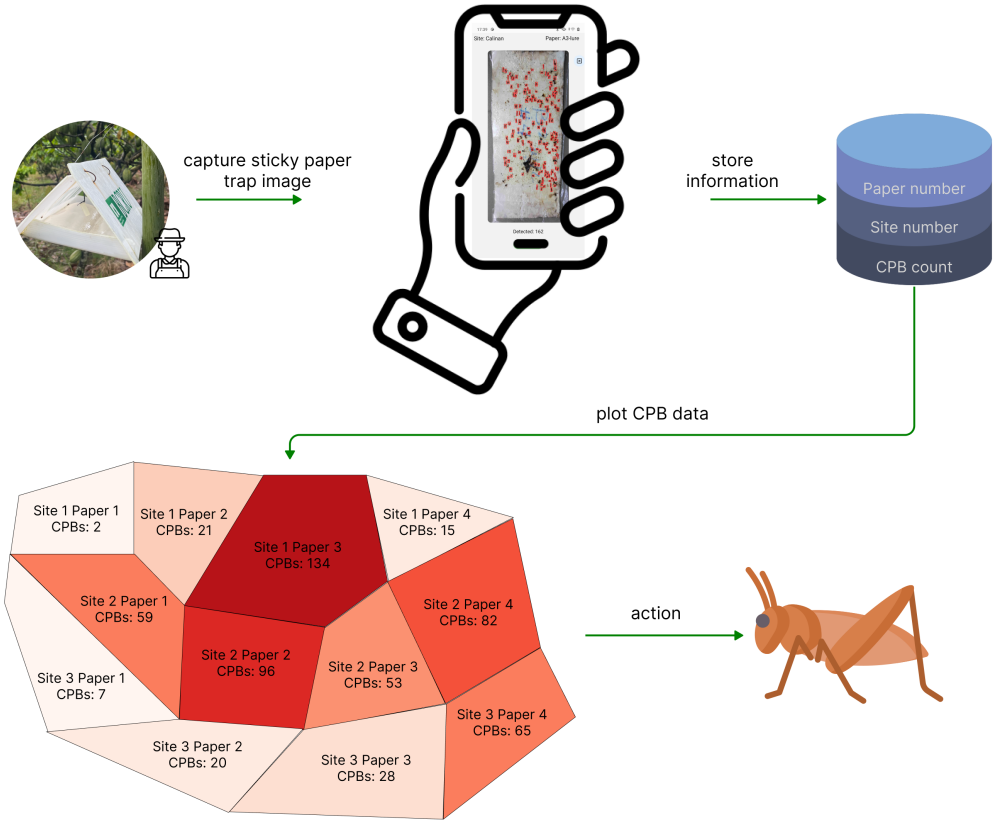



**Figure 9.** Field use diagram.

Farm managers and experts can visit various cacao plantations to assess the severity of CPB infestations. By storing critical information such as CPB counts, sticky paper trap images, and site data on the device, the application reduces the time required to take necessary actions against CPBs. This leads to a decrease in the manpower needed for field scouting and a reduction in human error, which is particularly beneficial for medium to high-density sticky paper traps where it takes an expert an average of 147 s to manually

count or annotate a trap. Lastly, the mobile application streamlines the consolidation of these data to the experts, enabling efficient communication even in the absence of on-site presence. This functionality diminishes the delay traditionally experienced between on-site diagnosis and subsequent actions.

## 4. Conclusions

In this work, a mobile application was designed to assist farm managers and experts in the monitoring and management of CPB infestation in CPB plantations. The proposed algorithm demonstrated robustness for real-world applications, accurate detection of CPBs, and showed a high correlation with expert counts in tests using recently collected sticky papers. Further experiments also indicated minimal degradation in detection performance when the application was deployed on low-end mobile phones. This application provides on-demand, automated pest count information, facilitating efficient pest control methods. The mobile application is accessible for low-income farms and remote locations without internet access, due to its compatibility with low-end devices and on-device processing capabilities. In the future, the mobile application shall be made available for other interested users for implementing advanced IPM programs.

**Author Contributions:** Conceptualization, D.J.A.R. and D.A.; methodology, E.A.S.H. and D.J.A.R.; software, E.A.S.H. and L.S.Q.; data curation, K.L.T.S., E.B.M., M.A. and R.C.C.; writing—original draft preparation, D.J.A.R., E.A.S.H. and K.L.T.S.; writing—review and editing, D.J.A.R.; visualization, E.A.S.H.; supervision, D.J.A.R.; project administration, D.A. All authors have read and agreed to the published version of the manuscript.

**Funding:** This work was funded by the Department of Science and Technology (DOST), Philippine Council for Agriculture, Aquatic, and Natural Resources Research and Development (PCAARRD) (grant number QMSR-FERD-CACAO-541-0), and through the DOST Balik Scientist Program.

**Data Availability Statement:** The data that support the findings of this study are available from the corresponding author, D.J.A.R., upon reasonable request.

**Acknowledgments:** The authors would like to thank the staff members of the Philippine Bureau of Plant Industry for providing the experimental sites.

**Conflicts of Interest:** The authors declare no conflict of interest.

## References

1. Amalin, D.M.; Arcelo, M.; Almarinez, B.J.M.; Castillo, R.C.; Legaspi, J.C.; Santos, K.L.T.; Tavera, M.A.A.; Janairo, J.I.B.; Zhang, A. Field evaluation of the sex pheromone of the cacao pod borer (*Conopomorpha cramerella snellen*) in the Philippines. *Front. Agron.* **2023**, *5*, 1165299. [CrossRef]
2. Shapiro, L.H.; Scheffer, S.J.; Maisin, N.; Lambert, S.; Purung, H.B.; Sulistyowati, E.; Vega, F.E.; Gende, P.; Laup, S.; Rosmana, A.; et al. *Conopomorpha cramerella (Lepidoptera: Gracillariidae)* in the malay archipelago: Genetic signature of a bottlenecked population? *Ann. Entomol. Soc. Am.* **2008**, *101*, 930–938. [CrossRef]
3. *Conopomorpha Cramerella* (Cocoa pod Borer) | CABI Compendium. Available online: https://www.cabidigitallibrary.org/doi/10.1079/cabicompendium.7017 (accessed on 25 January 2024) . [CrossRef]
4. Teh, C.L.; Pang, J.T.Y.; Ho, C.T. Variation of the response of clonal cocoa to attack by cocoa pod borer *Conopomorpha cramerella (Lepidoptera: gracillariidae)* in Sabah. *Crop Prot.* **2006**, *25*, 712–717. [CrossRef]
5. Beevor, P.S.; Mumford, J.D.; Shah, S.; Day, R.K.; Hall, D.R. Observations on pheromone-baited mass trapping for control of cocoa pod borer, *Conopomorpha cramerella*, in Sabah, East Malaysia. *Crop Prot.* **1993**, *12*, 134–140. [CrossRef]
6. Vanhove, W.; Vanhoudt, N.; Bhanu, K.R.M.; Abubeker, S.; Feng, Y.; Yu, M.; Van Damme, P.; Zhang, A. Geometric isomers of sex pheromone components do not affect attractancy of *Conopomorpha cramerella* in cocoa plantations. *J. Appl. Entomol.* **2015**, *139*, 660–668. [CrossRef]
7. Beevor, P.S.; Cork, A.; Hall, D.R.; Nesbitt, B.F.; Day, R.K.; Mumford, J.D. Components of female sex pheromone of cocoa pod porer moth, *Conopomorpha cramerella*. *J. Chem. Ecol.* **1986**, *12*, 1–23. [CrossRef] [PubMed]
8. Rossi, V.; Sperandio, G.; Caffi, T.; Simonetto, A.; Gilioli, G. Critical success factors for the adoption of decision tools in IPM. *Agronomy* **2019**, *9*, 710. [CrossRef]
9. Zhang, A.; Kuang, L.F.; Maisin, N.; Karumuru, B.; Hall, D.R.; Virdiana, I.; Lambert, S.; Bin Purung, H.; Wang, S.; Hebbar, P. Activity evaluation of cocoa pod borer sex pheromone in cacao fields. *Environ. Entomol.* **2008**, *37*, 719–724. [CrossRef]

10. Rustia, D.J.A.; Chiu, L.Y.; Lu, C.Y.; Wu, Y.F.; Chen, S.K.; Chung, J.Y.; Hsu, J.C.; Lin, T.T. Towards intelligent and integrated pest management through an AIoT-based monitoring system. *Pest Manag. Sci.* **2022**, *78*, 4288–4302. [CrossRef]
11. Liu, H.; Chahl, J.S. Proximal detecting invertebrate pests on crops using a deep residual convolutional neural network trained by virtual images. *Artif. Intell. Agric.* **2021**, *5*, 13–23. [CrossRef]
12. Karar, M.E.; Alsunaydi, F.; Albusaymi, S.; Alotaibi, S. A new mobile application of agricultural pests recognition using deep learning in cloud computing system. *Alex. Eng. J.* **2021**, *60*, 4423–4432. [CrossRef]
13. Chen, J.W.; Lin, W.J.; Cheng, H.J.; Hung, C.L.; Lin, C.Y.; Chen, S.P. A smartphone-based application for scale pest detection using multiple-object detection methods. *Electronics* **2021**, *10*, 372. [CrossRef]
14. Gonçalves, J.; Silva, E.; Faria, P.; Nogueira, T.; Ferreira, A.; Carlos, C.; Rosado, L. Edge-compatible deep learning models for detection of pest outbreaks in viticulture. *Agronomy* **2022**, *12*, 3052. [CrossRef]
15. StatCounter. Most Popular Phone Brands in the Philippines. 2023. Available online: https://www.statista.com/statistics/938806/philippines-market-share-of-leading-mobile-brands// (accessed on 25 January 2024).
16. Dwyer, B.; Nelson, J.; Solawetz, J. Roboflow (Version 1.0). Software Available from Roboflow. Computer Vision. 2022. Available online: https://roboflow.com/ (accessed on 28 January 2024).
17. Rustia, D.J.A.; Chao, J.J.; Chiu, L.Y.; Wu, Y.F.; Chung, J.Y.; Hsu, J.C.; Lin, T.T. Automatic greenhouse insect pest detection and recognition based on a cascaded deep learning classification method. *J. Appl. Entomol.* **2021**, *145*, 206–222. [CrossRef]
18. Unel, F.O.; Ozkalayci, B.O.; Cigla, C. The power of tiling for small object detection. In Proceedings of the 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), Long Beach, CA, USA, 16–17 June 2019; pp. 582–591. [CrossRef]
19. Cui, M.; Gong, G.; Chen, G.; Wang, H.; Jin, M.; Mao, W.; Lu, H. LC-YOLO: A lightweight model with efficient utilization of limited detail features for small object detection. *Appl. Sci.* **2023**, *13*, 3174. [CrossRef]
20. Saleem, M.H.; Velayudhan, K.K.; Potgieter, J.; Arif, K.M. Weed identification by single-stage and two-stage neural networks: a study on the impact of image resizers and weights optimization algorithms. *Front. Plant Sci.* **2022**, *13*, 850666. [CrossRef]
21. Zhang, D.; Zhang, W.; Li, F.; Liang, K.; Yang, Y. PNANet: Probabilistic two-stage detector using pyramid non-local attention. *Sensors* **2023**, *23*, 4938. [CrossRef]
22. Jocher, G.; Chaurasia, A.; Qiu, J. Ultralytics YOLO (Version 8.1.18). Software Available from GitHub. Computer Vision. 2023. Available online: https://github.com/ultralytics/ultralytics (accessed on 25 January 2024).
23. Lin, T.Y.; Dollár, P.; Girshick, R.; He, K.; Hariharan, B.; Belongie, S. Feature pyramid networks for object detection. *arXiv* **2017**, arXiv:1612.03144.
24. Liu, L.; Ouyang, W.; Wang, X.; Fieguth, P.; Chen, J.; Liu, X.; Pietikäinen, M. Deep learning for generic object detection: A survey. *Int. J. Comput. Vision* **2020**, *128*, 261–318. [CrossRef]
25. Xu, M.; Yoon, S.; Fuentes, A.; Park, D.S. A comprehensive survey of image augmentation techniques for deep learning. *Pattern Recognit.* **2023**, *137*, 109347. [CrossRef]
26. Shorten, C.; Khoshgoftaar, T.M. A survey on image data augmentation for deep learning. *J. Big Data* **2019**, *6*, 60. [CrossRef]
27. ONNX Runtime. Model Optimizations. Available online: https://onnxruntime.ai/docs/performance/model-optimizations/ (accessed on 16 January 2024).
28. Intel. Choose FP16, FP32 or Int8 for Deep Learning Models. Available online: https://www.intel.com/content/www/us/en/developer/articles/technical/should-i-choose-fp16-or-fp32-for-my-deep-learning-model.html (accessed on 16 January 2024).
29. Padilla, R.; Netto, S.; da Silva, E. *A Survey on Performance Metrics for Object-Detection Algorithms*. In Proceedings of the 2020 International Conference on Systems, Signals and Image Processing (IWSSIP), Online, 1–3 July 2020. [CrossRef]
30. Alfonso-Francia, G.; Pedraza-Ortega, J.C.; Badillo-Fernández, M.; Toledano-Ayala, M.; Aceves-Fernandez, M.A.; Rodriguez-Resendiz, J.; Ko, S.B.; Tovar-Arriaga, S. Performance Evaluation of Different Object Detection Models for the Segmentation of Optical Cups and Discs. *Diagnostics* **2022**, *12*, 3031. [CrossRef] [PubMed]
31. Kim, S.; Kim, H. A New Metric of Absolute Percentage Error for Intermittent Demand Forecasts. *Int. J. Forecast.* **2016**, *32*, 669–679. [CrossRef]
32. Virtanen, P.; Gommers, R.; Oliphant, T.E.; Haberland, M.; Reddy, T.; Cournapeau, D.; Burovski, E.; Peterson, P.; Weckesser, W.; Bright, J.; et al. SciPy 1.0: Fundamental algorithms for scientific computing in python. *Nat. Methods* **2020**, *17*, 261–272. [CrossRef]
33. Pandas Development Team. Pandas-Dev/Pandas: Pandas. Available online: https://pandas.pydata.org (accessed on 25 January 2024). [CrossRef]
34. Model Metadata Struct Reference. Available online: https://onnxruntime.ai/docs/api/c/struct_ort_1_1_model_metadata.html (accessed on 15 January 2024).
35. Hacinas, E.A.S.; Querol, L.S.; Acero, L.A.; Arcelo, M.; Amalin, D.M.; Rustia, D.J.A. *Automated Cocoa Pod Borer Detection Using an Edge Computing-Based Deep Learning Algorithm*; American Society of Agricultural and Biological Engineers: St. Joseph Charter Township, MI, USA, 2022. [CrossRef]