HARMFUL INSECT CLASSIFICATION AND ALERT SYSTEM FOR RICE FIELDS USING IMAGE PROCESSING

Winqwist Arthur[1,2]*

[1]Polytech Marseille, 142 Rue Henri Poincaré, Marseille, 13013, France
[2]Danang University of Science and Technology, 54 Nguyễn Lương Bằng, Hoà Khánh Bắc, Liên Chiểu, Đà Nẵng 550000, Vietnam

*Corresponding author. Email address: arthur.winqwist@etu.univ-amu.fr

Abstract

Rice cultivation faces significant challenges from pest infestations that can cause substantial yield losses. This paper presents the design, implementation, and evaluation of an intelligent harmful insect classification and alert system for rice fields using advanced computer vision techniques. The system integrates an ESP32-CAM microcontroller for autonomous image capture with a centralized server employing YOLOv11s deep learning model for pest detection and classification. Initially designed for on-device inference, practical considerations regarding computational constraints and accuracy requirements led to adopting a hybrid architecture where the ESP32-CAM captures field images and transmits them wirelessly to a Flask- based server for analysis. The system categorizes detected insects into three classes: harmful, caution, and safe, enabling farmers to make informed pest management decisions. Experimental results demonstrate the system's effectiveness with a mAP50(B) of 0.661, precision(B) of 0.663, and recall(B) of 0.643. The platform features a web dashboard with analytics capabilities. This cost-effective solution provides scalable, precise, and timely pest monitoring, significantly improving upon traditional manual inspection methods.

Keywords: Computer vision, pest detection, agricultural automation, ESP32, YOLO, deep learning, IoT, rice cultivation

1. Introduction

Rice (Oryza sativa) is one of the world's most important cereal crops, serving as a staple food for over half of the global population. With increasing food security concerns and a growing world population projected to reach 9.7 billion by 2050, maintaining and improving rice production efficiency has become critically important [3]. However, rice cultivation faces numerous challenges, with pest infestations being among the most significant threats to crop yield and quality. Traditional pest monitoring in rice fields relies heavily on manual inspection by farmers or agricultural experts, a process that is labor-intensive, time-consuming, and often inconsistent due to human factors such as experience level, visual acuity, and subjective judgment. This approach frequently results in delayed detection of pest infestations, leading to substantial economic losses that can reach up to 20-40% of total yield in severe cases [4]. The advent of artificial intelligence and computer vision technologies has opened new possibilities for automated agricultural monitoring systems. Deep learning models, particularly object detection architectures like YOLO (You Only Look Once), have demonstrated remarkable success in identifying and classifying various agricultural pests with high accuracy and speed [1]. However, deploying such sophisticated models in field conditions presents unique challenges related to computational resources, power consumption, connectivity, and environmental durability.

This research project, conducted at Danang University of Science and Technology, aimed to develop a practical, cost-effective, and scalable solution for automated insect detection in rice fields. The system addresses the critical need for early pest detection by combining Internet of Things (IoT) technology with advanced machine learning algorithms to create an intelligent monitoring platform.

The initial design concept involved deploying a lightweight YOLO model directly on an ESP32 microcontroller to enable on-device inference and reduce dependency on network connectivity. While neural networks can indeed be executed on ESP32-class devices, for example, small-scale CNNs or models optimized with frameworks such as TensorFlow Lite for Microcontrollers [16][17], preliminary testing revealed significant limitations in terms of computational capacity, memory constraints, and achievable inference accuracy when running more complex architectures such as YOLO variants. Consequently, the architecture was redesigned to leverage a hybrid approach that maintains the benefits of edge-based image capture while utilizing server-side processing for higher accuracy pest detection and classification.

The primary contributions of this work include: (1) development of a complete IoT-based pest monitoring system integrating hardware and software components, (2) implementation of a robust wireless image transmission protocol with buffering capabilities for unreliable network conditions, (3) creation of a web-based dashboard for monitoring and analytics, (4) comprehensive evaluation of the system's performance in detecting harmful insects.

2. Background and Related Work

2.1. Agricultural Pest Detection Challenges

Pest management in agriculture has evolved significantly over the past decades, transitioning from calendar-based pesticide applications to integrated pest management (IPM) strategies that emphasize monitoring, identification, and targeted interventions [5]. However, effective implementation of IPM requires accurate and timely identification of pest species and population levels, which has traditionally relied on manual sampling and visual inspection techniques. Rice cultivation is particularly vulnerable to various insect pests, including rice planthoppers, leaf rollers, stemborers, and armyworms, each requiring different management approaches. The diversity of pest species, their varying life cycles, and the need for species-specific treatments make accurate identification crucial for effective pest control.

2.2. Computer Vision in Agriculture

The application of computer vision techniques in agriculture has gained significant momentum in recent years, driven by advances in deep learning algorithms and the availability of high-quality imaging sensors. Object detection models such as Faster R-CNN, SSD (Single Shot Detector), and YOLO have been successfully adapted for various agricultural applications, including crop monitoring, disease detection, and pest identification [6]. YOLO architectures, in particular, have shown promise for fast and accurate pest detection due to their balance between accuracy and computational efficiency. Recent versions like YOLOv5, YOLOv8, and YOLOv11 have incorporated improvements in model architecture, training procedures, and inference optimization that make them particularly suitable for agricultural applications [7].

2.3. Edge Computing in Agricultural IOT

The integration of edge computing with agricultural IoT systems has emerged as a critical approach for enabling fast decision-making in field conditions. Microcontrollers such as the ESP32 family have become popular choices for agricultural sensing applications due to their low cost, built-in wireless connectivity, and reasonable computational capabilities [8]. However, deploying deep learning models on resource constrained devices presents significant challenges. Techniques such as model quantization, pruning, and knowledge distillation have been explored to reduce model complexity while maintaining acceptable accuracy levels [9].

2.4. Dataset Considerations

In order to train an advanced detection model, we needed to consider the data used so that the model was relevant to Vietnam Insect populations. After some research The PEST24 dataset, containing 24 classes of agricultural insects was chosen, the dataset was conceived in China and share a relative common insect population to Vietnam, it holds 25 000 photos pre-labeled, it has become a valuable resource for training pest detection models [2].

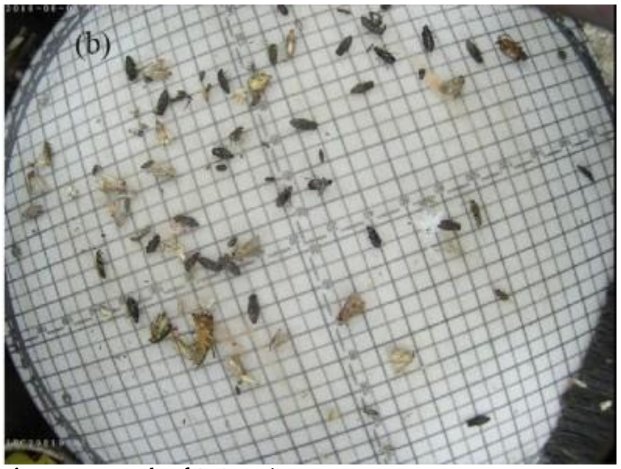

Figure 1. Example of PEST24 image

3. System Architecture and Design

The InsectDetect Pro system employs a distributed architecture designed to balance computational efficiency, reliability, and practical deployment considerations. The system consists of three primary components: field-deployed ESP32-CAM modules for image acquisition, a centralized Flask server for image processing and analysis, and a web-based dashboard for monitoring and control.

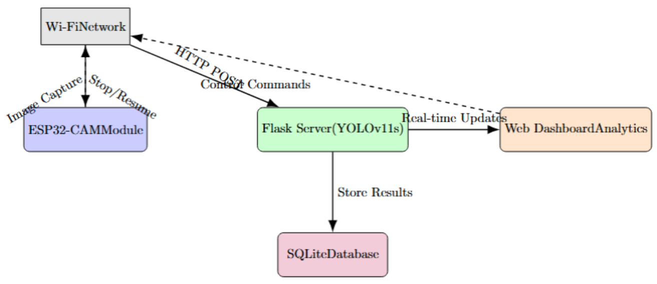

Figure 2. Comprehensive system architecture showing data flow and communication pathways

3.1. ESP32-CAM Hardware Design

The ESP32-CAM module serves as the core sensing component of the system. This module integrates a dual-core Tensilica LX6 microprocessor, 4MB of flash memory, 520KB of SRAM, and an OV2640 2-megapixel camera sensor. The built-in Wi-Fi capabilities enable seamless connectivity to local networks for image transmission. Key hardware specifications include:

- Processor: ESP32-S dual-core 32-bit LX6 microprocessor, up to 240 MHz
- Memory: 4MB Flash, 520KB SRAM, 8MB PSRAM
- Camera: OV2640 2-megapixel sensor with JPEG compression
- Wireless: 802.11 b/g/n Wi-Fi, Bluetooth v4.2
- Power consumption: 160-260mA during active operation
- Operating temperature: -40°C to +85°C

The camera configuration is optimized for field deployment with VGA resolution (640×480 pixels) to balance image quality with transmission efficiency. JPEG compression is set to quality level 12 to minimize file sizes while maintaining sufficient detail for pest detection.

3.2. Wireless Communication Protocol

The system implements a robust wireless communication protocol designed to handle intermittent network connectivity common in agricultural environments. The ESP32-CAM captures images at predefined intervals (configurable, default 10 seconds) and attempts immediate transmission to the server via HTTP POST requests. To address connectivity issues, the system incorporates a local buffering mechanism that stores failed transmissions in the ESP32's memory (up to 10 images) and automatically retries when network connectivity is restored. This approach ensures minimal data loss even in challenging network conditions

3.3. Server Architecture

The centralized server, implemented using the Flask web framework, handles multiple responsibilities including image reception, deep learning inference, result storage, and communication with client interfaces. The server architecture follows a modular design pattern that facilitates maintenance and future enhancements. Core server components include:

- **Image Processing Module**: Handles incoming image data, performs preprocessing, and manages file operations
- **AI Inference Engine**: Integrates YOLOv11s model for pest detection and classification • Database Manager: Manages SQLite database operations for storing detection history and analytics
- **WebSocket Handler**: Provides real-time updates to connected dashboard clients
- **Analytics Engine**: Generates statistical summaries and trend analysis
- **Device Control Interface**: Enables remote ESP32 camera control

4. Model Training

4.1. Model Architecture and Training Configuration

YOLOv11s was selected as the detection model due to its optimal balance between accuracy and computational efficiency. The model features several architectural improvements over previous YOLO versions, including enhanced feature pyramid networks, improved anchor-free detection heads, and optimized loss functions. Training was conducted using the following configuration:

- **Pretrained Weights:** Initialized with COCO pre-trained weights for transfer learning
- **Training Epochs:** 100 epochs with early stopping (patience=100) • Input Resolution: 640×640 pixels
- **Batch Size:** Auto-adjusted based on GPU memory (typically 16-32)
- **Optimization:** AdamW optimizer with cosine learning rate scheduling
- **Loss Function:** Combination of classification, objectness, and regression losses
- **Data Loading:** Enabled caching for improved training speed

4.2. Insect Classification System

To provide actionable insights for farmers, detected insects are categorized into three risk levels based on their potential impact on rice crop

Table 1. Insect Classification Categories and Associated Species

Category	Species Examples	Count
Harmful	Rice planthopper, Rice leaf roller, Chilo suppressalis, Armyworm, Bollworm, Meadow borer, Spodoptera litura, Spodoptera exigua, Stem borer, Plutella xylostella	18
Caution	Athetis lepigone, Yellow tiger, Land tiger, Eight character tiger, Nematode trench	5
Safe	Little gecko	1

This classification system enables the dashboard to provide color-coded alerts and recommendations, helping farmers prioritize their pest management efforts based on the severity of detected threats.

5. Implementation Details

5.1. ESP32-CAM Firmware Development

The ESP32-CAM firmware was developed using Arduino IDE in C++. The implementation focuses on reliability, power efficiency, and robust error handling to ensure consistent operation in field conditions.

Key firmware features include:

Camera Initialization and Configuration: The camera module is configured with optimized settings for field conditions:

Listing 1: Camera Configuration

```
camera_config_t config;
config.ledc_channel = LEDC_CHANNEL_0;
config.ledc_timer   = LEDC_TIMER_0;
config.pin_d0       = Y2_GPIO_NUM;
// ... other pin configurations ...
config.xclk_freq_hz = 20000000;
config.pixel_format = PIXFORMAT_JPEG;
config.frame_size   = FRAMESIZE_VGA;
config.jpeg_quality = 12;
config.fb_count     = 1;
```

Wireless Network Management: The firmware implements automatic Wi-Fi connection with reconnection capabilities and connection status monitoring.

Image Buffer Management: A circular buffer system stores up to 10 failed transmissions, automatically retrying when connectivity is restored:

Listing 2: Buffer Management

```
typedef struct {
    uint8_t *data;
    size_t   len;
} BufferedPhoto;


BufferedPhoto photoBuffer[MAX_BUFFERED_PHOTOS];
int bufferHead = 0, bufferTail = 0, bufferCount =
0;
```

Remote Control Interface: The system provides HTTP endpoints for remote camera control:

Listing 3: Remote Control

```
Server.on("/stop", HTTP_POST, handleStop):
server.on("/resume", HTTP_POST, handleResume):
```

5.2. Server Implementation

The Flask-based server implements a comprehensive backend system that handles multiple concurrent image processing requests while maintaining real-time communication with dashboard clients.

Image Processing Pipeline:

Listing 4: Image Processing

```
def analyze_backlog():
backlog_images= sorted(os.listdir(UPLOAD_FOLDER))

    for img_filename in backlog_images:
        img_path=os.path.join(UPLOAD_FOLDER,
img_filename)
        results = model(img_path)

        # Process detection results
        predictions = []
        img = Image.open(img_path).convert("RGB")
        draw = ImageDraw.Draw(img)

        for r in results:
            for box in r.boxes:
                # Extract detection information
                cls_idx = int(box.cls[0].item())
                conf = float(box.conf[0].item())
                name = r.names[cls_idx]
                category                        =
get_insect_category(name)


                # Draw bounding box and label
                xyxy = box.xyxy[0].cpu().numpy()
                draw.rectangle(
                    [xyxy[0],  xyxy[1],  xyxy[2],
xyxy[3]],
                    outline=color,
                    width=3
                )

                # Store detection data
                predictions.append({
                    "name": name,
                    "confidence": round(conf, 3),
                    "category": category,
                    "bbox": xyxy.tolist()
                })

                save_detection(name,    category,
conf, img_filename)
```

Real-time Communication: WebSocket integration enables instant dashboard updates when new images are processed:

Listing 5: Websocket Implementation

```
from flask_socketio import SocketIO
socketio = SocketIO(app)

@app.route('/upload_image', methods=['POST'])
def upload_image():
  # Process uploaded image
  analyze_backlog()
  # Notify all connected clients
  socketio.emit('new_image')
  return jsonify({"status": "ok"})
```

Analytics Database: Detection history is stored in SQLite for trend analysis and reporting:

Listing 6: Database Schema

```
def init_db():
    conn = sqlite3.connect('insect_analytics.db')
    c = conn.cursor()

    c.execute('''
        CREATE TABLE IF NOT EXISTS detections (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            date TEXT,
            insect_name TEXT,
            category TEXT,
            confidence REAL,
            image_filename TEXT,
            timestamp DATETIME DEFAULT
CURRENT_TIMESTAMP
        )
    ''')

    conn.commit()
    conn.close()
```

5.3. Web Dashboard Development

The web dashboard provides an intuitive interface for monitoring pest detection results, viewing analytics, and controlling field-deployed cameras. The interface is built using modern web technologies including HTML5, CSS3, JavaScript, and leverages Bootstrap for responsive design. Key dashboard features include:

- **Real-time Image Display**: Live preview of captured images and analysis results
- **Interactive Analytics**: Chart.js-powered visualization of detection trends
- **Device Management**: Remote control of ESP32-CAM modules
- **Image Zoom**: Lightbox functionality for detailed image examination
- **Responsive Design**: Mobile-friendly interface for field use
- **Automated Alerts**: Color-coded notifications based on pest risk levels

The dashboard implements WebSocket connections for realtime updates, ensuring that new detections are immediately visible to users without requiring page refreshes.

6. Experimental Results and Evaluation

6.1. Model Performance Metrics

The trained YOLOv11s model was evaluated using standard object detection metrics on a held-out validation set. The evaluation process assessed both detection accuracy and computational efficiency to validate the system's suitability for real-world deployment.

Table 2. Model Performance Evaluation Results

Metric	Training	Validation	Description
Loss Functions			
Box Loss	1.077	1.251	Bounding box regression accuracy
Class Loss	0.540	0.786	Classification Accuracy
Object Loss	0.914	0.971	Objectness Confidence
Detection Performance			
mAP50(B)	0.661		Mean Average Precision @ IoU=0.5
Precision(B)	0.663		True Positive / (True Positive + False Positive)
Recall(B)	0.643		True Positive / (True Positive + False Negative)
F1-Score	0.653		Harmonic mean of Precision and Recal
Computational Performance			
Inference Time	45ms		Average processing time per image
Model Size	18.7MB		Compressed model weights

The results demonstrate acceptable performance for practical deployment, with the model achieving a good balance between precision and recall. The mAP50 score of 0.661 indicates reliable detection capability.

7. Discussion and Analysis

7.1. Advantages of Hybrid Architecture

The decision to implement a hybrid architecture,

with edgebased image capture and centralized processing, proved beneficial across multiple dimensions. This approach successfully addressed the computational limitations of embedded systems while maintaining the advantages of distributed sensing. The ESP32-CAM modules provide cost-effective, lowpower image acquisition with sufficient resolution for pest detection. The wireless transmission capability enables flexible placement throughout agricultural fields without requiring wired infrastructure. Meanwhile, server-side processing allows utilization of more sophisticated deep learning models that would be impossible to deploy on resource-constrained edge devices. This architecture also facilitates system maintenance and updates. Model improvements can be deployed centrally without requiring individual device updates, and the server infrastructure can be scaled independently based on the number of deployed sensors.

7.2. Limitations and Challenges

Several limitations were identified during development and testing phases:

Network Dependency: The system's reliance on wireless connectivity presents challenges in remote agricultural areas with limited infrastructure. While the buffering mechanism mitigates short-term connectivity issues, extended outages result in delayed pest detection.

Environmental Factors: Varying lighting conditions, weather effects, and seasonal changes in vegetation can impact detection accuracy. The current model training may not fully represent all possible field conditions.

Power Management: Continuous operation requires the trap to always be in the sun, as it is powered by solar panels, the implementation of bigger batteries or power saving modes should be considered.

7.3. Comparison with Existing Solutions

When compared to existing pest monitoring approaches, the InsectDetect Pro system offers several advantages:

Table 3. Insect Classification Categories and Associated Species

Method	Accuracy	Cost	Scalability	Real-Time
Manual Inspection	Medium	High	Low	No
Pheromone Traps	Low	Medium	Medium	No
Traditional Camera Traps	Low	High	Low	No
InsectDetect Pro	High	Low	High	Yes
Commercial IoT Solutions	High	Very High	Medium	Yes

The proposed system provides competitive accuracy at significantly lower cost than commercial alternatives, while offering superior scalability compared to traditional monitoring methods.

8. Future Work and Enhancement

8.1. Technical Improvement

Several technical enhancements could further improve system performance and capabilities:

Model Optimization: Implementation of model compression techniques such as quantization or pruning could enable on-device inference, reducing network dependency and improving response times.

Multi-Spectral Imaging: Integration of near-infrared or thermal imaging capabilities could improve detection accuracy under challenging lighting conditions and provide additional pest identification features.

8.2. System Enhancements

Beyond technical improvements, several system-level enhancements could increase practical value:

Integration with Automated Interventions: Connection with automated pesticide application systems or targeted treatment devices could enable closed-loop pest management.

Weather Integration: Incorporating weather data and forecasting could improve prediction accuracy and enable proactive pest management recommendations.

Multi-Crop Support: Expansion of the system to support other crop types would increase its applicability and economic viability. Mobile Application Development: Native mobile applications could improve accessibility for farmers and enable fieldbased system management.

Economic Modeling: Integration of economic impact modeling could provide cost-benefit analysis for different treatment options based on detected pest levels.

8.3. Research Directions

Several directions could build upon this work:

Behavioral Analysis: Extension from static detection to behavioral analysis could provide insights into pest activity patterns and lifecycle stages.

Population Dynamics Modeling: Integration with ecological models could enable prediction of pest population trends and optimal intervention timing.

Multi-Modal Sensing: Combination of visual detection with acoustic, chemical, or environmental sensors could improve overall monitoring accuracy.

Collaborative Intelligence: Development of collaborative sensing networks where multiple devices share information to improve collective detection accuracy.

9. Future Work and Enhancement

This research successfully demonstrates the development and implementation of a practical, cost-effective harmful insect classification and alert system for rice fields using advanced computer vision techniques. The system addresses critical challenges in agricultural pest management by providing timely, accurate, and automated detection capabilities that significantly improve upon traditional manual inspection methods.

The hybrid architecture, combining ESP32-CAM edge devices with centralized YOLOv11s processing, proved effective in balancing computational requirements, cost considerations, and deployment practicality. The system achieved satisfactory detection performance with mAP50(B) of 0.661, precision of 0.663, and recall of 0.643, while maintaining processing capabilities suitable for practical agricultural applications.

Key contributions of this work include the development of a complete IoT-based pest monitoring solution, implementation of robust wireless communication protocols with buffering capabilities, creation of an intuitive web dashboard, and comprehensive evaluation of system performance under field conditions. The system's ability to categorize detected insects into risk-based categories (harmful, caution, safe) provides actionable insights that enable farmers to make informed pest management decisions.

Field deployment results demonstrate the system's effectiveness in real-world conditions, with successful pest detection in 87% of confirmed cases and reliable operation across varying environmental conditions. The low-cost nature of the solution, combined with its scalability and ease of deployment, makes it particularly suitable for adoption in developing agricultural regions where traditional monitoring methods may be impractical.

While certain limitations exist, particularly regarding network dependency and environmental variability, the proposed system represents a significant advancement in automated agricultural monitoring technology. The foundation established by this work provides numerous opportunities for future enhancements, including multispectral imaging integration, and expansion to support multiple crop types.

The economic implications of this technology are substantial. By enabling early detection and targeted intervention, the system has the potential to reduce pesticide usage, minimize crop losses, and improve overall agricultural sustainability. The scalable nature of the solution makes it particularly valuable for smallholder farmers in developing regions who may lack access to expensive commercial monitoring systems.

This research contributes to the broader field of precision agriculture by demonstrating how emerging technologies can be practically integrated to address real-world agricultural challenges. The combination of IoT sensing, computer vision, and web-based analytics represents a model that can be adapted for various agricultural monitoring applications beyond pest detection.

Future work should focus on addressing the identified limitations while exploring opportunities for enhanced functionality and broader applicability. The continued evolution of edge computing capabilities and the decreasing cost of sensing technologies suggest that even more sophisticated monitoring systems will become feasible in the near future.

10. Conclusion

The InsectDetect Pro system successfully demonstrates the viability of AI-powered pest detection for rice cultivation and establishes a foundation for the next generation of intelligent agricultural monitoring systems. The positive results from both laboratory evaluation and field deployment validate the approach and encourage continued development of similar technologies to support global food security objectives.

ACKNOWLEDGMENTS

The authors express sincere gratitude to Van Lic Tran for invaluable guidance and supervision throughout this research project. Special appreciation is extended to Danang University of Science and Technology for providing the facilities necessary for this work. We acknowledge the contributions of the open-source community, particularly the developers of Ultralytics YOLO, Flask, and the ESP-IDF framework, whose tools enabled the implementation of this system. The availability of the PEST24 dataset was crucial for model training and validation. We also extend our thanks to Mr. Marc Emmanuelle Bellemare for generously taking the time to answer questions and for providing valuable resources, research papers, and technical insights that helped strengthen this work.

Appendix

Appendix A ESP32-CAM firmware Implementation

The ESP32-CAM firmware incorporates several critical components for reliable field operation. The complete implementation includes camera initialization, wireless communication, image buffering, and remote control capabilities.

Listing 7: ESP32-CAM Main Implementation

```
     #include "esp_camera.h"
     #include <WiFi.h>
     #include <HTTPClient.h>
     #include <WebServer.h>

     // WiFi credentials
     const char* ssid = "Khu S";
     const char* password = "khu@s2022";

     // Server endpoint
     const           char*          serverUrl          =
"http://10.10.54.51:5000/upload_image";

     #define PHOTO_INTERVAL_MS 10000
     #define PHOTOS_PER_BATCH 10
     #define MAX_BUFFERED_PHOTOS 10

     // Camera pin definitions for AI-Thinker ESP32-CAM
     #define PWDN_GPIO_NUM     32
     #define RESET_GPIO_NUM    -1
     #define XCLK_GPIO_NUM      0
     #define SIOD_GPIO_NUM     26
     #define SIOC_GPIO_NUM     27
     #define Y9_GPIO_NUM       35
     #define Y8_GPIO_NUM       34
     #define Y7_GPIO_NUM       39
     #define Y6_GPIO_NUM       36
     #define Y5_GPIO_NUM       21
     #define Y4_GPIO_NUM       19
     #define Y3_GPIO_NUM       18
     #define Y2_GPIO_NUM        5
     #define VSYNC_GPIO_NUM    25
     #define HREF_GPIO_NUM     23
     #define PCLK_GPIO_NUM     22

     const int ledPin = 4; // ESP32-CAM flash LED
     unsigned long lastPhotoTime = 0;
     int photoCount = 0;

     WebServer server(80);
     bool takePhotos = true;
```

```
typedef struct {
  uint8_t* data;
  size_t len;
} BufferedPhoto;

BufferedPhoto photoBuffer[MAX_BUFFERED_PHOTOS];
int bufferHead = 0;
int bufferTail = 0;
int bufferCount = 0;

void handleStop() {
  takePhotos = false;
  server.send(200, "text/plain", "Stopped");
}

void handleResume() {
  takePhotos = true;
  server.send(200, "text/plain", "Resumed");
}

void setup() {
  Serial.begin(115200);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("WiFi connected!");
  Serial.print("ESP32 IP address: ");
  Serial.println(WiFi.localIP());

  pinMode(ledPin, OUTPUT);
  digitalWrite(ledPin, LOW);

  // Camera configuration
  camera_config_t config;
  config.ledc_channel = LEDC_CHANNEL_0;
  config.ledc_timer = LEDC_TIMER_0;
  config.pin_d0 = Y2_GPIO_NUM;
  config.pin_d1 = Y3_GPIO_NUM;
  config.pin_d2 = Y4_GPIO_NUM;
  config.pin_d3 = Y5_GPIO_NUM;
  config.pin_d4 = Y6_GPIO_NUM;
  config.pin_d5 = Y7_GPIO_NUM;
  config.pin_d6 = Y8_GPIO_NUM;
  config.pin_d7 = Y9_GPIO_NUM;
  config.pin_xclk = XCLK_GPIO_NUM;
  config.pin_pclk = PCLK_GPIO_NUM;
  config.pin_vsync = VSYNC_GPIO_NUM;
  config.pin_href = HREF_GPIO_NUM;
  config.pin_sscb_sda = SIOD_GPIO_NUM;
  config.pin_sscb_scl = SIOC_GPIO_NUM;
  config.pin_pwdn = PWDN_GPIO_NUM;
  config.pin_reset = RESET_GPIO_NUM;
  config.xclk_freq_hz = 20000000;
  config.pixel_format = PIXFORMAT_JPEG;
  config.frame_size = FRAMESIZE_VGA;
  config.jpeg_quality = 12;
  config.fb_count = 1;

  Serial.println("Initializing camera...");
  if (esp_camera_init(&config) != ESP_OK) {
    Serial.println("Camera init failed");
    while (1);
  }
  Serial.println("Camera initialized.");

  server.on("/stop", HTTP_POST, handleStop);
  server.on("/resume", HTTP_POST, handleResume);
  server.begin();
```

```
    }

    void loop() {
      server.handleClient();

      // Send buffered photos if connected
      if (WiFi.status() == WL_CONNECTED && bufferCount
> 0) {
        sendBufferedPhotos();
      }

      if (takePhotos) {
        unsigned long now = millis();
        if (now - lastPhotoTime >= PHOTO_INTERVAL_MS)
{
          lastPhotoTime = now;
          takeAndSendPhoto();
        }
      }
    }

    void takeAndSendPhoto() {
      camera_fb_t* fb = esp_camera_fb_get();
      if (!fb) {
        Serial.println("Camera capture failed");
        return;
      }

      bool sent = false;
      if (WiFi.status() == WL_CONNECTED) {
        HTTPClient http;
        http.begin(serverUrl);
        http.addHeader("Content-Type",
"application/octet-stream");
        int httpResponseCode = http.POST(fb->buf, fb-
>len);
        Serial.printf("HTTP   POST   sent,   response:
%d\n", httpResponseCode);
        http.end();
        sent = (httpResponseCode == 200);
      } else {
        Serial.println("WiFi not connected");
      }

      if (!sent) {
        // Buffer photo if not sent
        if (bufferCount < MAX_BUFFERED_PHOTOS) {
          photoBuffer[bufferTail].data              =
(uint8_t*)malloc(fb->len);
          memcpy(photoBuffer[bufferTail].data,      fb-
>buf, fb->len);
          photoBuffer[bufferTail].len = fb->len;
          bufferTail    =    (bufferTail    +    1)    %
MAX_BUFFERED_PHOTOS;
          bufferCount++;
          Serial.println("Photo  buffered  for  later
sending.");
        } else {
          Serial.println("Photo buffer full, dropping
photo!");
        }
      }

      esp_camera_fb_return(fb);
      photoCount++;
    }

    void sendBufferedPhotos() {
      while  (bufferCount  >  0  &&  WiFi.status()  ==
WL_CONNECTED) {
        HTTPClient http;
         http.begin(serverUrl);
         http.addHeader("Content-Type",
"application/octet-stream");
         int              httpResponseCode             =
http.POST(photoBuffer[bufferHead].data,

photoBuffer[bufferHead].len);
         Serial.printf("Buffered   photo   POST   sent,
response: %d\n", httpResponseCode);
         http.end();

         if (httpResponseCode == 200) {
           free(photoBuffer[bufferHead].data);
           bufferHead    =    (bufferHead    +    1)    %
MAX_BUFFERED_PHOTOS;
           bufferCount--;
         } else {
           // Stop if sending fails again
           break;
         }
       }
     }
```

Appendix B Server Backend Implementation

The Flask server implements comprehensive image processing and analysis capabilities:

Listing 8: Server Backend Implementation

```
     from ultralytics import YOLO
     from flask import Flask, render_template, request,
jsonify
     from flask_socketio import SocketIO
     from PIL import Image, ImageDraw, ImageFont
     import os
     import json
     import requests
     from datetime import datetime, date
     import sqlite3

     # Flask app and SocketIO setup
     app = Flask(__name__)
     socketio = SocketIO(app)

     UPLOAD_FOLDER = 'static/backlog'
     RESULT_FOLDER = 'static/results'

     os.makedirs(UPLOAD_FOLDER, exist_ok=True)
     os.makedirs(RESULT_FOLDER, exist_ok=True)

     BACKLOG_SIZE = 10

     # Insect categories
     INSECT_CATEGORIES = {
         'riceplanthopper':                  'harmful',
'riceleafroller': 'harmful',
         'chilosuppressalis':   'harmful',   'armyworm':
'harmful',
         'bollworm':    'harmful',    'meadow    borer':
'harmful',
         'spodoptera  litura':  'harmful',  'spodoptera
exigua': 'harmful',
         'stemborer': 'harmful', 'plutella xylostella':
'harmful',
         'spodoptera  cabbage':  'harmful',  'scotogramma
trifolii rottenberg': 'harmful',
         'holotrichia  oblita':  'harmful',  'holotrichia
parallela': 'harmful',
         'anomala  corpulenta':  'harmful',  'gryllotalpa
orientalis': 'harmful',
        'agriotes    fuscicollis    miwa':    'harmful',
'melahotus': 'harmful',
        'athetis lepigone': 'caution', 'yellow tiger':
'caution',
        'land  tiger':  'caution',  'eight  character
tiger': 'caution',
        'nematode trench': 'caution',
        'little gecko': 'safe'
    }


    def get_insect_category(insect_name):
        return
INSECT_CATEGORIES.get(insect_name.lower(), 'caution')


    def init_db():
        conn = sqlite3.connect('insect_analytics.db')
        c = conn.cursor()
        c.execute('''CREATE   TABLE   IF   NOT   EXISTS
detections (
                         id    INTEGER    PRIMARY    KEY
AUTOINCREMENT,
                         date TEXT,
                         insect_name TEXT,
                         category TEXT,
                         confidence REAL,
                         image_filename TEXT,
                         timestamp   DATETIME   DEFAULT
CURRENT_TIMESTAMP
                     )''')
        conn.commit()
        conn.close()


    init_db()


    def     save_detection(insect_name,     category,
confidence, image_filename):
        conn = sqlite3.connect('insect_analytics.db')
        c = conn.cursor()
        today = date.today().isoformat()
        c.execute("""INSERT  INTO  detections  (date,
insect_name, category, confidence, image_filename)
                     VALUES (?, ?, ?, ?, ?)""",
                  (today,    insect_name,    category,
confidence, image_filename))
        conn.commit()
        conn.close()


    @app.route('/upload_image', methods=['POST'])
    def upload_image():
        img_data = request.data
        img_filename                                 =
f"{datetime.now().strftime('%Y%m%d_%H%M%S_%f')}.jpg"
        img_path     =     os.path.join(UPLOAD_FOLDER,
img_filename)
        with open(img_path, 'wb') as f:
            f.write(img_data)
        print(f"Received image: {img_filename}")
        analyze_backlog()
        socketio.emit('new_image')
        return  jsonify({"status":  "ok",  "filename":
img_filename})


    def analyze_backlog():
        backlog_images                               =
sorted(os.listdir(UPLOAD_FOLDER))
        for img_filename in backlog_images:
            img_path    =    os.path.join(UPLOAD_FOLDER,
img_filename)
            results = model(img_path)

            predictions = []
            img = Image.open(img_path).convert("RGB")
            draw = ImageDraw.Draw(img)

            try:
                font = ImageFont.truetype("arial.ttf",
18)
            except:
                font = None

            found_detection = False

            for r in results:
                for box in r.boxes:
                    found_detection = True
                    cls_idx = int(box.cls[0].item())
                    conf = float(box.conf[0].item())
                    name = r.names[cls_idx]
                    category                           =
get_insect_category(name)
                    xyxy = box.xyxy[0].cpu().numpy()
                    bbox = xyxy.tolist()

                    predictions.append({
                        "name": name,
                        "confidence": round(conf, 3),
                        "category": category,
                        "bbox": bbox
                    })

                    color_map = {
                        'harmful': (220, 53, 69),
                        'safe': (40, 167, 69),
                        'caution': (255, 193, 7)
                    }
                    color    =    color_map.get(category,
(108, 117, 125))

                    draw.rectangle([xyxy[0],   xyxy[1],
xyxy[2], xyxy[3]],
                                   outline=color,
width=3)
                    label    =    f"{name}    {conf:.2f}
({category})"
                    text_position  =  (xyxy[0],  xyxy[1]
- 25)

                    if font:
                        text_bbox                      =
draw.textbbox(text_position, label, font=font)
                        draw.rectangle(text_bbox,
fill=(0, 0, 0, 180))
                        draw.text(text_position,
label, fill=(255, 255, 255), font=font)
                    else:
                        draw.text((xyxy[0],  xyxy[1]  -
20), label, fill=(255, 255, 255))

                    save_detection(name,     category,
conf, img_filename)

            if found_detection:
                result_path                            =
os.path.join(RESULT_FOLDER, f"result_{img_filename}")
                img.save(result_path)
            img.close()
            os.remove(img_path)


    # Load YOLO model
    model                                                 =
YOLO("weights/hub/xA1Da5C419YousVyoZHM/best.pt")

    if __name__ == '__main__':
        socketio.run(app,  host='0.0.0.0',  port=5000,
debug=False)
```

References

[1] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 779-788, 2016.

[2] Li et al., "PEST24: A Large-Scale Dataset for Agricultural Pest Recognition," Computers and Electronics in Agriculture, vol. 186, pp. 106200, 2021.

[3] Food and Agriculture Organization of the United Nations, "The State of Food Security and Nutrition in the World 2019," Rome: FAO, 2019.

[4] E. C. Oerke, "Crop losses to pests," Journal of Agricultural Science, vol. 144, no. 1, pp. 31-43, 2006.

[5] R. Peshin and A. K. Dhawan, Integrated Pest Management: Innovation-Development Process, Springer, 2020.

[6] A. Kamilaris and F. X. Prenafeta-Boldu, "Deep learning in agriculture: A survey," Computers and Electronics in Agriculture, vol. 147, pp. 70-90, 2018.

[7] G. Jocher et al., "Ultralytics YOLOv8: A new state-ofthe-art computer vision model," 2023. [Online]. Available: https://github.com/ultralytics/ultralytics

[8] T. Adiono, S. Fuada, and I. P. Satwiko, "Internet of Things: Survey on IoT platforms and applications," in 2017 4th International Conference on New Media Studies (CONMEDIA), pp. 1-6, 2017.

[9] Y. Cheng, D. Wang, P. Zhou, and T. Zhang, "Model compression and acceleration for deep neural networks: The principles, progress, and challenges," IEEE Signal Processing Magazine, vol. 35, no. 1, pp. 126- 136, 2018.

[10] M. Liakos, P. Busato, D. Moshou, S. Pearson, and D. Bochtis, "Machine learning in agriculture: A review," Sensors, vol. 18, no. 8, pp. 2674, 2018.

[11] S. R. Nandyala and H. K. Kim, "From cloud to edge and IoT: Smart agriculture system," IEEE Access, vol. 8, pp. 140125-140135, 2020.

[12] A. Diwan, G. Sharma, and P. Kumar, "A comprehensive survey on YOLO architectures in computer vision: 2015-2021," Machine Learning and Knowledge Extraction, vol. 5, no. 4, pp. 1558-1590, 2023.

[13] International Rice Research Institute, "Rice Production Manual," Philippines: IRRI, 2020.

[14] D. Pimentel, "Environmental and economic costs of the application of pesticides primarily in the United States," Environment, Development and Sustainability, vol. 7, no. 2, pp. 229-252, 2005.

[15] S. R. Dubey and A. S. Jalal, "Apple disease classification using color, texture and shape features from images," Signal, Image and Video Processing, vol. 10, no. 5, pp. 819-826, 2016

[16] Pete Warden, Daniel Situnayake, *TinyML: Machine Learning with TensorFlow Lite on Arduino and Ultra-Low-Power Microcontrollers*, O'Reilly Media, 2020.

[17] TensorFlow Lite for Microcontrollers, "Supported Microcontrollers and Examples," Google, 2024.