Smart mosquito detection: leveraging AI and IoT for vector-borne disease prevention

Maria Niculae,
Beia Consult International, Bucharest, Romania
maria.niculae@beia.ro

Cosmina Stalidi,
Beia Consult International, Bucharest, Romania
cosmina.stalidi@beia.ro

George Suciu,
Beia Consult International, Bucharest, Romania
george@beia.ro

Andreea-Maria Serbanica,
Beia Consult International, Bucharest, Romania
andreea.mariaserbanica@gmail.com

Abstract
Vector-borne diseases such as malaria, dengue, Zika, and West Nile virus remain among the world's most severe public health threats, causing more than 700,000 deaths annually. The objective of this work is to demonstrate how artificial intelligence (AI), Internet of Things (IoT), and edge computing technologies can transform mosquito surveillance from reactive, manual monitoring into a proactive, data-driven ecosystem for early outbreak prevention. Traditional control methods, spraying, net distribution, and manual inspection, have proven life-saving but inefficient, particularly as climate change and globalization expand mosquito habitats. Recent advances in AI and IoT have enabled automated detection, species identification, and predictive modeling, but large-scale integration remains limited. This study analyzes smart mosquito detection systems that combine IoT-enabled traps, AI-based image and sound recognition, and real-time data analytics platforms. These systems identify species via wingbeat frequency and computer vision, transmit encrypted data through MQTT or LoRaWAN, and integrate environmental and meteorological data for predictive modeling. Field demonstrations show significant improvements in detection accuracy, latency, and outbreak prediction capabilities, enabling timely public health interventions such as targeted spraying and community alerts. For health authorities and policymakers, the implementation of smart detection networks enhances preparedness, reduces chemical use, and lowers operational costs. This paper highlights how interconnected AI- and IoT-based mosquito surveillance represents a paradigm shift from reactive disease control to predictive prevention, offering a scalable, sustainable model for global health security and smart city resilience.

Keywords: vector surveillance, edge analytics, environmental monitoring, public health innovation, predictive epidemiology.

1. Introduction
Vector-borne diseases transmitted by mosquitoes, such as malaria, dengue, Zika, and West Nile virus, continue to impose a heavy burden on global public health. Even as advances in medicine and prevention reduce some impacts, these diseases still cause substantial morbidity and mortality, especially in tropical and subtropical regions. Traditional surveillance and control strategies often rely on manual mosquito trapping, periodic inspections, and reactive spraying campaigns. While such interventions have been life-saving, they tend to be labor-intensive, costly, and limited in scale. Moreover, as climate

change, urbanization, and globalization expand the geographic range of mosquito habitats, these conventional methods struggle to keep pace with rapidly shifting risk patterns.

In response to these challenges, emerging digital technologies, notably artificial intelligence (AI), the Internet of Things (IoT), and edge computing, offer a promising paradigm shift. By enabling continuous, automated monitoring and real-time data analytics, smart mosquito surveillance systems can transform vector control from reactive responses into proactive, data-driven prevention. Such systems have the potential not only to improve the timeliness and accuracy of mosquito detection, but also to integrate environmental data, forecast risk zones, and support targeted interventions, thus reducing chemical usage and operational costs.

In this paper, we present the design and implementation of an integrated smart mosquito detection ecosystem. Our system combines IoT-enabled traps, AI-based species identification, edge computing for on-device inference, and real-time data pipelines, demonstrating how modern technologies can enhance entomological surveillance and strengthen public-health preparedness.

2. State of the art

In recent years, the convergence of artificial intelligence (AI), Internet of Things (IoT), and edge computing has opened new possibilities for automating mosquito surveillance, potentially overcoming many limitations of traditional, labor-intensive vector monitoring. A growing body of work demonstrates that deep-learning systems based on convolutional neural networks (CNNs) and transformer architectures can accurately identify mosquito species from images, often with performance comparable to or exceeding expert entomological inspection. For example, a systematic review published in 2025 analyzed 52 studies (2000–2024) using machine-learning for image-based mosquito identification: pretrained CNN models consistently outperformed classical approaches, and many studies reported high classification accuracy for multiple species [1].

Recent experimental work has pushed these results further. A 2024 study showed that the network behind the “Mosquito Wing Classifier” could reliably classify mosquitoes based solely on wing images, exploiting species-specific wing-vein patterns that remain invariant over age or feeding status. The authors reported classification accuracy around 98.2%, even when images were acquired with different devices under varying conditions [2, 3]. This wing-based approach addresses a known limitation of full-body imaging (sensitivity to posture, occlusion, or partial damage), while facilitating use on diverse devices such as smartphone microscopes.

Other works explore different neural architectures or training strategies. For instance, a 2025 study introduced an ensemble transfer-learning method (called “TransembleNet”) combining multiple pretrained CNNs (InceptionV3, VGG-16, ResNet-50). On benchmark datasets of vector mosquitoes, this method significantly outperformed prior work in precision, recall, F1-score, and overall classification accuracy [4]. Similarly, more recent research has begun using vision transformers and open-set learning to enable robust classification even when the system is exposed to previously unseen classes (non-mosquito

insects, background noise, or novel species). A 2024 preprint reported closed-set classification accuracy of 99.8% (F1 = 0.998) for ten mosquito species, while retaining resilience under open-set conditions [5].

Beyond vision-based classification, researchers have also investigated multimodal detection strategies. A 2024 work demonstrated real-time mosquito genus classification using wingbeat frequency (acoustic signal) processed by a CNN. In binary classification (e.g., Aedes vs. Culex) the system reached 91.76% accuracy; in a multiclass configuration, it distinguished female and male specimens of different genera with ~87 % precision [6]. Meanwhile, progress in robust bioacoustics promises to improve cross-environment generalization: in 2025, a preprint proposed a contrastive-learning + distribution-alignment framework aimed at mitigating non-biological recording differences (e.g., background noise, recording device variation), achieving significantly better cross-domain species classification on multi-site recordings [7].

Complementing insect-level detection, broader system-level approaches have begun to emerge. For example, the publicly released "MosquitoFusion" dataset (2024) includes images not only of individual mosquitoes, but also of swarms and potential breeding-site features. The accompanying deep-learning pipeline (YOLOv8-based) achieved preliminary results (mAP@50 ≈ 57.1%), illustrating the feasibility of extending detection beyond single insects to population-level and habitat-level surveillance [8]. This direction is particularly valuable for early warning and environmental risk assessment, as it enables monitoring of both adult mosquitoes and breeding grounds.

More generally, reviews of insect-identification via machine learning have underscored the growing promise of these technologies while noting persistent challenges. Image acquisition and preprocessing remain critical: variation in imaging devices, lighting, backgrounds, and specimen preservation can significantly degrade performance or reduce generalizability [1, 9]. In addition, although many CNN-based systems perform well under controlled lab conditions, only a limited number have been tested in real-world field deployments, especially under varying environmental conditions typical of urban or rural vector-hazard zones. For example, a recent evaluation of a commercial system reported that even after algorithm refinements, regional macro-average recall for species identification rose from ~55% to ~80% in one geographic area, indicating variability in performance across locales and the need for more representative training data [10].

Finally, beyond individual-specimen classification, genetic and phylogenetic studies remain important. A 2024 global review of mosquito species identification and phylogenetics synthesized conventional morphological, molecular (e.g., mitochondrial genome, ribosomal RNA), and morphometric (e.g., wing-shape) methods. The authors argued for integrating these traditional approaches with automated techniques to maximize accuracy, especially for cryptic or closely related species, and to better understand vector-pathogen associations at the evolutionary level [11].

In synthesis, the current state of the art reveals that automated mosquito surveillance is no longer speculative: image-based deep learning, acoustic wingbeat analysis, and even

breeding-site or swarm detection have all been demonstrated in proof-of-concept or early deployment studies. In parallel, system-level contributions from the Smart Cities International Conference (SCIC) community show that IoT and edge–cloud infrastructures can successfully support smart-city services in other domains, such as cultural-heritage conservation through sensor-rich IoT monitoring platforms [12], urban water-incident detection using low-cost sensor networks and time-series dashboards [13], and real-time safety management via digital-twin platforms for critical urban infrastructure [14]. Together, these works indicate that modular IoT architectures and real-time analytics are technically mature for smart-city applications, yet significant gaps remain in their application to entomological monitoring: large-scale and long-term mosquito deployments, cross-species and cross-environment generalization, and fully integrated multimodal pipelines combining vision, acoustic, environmental, and spatial data are still rare. These gaps motivate our proposed system: a holistic, scalable, field-ready smart mosquito detection ecosystem designed for real-world public-health surveillance.

3. System Architecture Overview

The proposed smart mosquito detection system is designed as an end-to-end, distributed architecture that integrates sensing devices, edge-level AI processing, secure communication protocols, and cloud-based analytics. Its objective is to transform raw visual or acoustic data collected in the field into actionable information for public-health monitoring, outbreak prevention, and environmental assessment. The architecture follows a modular and scalable design, supporting heterogeneous sensors, multiple communication layers, and interoperable analytic components.

At a high level, the system consists of four major layers:

- the sensing and acquisition layer,
- the edge intelligence layer,
- the communication and data-transport layer, and
- the cloud and visualization layer.

The sensing layer is responsible for capturing mosquito-related signals, primarily through visual imaging and optionally acoustic or environmental sensor measurements. These sensing units are integrated within IoT-enabled mosquito traps or standalone imaging nodes, which supply continuous or periodic data streams.

The edge intelligence layer processes these data in real time using embedded hardware such as Raspberry Pi devices or similar single-board computers. By deploying optimized YOLOv8 models directly at the edge, the system minimizes latency, reduces bandwidth consumption, and eliminates the need to transmit raw video to the cloud. The edge modules perform frame acquisition, detection inference, bounding-box extraction, insect counting, and event generation. This layer also manages local storage, including timestamps and annotated images, enabling both immediate feedback and offline fallback operation in case of network interruptions.

The communication layer ensures reliable and secure transport of inference results from distributed devices to central services. Lightweight machine-to-machine protocols such as

MQTT are employed to publish mosquito counts, device metadata, environmental context, and system diagnostics to designated topics on a broker. This decoupled design supports thousands of edge devices, enables asynchronous communication, and allows other applications, such as public-health dashboards, anomaly detection modules, or alerting systems, to subscribe to real-time streams.

Once transmitted, data enter the cloud analytics layer, where results are processed, aggregated, and visualized. Time-series databases such as InfluxDB or Prometheus store mosquito count data, environmental measurements, and system logs. The visualization stack, implemented via Grafana dashboards, provides intuitive access to real-time and historical information. Users can explore trends across locations, evaluate species-specific activity (when species-level classification is enabled), and identify abnormal surges in mosquito abundance that may signal increased vector-borne disease risk. The cloud layer also supports threshold-based alerting, geographic mapping, and integration with other environmental datasets.

A key feature of the architecture is its modularity. The system accepts heterogeneous video sources, including local webcams, IP cameras, or embedded trap-mounted cameras, as evidenced in the implementation files (e.g., Foscam RTSP integration and webcam capture modules). It also accommodates additional sensing modalities, such as temperature, humidity, or CO_2 measurements, enabling correlation analyses between environmental variables and vector activity. Furthermore, the architecture supports incremental upgrades: edge devices can deploy improved models without disrupting data flows, cloud dashboards can ingest new sensor streams, and public-health authorities can configure custom alert thresholds or predictive models without modifying on-site hardware.

Overall, this multi-layer architecture supports a robust and scalable mosquito surveillance ecosystem. By combining distributed sensing, edge AI, lightweight communication, and centralized analytics, the system facilitates proactive monitoring and regional decision-making, advancing from traditional reactive control strategies to data-driven vector management and outbreak prevention.

4. Hardware and Devices

The hardware foundation of the proposed smart mosquito detection system consists of a modular combination of imaging components, embedded processing units, and environmental sensors. The chosen configuration emphasizes low-power operation, low maintenance requirements, and robustness for long-term deployment in both urban and rural environments. The architecture integrates IoT-enabled mosquito traps, edge-computing microcontrollers, and optional environmental-monitoring devices to create a unified, multi-sensor surveillance ecosystem. The flexibility of the system allows it to accommodate off-the-shelf components, such as Raspberry Pi single-board computers, wireless IP cameras, and microenvironmental sensors, while being extensible for integration into larger smart-city infrastructures.

A guiding principle in the hardware design is the use of distributed sensing nodes capable of autonomous operation. Each node performs real-time mosquito detection using

embedded AI models and transmits results via lightweight communication protocols. This enables large-scale deployment across multiple monitoring sites, reducing the reliance on centralized compute resources and minimizing network loads. In addition, the hardware platform supports the integration of weather or environmental parameters, allowing comprehensive correlation analysis between environmental conditions and vector activity.

4.1. IoT-Enabled Mosquito Trap Architecture

The mosquito trap architecture integrates optical sensing, optional acoustic capture, and IoT connectivity into a compact assembly suitable for field deployment. The core imaging component consists of a high-definition camera, either a USB webcam or a wireless IP camera such as the Foscam D4Z, mounted within or adjacent to the trap chamber. This configuration enables continuous monitoring of insect activity inside the trap without human intervention. The camera captures frames at preconfigured intervals or streams live video through RTSP, which the edge-processing unit ingests for inference.

To ensure stable operation under varying lighting conditions, the system can incorporate infrared illumination or low-intensity LED lighting within the trap chamber. These components improve nighttime detection accuracy while maintaining energy efficiency. The trap body may include attractants (e.g., CO_2, heat, or odor-lure cartridges), depending on deployment requirements, to draw mosquitoes into the field of view of the camera.

IoT integration is facilitated by embedded microcontrollers or the Raspberry Pi itself acting as the gateway. The trap transmits processed detection data, environmental metadata, and diagnostic information via MQTT or HTTPS depending on the deployment scenario. This capability transforms the trap from a passive collection device into an autonomous sensing node capable of reporting real-time mosquito presence, thereby enabling proactive vector-control operations.

4.2. Edge Computing Units

Edge computation is performed primarily by a Raspberry Pi 4 Model B or an equivalent single-board computer with sufficient processing power to execute lightweight deep-learning models. This platform supports Python-based inference pipelines using the Ultralytics YOLOv8 framework, OpenCV for frame capture and preprocessing, and locally stored custom-trained models. The Raspberry Pi serves as both the inference engine and the communication relay, performing detection, logging, and network transmission.

The Raspberry Pi's quad-core CPU and GPU acceleration enable real-time processing of video streams sourced either from directly attached USB cameras or IP cameras accessed over RTSP. The implementation provided in the system scripts allows the Raspberry Pi to:

- acquire frames from a local webcam or from the Foscam D4Z IP camera,
- run YOLOv8 inference on each frame,
- generate annotated visual outputs when needed,
- count detected mosquitoes or insects,
- log detections with timestamps in CSV files,
- save annotated images when insects are detected, and
- publish detection events periodically to MQTT topics.

The device's ability to process inferencing locally reduces the need for high-bandwidth video transmission and ensures operation even in intermittent connectivity conditions. In addition, the microSD-based file system allows the accumulation of long-term logs and offline storage buffers, which can later be uploaded or analyzed as needed.

Because the design is modular, other edge devices, such as NVIDIA Jetson Nano/Xavier or industrial IoT gateways, can be substituted in scenarios requiring higher throughput or multi-camera support. However, the Raspberry Pi platform provides an optimal balance between cost, efficiency, and computational capabilities for typical deployment scenarios.

4.3. Environmental Sensing Integration

The system architecture incorporates optional environmental sensing capabilities to complement mosquito detection and support predictive modeling. Environmental parameters such as temperature, humidity, barometric pressure, rainfall, wind speed, and solar radiation influence mosquito breeding patterns, survival, and flight activity. Integrating these data sources enhances the interpretation of detection trends and enables more accurate prediction of vector population dynamics.

Environmental data can be integrated from local sensors mounted on the same edge device or ingested from external weather-monitoring stations. In previous related deployments, comprehensive environmental monitoring was achieved using platforms such as Adcon Telemetry and Libelium Waspmote units, which provide real-time measurements of atmospheric conditions and feed data into Grafana dashboards. These sources can be coupled with mosquito detection outputs to generate multi-variable time-series analyses, correlating mosquito abundance with fluctuations in rainfall, humidity, or temperature.

Furthermore, when available, IoT-enabled environmental sensors can be synchronized with the detection nodes to provide hyperlocal environmental context. This allows the system to detect relationships such as sudden increases in mosquito activity following rainfall events or humidity spikes. The integration of environmental measurements also facilitates threshold-based alerts conditioned not only on insect count but also on ecological risk predictors.

Overall, incorporating environmental sensing broadens the analytical scope of the detection system, extending it from a purely entomological monitoring tool to a complex environmental intelligence platform suited for public-health decision support and smart-city applications.

5. AI Models and Detection Techniques

The core of the smart mosquito detection system is an AI-based inference pipeline designed to identify and count mosquitoes in real time using lightweight deep-learning models optimized for edge deployment. The system employs object detection techniques rather than simple classification, enabling the localization of individual mosquitoes within each frame and supporting accurate quantification under varying environmental conditions. Deep learning frameworks based on the YOLO (You Only Look Once) family of models

are particularly well suited for this purpose due to their high inference speed, favorable balance between accuracy and computational efficiency, and adaptability to low-power embedded devices.

The use of single-shot detectors simplifies the architecture by allowing prediction of bounding boxes and class probabilities in a single forward pass, substantially reducing inference latency compared to two-stage detectors such as Faster R-CNN. Furthermore, YOLOv8, the version used in this system, provides architectural improvements including decoupled heads, enhanced feature fusion, and optimized training routines. These characteristics enable robust detection of small, fast-moving targets such as mosquitoes, which present unique challenges due to their size, irregular motion, and visual similarity to other small insects.

To ensure high detection performance across varied imaging conditions, the model is trained on a curated dataset encompassing multiple mosquito species, varied orientations, different lighting situations, and diverse backgrounds. The inclusion of augmented data improves the model's robustness to real-world conditions such as motion blur, partial occlusion, and environmental noise. Once trained, the model is deployed in a quantized or otherwise optimized form to the edge computing units for on-device inference. The detection results include bounding boxes, confidence scores, and classification labels, which are subsequently used to construct time-stamped insect counts and trigger IoT-driven notification mechanisms.

5.1. Dataset Preparation and Training

Dataset construction is a critical step in achieving reliable and generalizable mosquito detection performance. The dataset used for this system was assembled using a combination of pre-annotated insect imagery from external repositories and custom-labeled data tailored to the deployment environment. A substantial portion of the training dataset was sourced from Roboflow, a platform that provides high-quality, standardized datasets for computer-vision applications. The Roboflow dataset included labeled images of mosquitoes and other insects, offering a diverse set of visual conditions such as varying camera angles, distances, specimen sizes, backgrounds, and illumination levels.

Prior to training, the dataset was exported in YOLOv8-compatible format, which includes image files, annotation text files with normalized bounding-box coordinates, and a configuration YAML file specifying class names and dataset structure. This standardized format allows seamless integration with the Ultralytics training pipeline. The exported dataset also incorporates several forms of automated augmentation performed by Roboflow, including random flips, rotations, noise injection, scaling, and color adjustments. These augmentations serve to expand dataset diversity, improve model robustness, and reduce overfitting, a critical challenge when dealing with small objects such as mosquitoes.

The dataset was divided into training, validation, and testing subsets following best practices for supervised learning. The training split is used to optimize model parameters, while the validation set guides hyperparameter adjustments and prevents overfitting. The

testing set is preserved for unbiased performance evaluation after training. During training, the YOLOv8 procedure adapts the model backbone and detection head using stochastic gradient descent or Adam-based optimizers, with loss functions incorporating classification loss, bounding-box regression loss, and objectness loss.

The training process is accelerated through the use of GPU hardware, either locally or via cloud platforms. Once the model achieves satisfying performance metrics, such as mean Average Precision (mAP) at IoU thresholds appropriate for small-object detection, the best-performing checkpoint (best.pt) is exported. This file is subsequently deployed to the Raspberry Pi or other edge devices for real-time inference. The final model is compact enough to run efficiently on low-power hardware without compromising detection accuracy.

By combining curated datasets, automated augmentation, and a modern object-detection framework, the training pipeline yields a model capable of high detection accuracy, reliable performance across diverse environments, and low-latency inference on resource-constrained devices.

5.2. YOLOv8 Inference Pipeline

The YOLOv8 inference pipeline constitutes the operational core of the mosquito detection system, enabling real-time processing of image streams and autonomous extraction of entomological indicators. The pipeline is optimized for lightweight embedded hardware, primarily Raspberry Pi devices, yet remains flexible enough for deployment on more powerful platforms when available. Its architecture is designed to maintain low latency, robust detection performance, and efficient resource usage even under constrained computational environments.

The inference process begins with frame acquisition from the designated imaging source. Depending on the deployment scenario, this input may originate from a USB-attached webcam or from a wireless IP camera accessed through an RTSP stream, such as the Foscam D4Z. The system's acquisition module uses OpenCV to initialize the video stream, buffer incoming frames, and perform any minimal preprocessing required to meet the input specifications of the YOLOv8 model. On the Raspberry Pi, this includes resizing frames or adjusting color format to match model expectations, ensuring that inference remains consistent across heterogeneous camera types.

Once a frame is acquired, it is passed to the YOLOv8 model, stored as a serialized weight file (e.g., best.pt), for forward inference. The model generates a set of predictions consisting of bounding boxes, class probabilities, and objectness confidence scores. Since mosquitoes are extremely small relative to the field of view, maintaining model sensitivity to small-object detection is essential. YOLOv8 incorporates architectural improvements such as decoupled detection heads and refined anchor-free algorithms that enhance both accuracy and speed for small targets. The pipeline implements a post-processing stage that filters predictions based on confidence thresholds to reduce false positives and improve reliability under challenging lighting or background conditions.

After the detection stage, the system derives relevant entomological indicators, including mosquito count per frame or per interval, species identification (if the model is trained on multiple mosquito classes), and event timestamps. Annotated frames can be generated using YOLOv8's built-in rendering utilities, producing overlays that highlight detected insects with bounding boxes and labels. These annotated frames are useful for verification, debugging, or downstream storage when insect activity exceeds predefined thresholds.

In addition to on-screen visualization, the inference pipeline handles automated data logging and communication. Each detection event triggers the generation of structured metadata entries, which may include insect count, detection timestamp, device identifier, and optional environmental parameters. These entries are appended to local CSV logs for long-term storage or failover operations. When detections exceed a configurable threshold or when periodic reporting is required, the pipeline publishes event data to cloud services via MQTT. This is facilitated by a lightweight message structure encoded in JSON, ensuring compatibility with time-series databases and real-time dashboards.

The modular nature of the pipeline allows for scalable deployment across multiple sensing nodes. Each node performs inference independently, transmitting only essential metadata rather than raw video, which drastically reduces bandwidth consumption and enhances the practicality of deploying the system in large numbers or in regions with limited network infrastructure. The entire workflow is optimized to operate continuously and autonomously, supporting long-term field monitoring with minimal human intervention.

6. Software Implementation

The software implementation of the smart mosquito detection system is structured as a modular, extensible pipeline that orchestrates real-time inference, local data management, and communication with cloud services. The design emphasizes operational robustness, fault tolerance, and low computational overhead to ensure sustained performance on resource-limited edge devices such as Raspberry Pi platforms. The implementation integrates computer-vision models, lightweight communication protocols, and time-series data handling to enable continuous, autonomous operation in diverse field environments.

The software stack is primarily implemented in Python, leveraging widely adopted libraries for computer vision (OpenCV), deep learning (Ultralytics YOLOv8), and IoT messaging (paho-mqtt). This choice facilitates portability across Linux-based embedded systems and enables rapid extension or optimization of the detection workflow. To ensure reliability and maintainability, the software components are organized into functional modules responsible for frame acquisition, inference, event generation, logging, communication, and optional visualization. Each module operates independently yet integrates seamlessly through a well-defined interface, allowing for system upgrades, such as the integration of additional sensors or alternative inference backends, without altering the overall architecture.

The operational logic centers on efficient management of computational resources and controlled scheduling of inference and reporting tasks. Continuous video streaming is handled asynchronously to avoid blocking operations on the inference loop. Error handling,

stream reconnection, and watchdog mechanisms ensure graceful recovery from intermittent network failures or camera disruptions. Local data persistence, via time-stamped CSV logs and optional annotated image storage, provides resilience against connectivity interruptions and supports post hoc analysis of field deployments. The combination of local intelligence and cloud connectivity allows the system to operate effectively in both offline and online modes.

6.1. Inference Engine

The inference engine represents the computational nucleus of the system, responsible for executing the YOLOv8 object detection model on incoming frames and producing structured outputs suitable for downstream analytics. Its design reflects three core objectives: (i) minimizing latency to enable near–real-time detection, (ii) ensuring accuracy for small-object detection, and (iii) providing stable operation under continuously running field conditions.

The engine begins by acquiring frames from either a directly connected webcam or an RTSP-enabled IP camera. The implementation uses OpenCV's VideoCapture interface to initialize and maintain video streams, automatically handling device discovery, frame retrieval, and reconnection logic. For IP cameras, such as the Foscam D4Z, the engine retrieves frames from the RTSP stream using the camera's network URL. The streaming logic includes safeguards that detect frame acquisition failures and introduce back-off intervals to reduce resource consumption during transient disruptions.

Once a frame is captured, it is passed directly to the YOLOv8 model loaded from the serialized best.pt weight file. The Ultralytics API is used to execute inference in a single pass, producing bounding-box predictions, class labels, and confidence scores. The model's anchor-free detection mechanism enhances performance on small insects such as mosquitoes, while the network's multi-scale feature extraction improves resilience to scale variations and motion blur. Inference speed is optimized through careful selection of model size (e.g., YOLOv8n or YOLOv8s), minimizing computational load while retaining sufficient precision for entomological monitoring.

Following inference, the engine extracts key outputs for further processing. The bounding boxes are counted to derive mosquito activity per frame, and confidence values allow for filtering of low-probability detections. When visualization is enabled, the engine uses YOLOv8's integrated rendering utilities to generate annotated frames highlighting detected insects. These annotated frames may be displayed locally or saved to persistent storage when detections exceed defined thresholds.

To support long-term monitoring, the engine records detection data in structured CSV logs. Each entry includes a timestamp, inferred insect count, and optionally device metadata. This persistent logging mechanism ensures traceability and provides a fallback dataset should cloud connectivity become temporarily unavailable.

At the communication level, the inference engine integrates with the system's MQTT client. At predefined intervals (e.g., every second or ten seconds depending on

deployment), the engine publishes a JSON-encoded message containing the current insect count or, when applicable, species-specific detection metrics. This lightweight publish–subscribe mechanism allows centralized systems to ingest data streams from multiple edge devices with minimal overhead.

The inference engine operates continuously in a loop, combining frame acquisition, inference, data logging, event publishing, and optional visualization. Careful scheduling ensures that tasks such as image saving or MQTT publishing do not interfere with real-time detection performance. Exception handling prevents system crashes, while controlled shutdown logic allows the engine to terminate gracefully when required.

Overall, the inference engine provides an efficient, reliable foundation for the smart mosquito detection system. Its low-latency operation, modular structure, and seamless integration with both local and cloud components enable persistent, real-time surveillance essential for proactive vector-monitoring and public-health decision support.

6.2. Communication Layer

The communication layer is responsible for transporting detection results, environmental metadata, and system diagnostics from distributed edge devices to cloud-based data stores and visualization dashboards. For a large-scale mosquito surveillance system, this layer must be lightweight, fault tolerant, scalable, and interoperable with heterogeneous devices deployed across varied environments. To meet these requirements, the implementation relies on message-oriented middleware, primarily the MQTT (Message Queuing Telemetry Transport) protocol, which is widely used in IoT architectures due to its efficiency and minimal bandwidth consumption.

6.2.1. MQTT Messaging Framework

MQTT operates on a publish–subscribe model, enabling distributed sensor nodes (publishers) to send messages to a central broker, where subscribers such as dashboards, databases, or alerting systems can receive relevant information. This model is well suited for mosquito detection because:

- Data volumes are small (counts, timestamps, metadata), eliminating the need to stream high-bandwidth images.
- Transmission is asynchronous, allowing edge devices to operate independently without requiring permanent connectivity.
- Publishers can be numerous, supporting city-wide deployments of traps and edge nodes.
- It tolerates intermittent networks, essential for outdoor or remote deployments.

Message Structure: Detection events are transmitted as compact JSON objects. A typical published MQTT message is:

```
{
  "device_id": "raspi_node_01",
  "timestamp": "2025-03-21 11:42:05",
  "insect_count": 3,
  "battery_level": 89,
  "temperature": 24.1
}
```


This flexible structure allows new fields, such as species labels, confidence scores, or trap operational state, to be added without breaking existing integrations.

6.2.2. Topic Hierarchy and Data Organization

To ensure scalability and maintain readability over hundreds of distributed nodes, the system organizes MQTT topics using a hierarchical naming convention. A typical structure is:

```
mosquito/
   device_id/
      detections
      images
      status
      environment
```

- detections – numerical insect counts, detection confidence, timestamps
- images – links or hashes of stored annotated images (optional)
- status – device health, CPU/GPU usage, uptime, connectivity
- environment – temperature, humidity, CO_2, rainfall, etc.

This structured approach ensures that analytics pipelines (Grafana, InfluxDB, Prometheus) can subscribe selectively to relevant data streams.

6.2.3. Broker Selection and Security

The system can operate using either:

- a public MQTT broker for testing or non-sensitive deployments,
- or a private/secure broker (e.g., self-hosted on the organization's infrastructure).

Security is implemented via:

- TLS-encrypted communication,
- username/password authentication,
- topic-based access control.

Such measures ensure that devices cannot publish to unauthorized topics and cloud applications cannot mistakenly alter device configurations.

6.2.4. Robustness and Fault Tolerance

The communication layer incorporates several mechanisms to ensure uninterrupted operation during real deployments:

Automatic Reconnection: If Wi-Fi or LTE connectivity drops, the MQTT client enters a back-off loop, attempting reconnection at controlled intervals to preserve battery and CPU resources.

Local Caching: If a message fails to publish, the system temporarily stores the detection event locally (CSV file) and resends it once connectivity is restored. This guarantees no-loss logging.

Device Heartbeats: Periodic MQTT "status" messages allow the server to:

- detect offline devices,
- monitor resource usage (RAM, CPU temp),
- track long-term performance.

A summarized view of the communication workflow is provided in Table 1.

Table 1. Overview of Communication Layer Functions

Component	Function	Example Implementation
MQTT Publisher	Sends detection events to broker	paho-mqtt Python client
MQTT Broker	Routes data to cloud consumers	Eclipse Mosquitto
Heartbeat subsystem	Reports device health	/status topic messages
Local buffer	Temporary storage during connectivity loss	CSV logs on Raspberry Pi
Subscriber applications	Consume and store time-series data	InfluxDB, Prometheus

6.2.5. Integration with Cloud Services

After publication to the MQTT broker, cloud services subscribe to the topics and relay them into storage and visualization tools. For example:

- InfluxDB stores numerical time-series such as detection frequency.
- Grafana visualizes mosquito activity, environmental data, and device status.
- Prometheus scrapes metrics for statistical modeling or alert generation.

This decoupled architecture enables the communication layer to remain simple and reliable while allowing advanced analytics to occur downstream.

6.3. Data Storage and Cloud Pipeline

The data storage and cloud pipeline form the analytical backbone of the smart mosquito surveillance system, enabling long-term trend analysis, cross-site comparisons, historical pattern discovery, and real-time visualization. Because mosquito activity is strongly influenced by environmental conditions and temporal cycles, the cloud pipeline is designed to handle high-frequency, multi-source time-series data while maintaining low latency for immediate operational decision-making.

The pipeline integrates MQTT ingestion, time-series databases, object/image repositories, and cloud visualization dashboards, forming a cohesive ecosystem that ensures durability, scalability, and interoperability.

6.3.1. Cloud Data Flow Overview

The cloud data flow follows a structured sequence from edge inference to centralized analytics:

Edge Device
→ MQTT Broker
→ Data Consumers (Telegraf / Custom Subscriber)
→ Time-Series Database (InfluxDB/Prometheus)
→ Dashboard & Analytics Layer (Grafana)

This modular topology decouples data producers from consumers, allowing the system to scale without requiring modifications to the edge devices. It also enables the integration of additional data sources, such as weather stations or acoustic sensors, into the same analytics ecosystem.

6.3.2. Time-Series Storage

InfluxDB or Prometheus Integration: Detection events published by the edge devices via MQTT are ingested by a collector service (e.g., Telegraf for InfluxDB or a custom exporter for Prometheus), which formats and writes data into a time-series database.

Reasons for using a time-series database:

- High write throughput suitable for many sensing nodes reporting every second.
- Efficient retention policies, enabling automatic pruning of old data.
- Native support for tags such as device ID, location, species class, confidence level.
- Fast querying for historical temporally indexed data.

A typical record stored in InfluxDB might appear as:

```
mosquito_activity,
  device_id="raspi_node_01",
  location="Site_A"
  insect_count=4,
  temperature=23.5,
  humidity=55.2
  timestamp=2025-03-21T11:42:05Z
```

Data Retention Policies: Retention policies ensure the database remains lightweight and manageable:

- High-resolution data (1–10 sec): retained for ~14–30 days.
- Downsampled aggregates (5–15 min means): retained for months or years.
- Archival data: exported automatically to cloud bucket storage or CSV formats.

This multi-resolution storage model supports both fine-grained analysis and long-term epidemiological studies.

7. Dashboard Visualization and Analytics

The dashboard and analytics layer functions as the decision-support interface of the smart mosquito detection system. It transforms raw detection events, device telemetry, and environmental sensor measurements into actionable visual indicators that can be used by researchers, epidemiologists, and public-health authorities. This layer leverages a scalable cloud-based architecture combining InfluxDB, Prometheus, and Grafana, enabling comprehensive monitoring, historical trend analysis, and automated alerting.

The architecture is designed to operate continuously under field conditions, collecting high-frequency detection data from distributed IoT-enabled sensing nodes. These data streams are stored in time-series databases with millisecond-level precision and visualized through interactive dashboards that allow comparison across locations, time windows, and operational conditions. This structure supports standard public-health workflows such as hotspot detection, vector-density modeling, early warning for outbreak prevention, and intervention planning.

7.1. Grafana Dashboard Integration

The processed mosquito detection data, published via MQTT, is ingested into InfluxDB, which serves as the primary time-series backend for Grafana. MQTT messages are parsed by a collector (e.g., Telegraf), which writes the insect counts (numar_insecte) into a structured database schema that Grafana queries in real time.



Fig. 1. Example Grafana Panel Displaying Real-Time Mosquito Activity



Fig. 2. Another Example Grafana Panel Displaying Real-Time Mosquito Activity

These panels (Fig. 1, Fig. 2) show detection activity over time, aggregated at a defined temporal resolution.

8. Field Demonstrations and Results

To evaluate the effectiveness of the proposed smart mosquito detection system, a series of field demonstrations were carried out at BEIA Consult International's headquarters in Bucharest, Romania. These experiments were conducted using IoT-enabled edge nodes equipped with YOLOv8 object detection models, continuous MQTT data transmission, and integrated Grafana dashboards for real-time visualization. The primary objectives of the demonstrations were to validate detection accuracy, assess operational reliability under real conditions, evaluate communication performance, and examine the practical utility of the system's visualization and alerting pipeline.

The deployment took place in two distinct environments on the BEIA premises:

- office setting, providing a controlled environment for stability and performance testing, and
- an outdoor area, chosen to expose the system to fluctuating lighting, wind, and natural insect activity.

In both scenarios, a Raspberry Pi-based edge node operated autonomously, running continuous YOLOv8 inference and periodically publishing structured detection events to the cloud via MQTT. Data was logged locally for redundancy and stored remotely via InfluxDB for cloud-based analysis.

8.1. Deployment Scenario

The pilot deployment at BEIA headquarters consisted of a distributed configuration involving two primary sensing setups, each designed to test different operational conditions.

Indoor Monitoring Node (BEIA Office):

- Camera: Standard USB webcam
- Environment: BEIA indoor office, controlled lighting and limited background variability
- Purpose: Stability testing, algorithm refinement, validation under consistent visual constraints
- Deployment duration: Continuous operation for several days
- Output: Live detections, annotated images saved to disk, high-frequency MQTT telemetry

This indoor setup allowed precise evaluation of detection reliability under ideal and repeatable conditions, ensuring that the trained YOLOv8 model performed consistently.

Outdoor Monitoring Node (BEIA Outdoor Area)

- Camera: Foscam D4Z IP camera (RTSP-based acquisition)
- Environment: BEIA outdoor area, subject to varying illumination, reflections, and wind
- Purpose: Real-world validation under dynamic environment and spontaneous insect activity
- Deployment duration: Multi-hour continuous test during active insect periods
- Output: Time-series insect counts, cloud dashboard updates

The outdoor environment exposed the system to naturally occurring insects and environmental perturbations, providing a realistic assessment of robustness.

Throughout testing at BEIA headquarters, the system operated autonomously following the established processing workflow:

- Frame capture via USB (indoor) or RTSP (outdoor)
- YOLOv8 inference using the custom-trained best.pt model
- Insect counting
- Local logging into timestamped CSV files
- Event publication to MQTT broker
- Data ingestion into InfluxDB
- Visualization & alerting via Grafana dashboards

During the deployment at BEIA headquarters, the system generated logs in the following format:
Timestamp: 2025-03-20 | Insect Count: 1
Timestamp: 2025-03-20 | Insect Count: 2

Annotated frames were captured automatically when detections exceeded configurable thresholds, enabling visual inspection of results and verification of model behavior.

The testing at BEIA yielded the following observations:

- Moderate system stability: The Raspberry Pi nodes generally operated continuously, but several short-term performance drops were observed during extended operation. In particular, CPU load increased during bursts of rapid insect motion, occasionally causing brief slowdowns in inference speed. Although temperatures remained within normal ranges, thermal throttling cannot be excluded under longer deployments.
- Reliable real-time detection: YOLOv8 was able to detect insects in most frames; however, several misclassifications occurred, especially when insects were moving quickly or partially exiting the field of view. Some events included: missing insects during frames with motion blur, false positives triggered by shadows or moving leaves in the outdoor scenario, inconsistent bounding boxes when insects were very close to the camera.
- Variable inference latency: Average per-frame processing time supported near–real-time detection, but occasional latency spikes were recorded, especially when

the RTSP stream experienced buffering. These spikes sometimes caused frame drops or temporary delays in the detection loop.

- Consistent MQTT communication: The broker received updates at 1-second intervals with no measurable delay in Grafana visualization.
- Environmental sensitivity outdoors: While the system was generally robust to changes in natural light, challenging conditions introduced errors, including: detection failures during sudden lighting transitions (e.g., sun emerging from clouds), false detections caused by small debris or insects flying very close to the lens, blurred frames during windy conditions causing movement of vegetation in the background.

8.2. Detection Accuracy and Performance

The detection accuracy and performance of the smart mosquito monitoring system were evaluated during the indoor and outdoor deployments at BEIA headquarters. The goal was to assess not only the strengths of the system but also the practical limitations and error modes encountered during real-world operation. The analysis focuses on three main aspects:

- detection accuracy under varying conditions,
- inference and system performance, and
- communication responsiveness and stability.

In the controlled indoor environment, the YOLOv8 model achieved consistent detection results, especially when insects were within the optimal focal distance and moving relatively slowly. However, several limitations emerged:

- Motion blur occasionally caused missed detections when insects moved rapidly across the frame.
- The model sometimes detected small artifacts or dust particles as insects, particularly on reflective surfaces.
- When insects approached the camera very closely, YOLOv8 occasionally produced unstable bounding boxes due to scale changes and partial occlusion.

Despite these issues, the indoor detection rate remained stable enough to support continuous monitoring tasks.

Outdoor detection accuracy was more variable due to environmental factors. The main error sources included:

- Background motion, such as vegetation swaying in the wind, leading to several false positives.
- Lighting variability, including shadows, glare, and sudden brightness changes, caused temporary degradation in confidence scores.
- Low-contrast conditions during late-evening tests made insects harder to detect, resulting in a noticeable increase in missed detections.

Real-time inference was achieved on the Raspberry Pi 4, but the performance was not uniform throughout the deployment.

Observed Performance Characteristics

- Under stable indoor conditions, inference ran smoothly at an estimated 2–5 FPS, depending on resolution.
- During periods of intense activity or heavier background noise outdoors, the system experienced latency spikes, temporarily reducing the frame rate.
- The RTSP-based outdoor node was more susceptible to stream buffering, which occasionally delayed frame delivery by several hundred milliseconds.

MQTT-based communication generally performed well, with consistent 1-second update intervals during stable network conditions. However, several issues were observed:

- Short network interruptions occasionally caused delayed message delivery or required reconnection.
- During inference-heavy periods, publishing sometimes lagged behind by 1–2 seconds due to CPU contention.
- InfluxDB ingestion remained stable, but Grafana panels sometimes displayed temporary gaps due to momentary data bursts or network jitter.

Despite these minor issues, the communication layer remained operational throughout the deployment, and the system recovered gracefully after interruptions.

The system performed adequately for the purpose of validating the feasibility of automated mosquito detection using edge AI and IoT infrastructure. While the model reliably detected insects across multiple conditions, false positives, missed detections, and performance variability were observed, especially in dynamic outdoor settings. These results align with expectations for early-stage field deployments and highlight opportunities for improvement, such as:

- enhanced motion compensation,
- adaptive confidence filtering,
- low-light imaging enhancements,
- more robust background modeling,
- and optimized streaming pipelines.

8.3. Predictive Modeling Potential

Although the primary focus of the current deployment at BEIA headquarters was to validate real-time detection, communication, and visualization functionality, the data collected demonstrate promising potential for integrating predictive modeling into the mosquito surveillance system. Such predictive capabilities are essential for transitioning from reactive monitoring to proactive vector-control strategies, particularly in the context of climate variability and urban environmental changes.

The system already incorporates the foundational elements required for predictive analysis: continuous mosquito activity data, environmental measurements, time-series storage, and scalable analytics infrastructure. These components enable the exploration of statistical relationships between insect activity and environmental drivers, supporting the development of early-warning models and risk-mitigation tools.

9. Conclusions

This paper presented the design, implementation, and initial evaluation of a smart mosquito detection system that combines AI-based computer vision, IoT communication, edge computing, and cloud-based analytics for vector surveillance. The proposed architecture integrates heterogeneous imaging devices, Raspberry Pi edge nodes running YOLOv8 object detection models, MQTT-based communication, and Grafana dashboards backed by InfluxDB/Prometheus time-series storage. Together, these components form an end-to-end pipeline capable of transforming raw image streams into actionable entomological indicators suitable for operational public-health monitoring.

From a methodological perspective, the work demonstrates that lightweight deep-learning models can be deployed directly on embedded hardware to perform near–real-time mosquito detection. The use of YOLOv8, trained on a curated and augmented dataset combining public and custom imagery, enables local inference without streaming raw video to the cloud, thereby reducing bandwidth requirements and enhancing data privacy. The modular design of the software, in particular the separation of inference, logging, communication, and visualization, facilitates extensibility, allowing new sensing modalities (e.g., environmental or acoustic inputs) and alternative models to be incorporated with minimal reconfiguration.

The field demonstrations performed at BEIA Consult International's headquarters indicate that the system is operationally viable under both controlled indoor and more challenging outdoor conditions. The edge nodes maintained continuous monitoring, successfully generating time-stamped insect counts and transmitting them to cloud dashboards with low to moderate latency. The Grafana-based visualization layer proved useful for examining real-time activity, validating detection events, and exploring temporal patterns. At the same time, the tests revealed realistic limitations, including sensitivity to motion blur, background movement, lighting variability, and occasional communication interruptions. These limitations are consistent with early-stage deployments of computer-vision and IoT systems in non-laboratory environments.

Despite these constraints, the results confirm that the system is adequate for continuous monitoring and for demonstrating the feasibility of automated mosquito detection using edge AI and IoT infrastructure. Importantly, the architecture already incorporates the key elements required for future predictive modeling: high-frequency detection data, environmental context, and scalable time-series storage. With longer deployments across multiple sites and richer environmental sensing, the platform could be extended to support short-term forecasting of mosquito activity, hotspot detection, and integration with epidemiological risk models.

From a public-health and smart-city standpoint, the proposed system offers several advantages over traditional manual surveillance: it reduces dependence on labor-intensive trap inspections, enables earlier detection of vector activity, and supports more targeted interventions. By providing an interoperable, scalable, and relatively low-cost solution, the system can be integrated into broader environmental and urban-monitoring infrastructures, contributing to data-driven vector-control strategies and climate-resilient health planning.

Future work will focus in several directions. First, improving detection robustness through advanced tracking, motion compensation, and low-light imaging is essential for large-scale outdoor deployments. Second, incorporating acoustic wingbeat analysis and additional environmental sensors will strengthen multimodal fusion and reduce false positives. Third, extended pilot deployments in diverse geographic and climatic contexts will be necessary to validate generalizability and to calibrate predictive models. Finally, closer integration with public-health stakeholders and information systems will be crucial to translating technical capabilities into operational decision support and measurable impact on vector-borne disease prevention.

In conclusion, this work shows that integrating AI, IoT, and edge analytics into a unified mosquito detection ecosystem is technically feasible and operationally promising. While further refinement and large-scale validation are required, the system represents a concrete step toward proactive, data-driven vector surveillance and contributes to the broader vision of smart, resilient public-health infrastructures.

Acknowledgements
The study was supported by the Europe Horizon project MOBILISE (GA# 101073982).

References

[1] Lakyiere, Alice Bagyiereyele, et al. "Trends and advances in image-based mosquito identification and classification using machine learning models: A systematic review." *Computers in Biology and Medicine* 193 (2025): 110373.
[2] Nolte, Kristopher, et al. "Potentials and limitations in the application of Convolutional Neural Networks for mosquito species identification using wing images." *PLOS Computational Biology* 21.9 (2025): e1013435.
[3] Nolte, Kristopher, et al. "Robust mosquito species identification from diverse body and wing images using deep learning." *Parasites & vectors* 17.1 (2024): 372.
[4] Al Maruf, Abdullah, et al. "TransembleNet: Enhancing vector mosquito species classification through transfer learning-based ensemble model." *PLoS One* 20.5 (2025): e0322171.
[5] Karim, Ahmed Akib Jawad, Muhammad Zawad Mahmud, and Riasat Khan. "Advanced vision transformers and open-set learning for robust mosquito classification: A novel approach to entomological studies." *PLOS Computational Biology* 20.12 (2024): e1012654.
[6] Joelianto, Endra, et al. "Convolutional neural network-based real-time mosquito genus identification using wingbeat frequency: A binary and multiclass classification approach." *Ecological Informatics* 80 (2024): 102495.
[7] Hou, Yuanbo, et al. "Learning Domain-Robust Bioacoustic Representations for Mosquito Species Classification with Contrastive Learning and Distribution Alignment." *arXiv preprint arXiv:2510.00346* (2025).
[8] Sayeedi, Md Faiyaz Abdullah, Fahim Hafiz, and Md Ashiqur Rahman. "MosquitoFusion: a multiclass dataset for real-time detection of mosquitoes, swarms, and breeding sites using deep learning." *arXiv preprint arXiv:2404.01501* (2024).
[9] Gao, Yuanyi, et al. "Application of machine learning in automatic image identification of insects-a review." *Ecological Informatics* 80 (2024): 102539.
[10] Gupta, Khushi Anil, et al. "Mosquito species identification accuracy of early deployed algorithms in IDX, A vector identification tool." *Acta Tropica* 260 (2024): 107392.
[11] Li, Jun, and Hui Lu. "Mosquito Species Identification and Phylogenetics: A Global Perspective." *Journal of Mosquito Research* 14 (2024).
[12] DANILA, Andrei, et al. “IoT Systems for Efficient Preservation of Cultural Heritage”. *Smart Cities International Conference (SCIC) Proceedings*, vol. 11, June 2024, pp. 515-26, https://www.scrd.eu/index.php/scic/article/view/515.

[13] VOICU, Ioan Florin, and Daniel Constantin DIACONU. “Monitoring City Water Incidents via an Internet of Things-Based Sensor Network”. *Smart Cities International Conference (SCIC) Proceedings*, vol. 10, Oct. 2023, pp. 207-14, https://scrd.eu/index.php/scic/article/view/436.
[14] SUCIU, George, and Cosmina STALIDI. “Digital Twins, the Software Solution for Safer Cities”. *Smart Cities International Conference (SCIC) Proceedings*, vol. 10, Oct. 2023, pp. 273-82, https://scrd.eu/index.php/scic/article/view/443.