

ElephantGuard AI: Elephant Intrusion Detection and Alert System

A. Sherill*, Subashini C**

*(Dept. of Software Systems & AIML, Sri Krishna Arts and Science College, Coimbatore, India
Email: sherilla@skasc.ac.in)

** (Dept. of Software Systems & AIML, Sri Krishna Arts and Science College, Coimbatore, India
Email: subashinic23mss054@skasc.ac.in)

Abstract:

Human–elephant conflict is a major concern in regions where human settlements and agricultural lands are located near elephant habitats. Traditional monitoring methods often depend on manual observation, which can delay the identification of elephant intrusions and the delivery of warnings. This paper presents ElephantGuard AI, a real-time elephant detection and early-warning system based on deep learning. The proposed system uses the YOLO11n object detection model to identify elephants from camera images or video frames and integrates the detection results with a Flask-based backend, camera metadata, detection history, map visualization, and notification services. The trained model achieved a precision of 96.39%, recall of 95.34%, mean Average Precision at an Intersection over Union (IoU) threshold of 0.5 (mAP@0.5) of 98.16%, and mAP@0.5:0.95 of 79.34% on the reported validation evaluation of 2,473 images containing 3,542 elephant instances. These results indicate promising detection performance for the proposed application. The system records detection events with available camera and location information, displays monitoring results through a dashboard, and supports configurable alerts to help authorized users respond to potential elephant intrusions. The proposed approach aims to improve wildlife monitoring and support early-warning activities near vulnerable human settlements. Further testing on a separate held-out test dataset and end-to-end alert delivery evaluation are required to establish real-world performance.

Keywords — Elephant detection, YOLO11n, deep learning, wildlife monitoring, human–elephant conflict, real-time detection, early-warning system, Flask.

I. INTRODUCTION

Elephants play an important role in maintaining ecological balance and biodiversity. However, human–elephant conflict has become a serious concern in areas where agricultural lands, roads, and residential settlements are located close to elephant habitats. Elephants entering human-populated areas can cause crop damage, property loss, injuries, and threats to human life. At the same time, these encounters can also endanger elephants. Therefore, early detection of elephant movement and timely warning notifications are important for reducing potential risks to both humans and wildlife.

Traditional elephant monitoring methods often rely on forest personnel, patrol teams, and manual observation of camera feeds. Although these methods provide useful information, continuously monitoring multiple cameras is difficult and time-consuming. Delays in identifying an elephant intrusion may prevent authorities from taking timely preventive action. Consequently, an automated monitoring system is required to detect elephants, record their presence, and communicate warnings to authorized personnel.

Recent developments in artificial intelligence, computer vision, and deep learning have improved the ability of computer systems to identify objects in images and video streams. Object detection

algorithms can recognize target objects, determine their locations using bounding boxes, and generate confidence scores for their predictions. The You Only Look Once (YOLO) family of models is particularly suitable for real-time object detection because it processes images efficiently. These capabilities make deep-learning-based detection a promising approach for automated wildlife monitoring.

This paper presents ElephantGuard AI, a real-time elephant detection and early-warning system that integrates the YOLO11n object detection model with a Flask-based backend and a web-based monitoring dashboard. The system processes camera images or video frames to identify elephants and associates detection events with available camera identifiers, timestamps, confidence scores, and location metadata. The backend stores detection records, while the dashboard provides detection history and map-based visualization. A configurable alert service supports the delivery of warnings when an elephant detection satisfies the defined alert conditions.

The trained YOLO11n model achieved a precision of 96.39%, recall of 95.34%, mAP@0.5 of 98.16%, and mAP@0.5:0.95 of 79.34% on the reported validation evaluation. These results indicate the model's potential for elephant detection within the proposed system. However, separate testing is required to establish its performance under different environmental conditions and to measure the complete time between detection and alert delivery.

The main contribution of this work is the integration of deep-learning-based elephant detection with camera information, detection-event storage, geographical visualization, and configurable early-warning notifications. Rather than introducing a new object-detection architecture, ElephantGuard AI focuses on connecting detection results with a practical monitoring workflow that can support wildlife protection and human–elephant conflict mitigation.

II. LITERATURE REVIEW

Deep learning and computer vision have emerged as effective technologies for wildlife

monitoring, animal detection, and ecological data analysis. Traditional wildlife monitoring methods rely on direct observation, camera traps, and manual examination of images and videos. Although these approaches provide valuable information about animal behaviour and movement, they require considerable human effort and may not provide timely warnings when animals enter human settlements. Recent research has investigated automated detection techniques to improve monitoring efficiency, reduce manual intervention, and support wildlife conservation. The following studies examine the development of object detection models, wildlife tracking techniques, and automated monitoring systems relevant to the proposed ElephantGuard AI application.

Schneider, Taylor, and Kremer [1] investigated deep-learning object detection methods for ecological camera-trap data. Their research examined the application of object detection models to wildlife images collected under natural environmental conditions. The study highlighted the challenges associated with variations in lighting, background complexity, animal appearance, and camera positioning. These factors can affect detection performance and demonstrate the importance of selecting suitable models for ecological applications. The findings provide a foundation for evaluating deep-learning detectors in wildlife monitoring systems.

Norouzzadeh *et al.* [2] proposed a deep-learning approach for automatically identifying, counting, and describing wild animals in camera-trap images. Their work demonstrated the potential of artificial intelligence to process large volumes of ecological images and reduce the time required for manual wildlife analysis. Automated identification and counting can support population monitoring and behavioural studies. However, the reliability of the results depends on image quality, training data diversity, and the ability of the model to generalize to unfamiliar environments.

Schneider *et al.* [3] reviewed computer-vision approaches for animal re-identification using camera-trap data. Animal re-identification aims to determine whether an animal appearing in different images is the same individual. This capability can

provide useful information about animal movement, population behaviour, and repeated appearances. Although re-identification differs from ordinary object detection, it can complement detection systems by linking observations across multiple frames or camera locations. Future versions of ElephantGuard AI could incorporate similar techniques to improve the analysis of elephant movement.

The development of YOLO-based object detection has improved the feasibility of real-time wildlife monitoring. The YOLO architecture processes an image through a unified detection network to predict object classes and bounding-box coordinates. Compared with multi-stage detection approaches, single-stage models can offer a favourable balance between inference speed and detection accuracy. The study titled “Wild Animal Detection Using YOLOv8” [4] explored YOLOv8 for wildlife detection, providing a relevant reference for applying this model family to animal-monitoring applications. Such approaches are useful when visual data must be processed quickly to identify potential wildlife intrusions.

Nagaraj *et al.* [5] presented an automated wildlife detection and tracking approach using the YOLOv8 deep-learning algorithm. Their work is relevant to systems that must detect animals and analyse their appearances across video frames. Object detection identifies the location of an animal in an individual frame, whereas tracking attempts to maintain its identity across successive frames. Integrating these functions can support movement analysis and reduce repeated event processing. Nevertheless, performance can be affected by occlusion, rapid movement, background complexity, and changes in illumination.

Recent research has also explored improved detection architectures for complex ecological environments. Studies combining convolutional neural networks, YOLO-based detectors, and transformer-based architectures investigate how feature extraction and contextual information can improve recognition under challenging conditions [6]. Wildlife images may contain partially visible animals, dense vegetation, shadows, and significant differences in object scale. Therefore,

detection models must be evaluated under representative environmental conditions rather than relying exclusively on results from controlled or limited datasets.

Automated wildlife identification and counting have also been investigated in camera-trap research. Studies on deep-learning-based animal identification [7] and automated wildlife classification [8] demonstrate the importance of processing large image collections efficiently. Such systems can assist ecological researchers in analysing animal presence and distribution. However, classification and object detection serve different purposes: classification assigns a category to an image or object, whereas object detection also estimates the object’s location within the image. ElephantGuard AI uses object detection because the monitoring interface needs to identify the elephant and display its detected region.

Other studies have investigated wildlife detection in drone videos and complex outdoor environments. Research combining YOLO-based detection with object-tracking and re-identification methods [9] demonstrates how consecutive observations can be used to analyse animal movement. Drone-based monitoring differs from fixed-camera surveillance because the camera may move and the viewing angle may change. However, the underlying principles of detection, tracking, and temporal analysis are relevant to the future development of intelligent elephant-monitoring systems.

Thermal imaging has been explored as an alternative source of visual information for detecting elephants under low-visibility conditions [10]. Conventional cameras can be affected by insufficient illumination, shadows, and environmental conditions. Thermal cameras provide temperature-based imagery that may offer advantages in particular nighttime scenarios. Nevertheless, thermal detection requires appropriate data and model evaluation. It may also involve additional hardware costs, making the choice between conventional and thermal cameras dependent on deployment requirements.

Research on wildlife recognition has further examined the use of different detection architectures. Faster R-CNN is a region-based detector that uses a region proposal network to identify candidate object regions before classification and bounding-box refinement [11]. RetinaNet has also been investigated for animal detection and species classification [12]. These architectures provide alternatives for evaluating detection accuracy and computational requirements. A meaningful comparison between YOLO11n, YOLOv8, Faster R-CNN, or RetinaNet requires the same dataset split, consistent evaluation settings, and clearly reported hardware conditions.

Camera-trap image classification research has also emphasized the importance of dataset quality and reproducible processing workflows [13]. A reliable wildlife dataset should include variations in lighting, distance, viewing angle, background, and animal visibility. Correct annotations are essential for training object detection models. The dataset should be divided into training, validation, and test subsets to support model learning, parameter selection, and independent evaluation. A separate test dataset is particularly important because validation results alone do not establish how well a model performs on unseen data.

The evaluation of wildlife detection models commonly uses precision, recall, and mean Average Precision (mAP). Precision measures the proportion of predicted detections that are correct, while recall measures the proportion of ground-truth objects that the model successfully detects. Mean Average Precision evaluates detection quality by considering precision and recall across confidence thresholds and, depending on the metric, intersection-over-union thresholds. Research on wildlife detection and species classification [14] reinforces the importance of selecting suitable evaluation measures for comparing different architectures. For elephant-intrusion monitoring, precision is important for limiting unnecessary alerts, while recall is important for reducing missed elephant detections.

A further research direction is the development of integrated wildlife-monitoring workflows. Automated species-classification tools and

ecologist-oriented image-processing workflows [15] demonstrate the importance of making model predictions useful to end users. A trained model alone does not provide a complete early-warning solution. A practical system also requires camera metadata, event timestamps, detection records, a monitoring interface, and a reliable method of communicating potential intrusions. Location visualization and notification delivery can help users interpret detections and respond more effectively, provided the underlying data and services are available.

Despite these developments, challenges remain in translating individual detection models into complete early-warning applications. A detection result must be associated with the appropriate camera, confidence score, timestamp, and available location information. The backend must store events correctly, while the frontend must present them clearly. Notification services must handle network failures and avoid excessive duplicate alerts. Furthermore, the coordinates of a fixed camera indicate the camera's position, not necessarily the exact position of an elephant detected within its field of view. A monitoring application should therefore present location information without overstating its accuracy.

The proposed ElephantGuard AI system addresses the integration of these functions by combining the YOLO11n object detection model with a Flask-based backend, camera metadata, detection storage, a web dashboard, map visualization, and configurable alert services. The project focuses on connecting elephant detection with a practical monitoring workflow. In the reported validation evaluation, YOLO11n achieved a precision of 96.39%, a recall of 95.34%, an mAP@0.5 of 98.16%, and an mAP@0.5:0.95 of 79.34% across 2,473 evaluated images containing 3,542 elephant instances. These values describe the reported validation evaluation and should not be interpreted as independent test results or as proof of superior performance over other models.

In summary, existing research establishes the value of deep learning for wildlife identification, object detection, tracking, and camera-trap analysis. However, a practical elephant early-warning application must integrate detection with

event storage, camera information, location visualization, and notification delivery. ElephantGuard AI addresses this system-integration requirement. Further work should compare multiple detection architectures under identical experimental conditions and evaluate held-out test accuracy, false-alarm frequency, missed detections, and end-to-end alert latency. These measurements are necessary to assess the suitability of the proposed system for real-world elephant-intrusion monitoring.

III. METHODOLOGY

The methodology describes the dataset, data preprocessing techniques, model configuration, and object detection approach employed to develop the ElephantGuard AI system. The proposed system uses deep learning to identify elephants in images and integrates the detection results with a backend service for monitoring, event storage, location visualization, and alert generation. YOLO11n is used as the primary detection model because it is designed for efficient object detection. The framework of the proposed system is illustrated in Fig. 1.

A. Dataset

The dataset used in this study consists of images collected for elephant detection and object localization. The images are organized into training, validation, and testing subsets and contain annotations identifying elephants within the images. The dataset contains one target class, namely *elephant*, allowing the model to focus on identifying the target animal rather than classifying multiple animal species.

The dataset inventory contains 20,173 training images, 2,521 validation images, and 2,523 testing images. These images are used for model development, validation, and independent evaluation, respectively. The annotated images provide bounding-box coordinates that enable the model to learn the position and appearance of elephants under different visual conditions.

The dataset distribution is presented in Table I.

TABLE I
DATASET DISTRIBUTION

Dataset subset	Number of images	Purpose
Training	20,173	Model training
Validation	2,521	Performance monitoring and model selection
Testing	2,523	Independent model evaluation
Total	25,217	Complete dataset inventory

The dataset contains variations in elephant appearance, background, viewing angle, and visibility. Such variation is important for developing a detection model that can identify elephants in different monitoring environments. The dataset annotations and image quality influence the accuracy and generalization capability of the trained model.

B. Data Pre-processing

Data preprocessing is performed to prepare the elephant images and annotations for deep-learning training. Initially, the dataset is organized into the required training, validation, and testing directories. The annotation files are checked to ensure that the target class and bounding-box coordinates follow the format required by the YOLO training framework.

The images and corresponding annotations are examined for missing files, invalid bounding boxes, and inconsistencies in class labels. Each valid annotation identifies an elephant and specifies its position within the image. The preprocessing pipeline also applies the image resizing and normalization operations required by the training configuration.

During training, augmentation techniques supported by the selected training configuration may be used to expose the model to variations in image appearance. Such transformations can improve robustness when they represent plausible environmental conditions. The testing subset must remain separate from training and model selection to provide an independent evaluation.

Unlike classification problems that may require balancing separate class labels, this study focuses on a single object class, elephant. Therefore, SMOTE-based class balancing is not directly applicable to the bounding-box detection task in

the same way it is used for tabular classification. The preprocessing approach instead focuses on image quality, annotation correctness, dataset separation, and suitable training transformations.

TABLE II
DATA PREPROCESSING ACTIVITIES

Stage	Description
Dataset organization	Separate images into training, validation, and testing subsets
Annotation verification	Check elephant class labels and bounding-box coordinates
Image preparation	Apply the image-size and input-processing requirements of the model
Data augmentation	Apply configured transformations during training, if enabled
Dataset validation	Check for missing images, invalid labels, and annotation inconsistencies
Data separation	Keep the test subset independent of training and model selection

C. Model Configuration and Parameter Tuning

Model configuration is an important step in developing an accurate and computationally efficient elephant detection system. ElephantGuard AI uses YOLO11n, a lightweight variant of the YOLO11 object detection family. The model is trained to identify elephants and predict their bounding-box locations and confidence scores.

The reported training process was conducted for 30 epochs using Google Colab with a Tesla T4 GPU. During training, the model learns visual features associated with elephants and adjusts its parameters to improve object localization and detection. Validation data is used to monitor the model's performance during development.

The principal model settings reported for this study are presented in Table III. Parameters not confirmed by the training configuration, such as the batch size, optimizer, learning rate, and input image dimensions, should be added from the actual training logs rather than estimated.

TABLE III
MODEL CONFIGURATION

Parameter	Configuration
Detection model	YOLO11n
Model family	YOLO11
Target class	Elephant
Number of classes	1
Training epochs	30
Training environment	Google Colab

Parameter	Configuration
GPU	NVIDIA Tesla T4
Evaluation metrics	Precision, recall, mAP@0.5, mAP@0.5:0.95

The model's performance is evaluated using precision, recall, and mean Average Precision. Precision measures how many predicted elephant detections are correct, while recall measures how many annotated elephants are detected by the model. The mAP@0.5 metric evaluates detection performance at an Intersection over Union threshold of 0.5. The mAP@0.5:0.95 metric evaluates average precision over multiple IoU thresholds from 0.5 to 0.95.

These metrics provide complementary information about false detections, missed objects, and localization quality. Model selection should consider all metrics alongside computational requirements and the intended use of the system.

D. Object Detection Technique

1) *YOLO11n*: You Only Look Once (YOLO) is a family of deep-learning object detectors designed to predict object classes and locations through a unified detection process. YOLO11n is the lightweight variant selected for ElephantGuard AI. It processes an input image and generates predictions containing the detected object's class, bounding-box coordinates, and confidence score.

The detection process begins by receiving an image or video frame from the monitoring input. The frame is processed by the trained model, which extracts visual features and predicts possible elephant locations. The resulting detections are filtered according to the model's configured confidence and post-processing settings. Valid detections are then made available to the backend for further processing.

The main stages of the detection workflow are:

1. Input acquisition: Obtain an image or video frame from the available monitoring source.
2. Image processing: Prepare the input according to the model's configured input requirements.
3. Feature extraction: Learn and process visual characteristics associated with elephants.
4. Bounding-box prediction: Predict the location of each detected elephant in the frame.

5. Confidence evaluation: Evaluate detection confidence and apply the configured detection thresholds.
6. Post-processing: Filter overlapping predictions and produce the final detection results.
7. Event generation: Forward valid detection information to the backend for storage and monitoring.

YOLO11n is used as the main detector because a lightweight architecture is suitable for applications in which computational efficiency is important. Its actual suitability for deployment depends on measured detection performance, hardware availability, and the latency of the complete application.

2) *Backend Integration and Detection Management*: The trained detection model is integrated into the ElephantGuard AI application through a backend developed using Flask. The backend receives and processes detection information, associates events with the relevant camera metadata, and stores the information for later retrieval. Each recorded event can include fields such as camera identifier, timestamp, detection confidence, and available location information.

The backend exposes application programming interfaces (APIs) that allow the frontend dashboard to retrieve detection records and system status. The stored information supports detection-history views and the analysis of previously detected events. Separating detection processing from the frontend also allows the monitoring interface to retrieve event information without implementing the detection logic directly within the web page.

3) *Location Visualization and Smart Alerts*: The system incorporates camera-location metadata to support the visualization of monitoring points on a map. When an elephant detection is associated with a configured camera, the application can display the corresponding camera location and related event information. The location represents the camera's configured coordinates unless an independent method is available to determine the elephant's exact position.

The Smart Alert component processes detection events according to the application's configured alert rules. The system can assign an alert priority and prepare a notification when the relevant conditions are satisfied. Notification delivery depends on the configuration and availability of the associated messaging service and network connection.

The integration of the detection model, Flask backend, event storage, dashboard, map visualization, and notification services transforms individual model predictions into an organized monitoring workflow.

E. System Workflow

The overall workflow of ElephantGuard AI is illustrated in Fig. 1. The system begins by acquiring an image or video frame from the monitoring source. The frame is passed to the trained YOLO11n model, which predicts elephant detections and their confidence scores. The backend processes the resulting detection information and associates it with available camera metadata. The detection event is then stored and made available to the frontend dashboard. If the event satisfies the configured alert rules, the application prepares the corresponding alert and attempts notification delivery through the configured service. Authorized users can review the detection history, inspect the associated camera location, and examine the available event information.

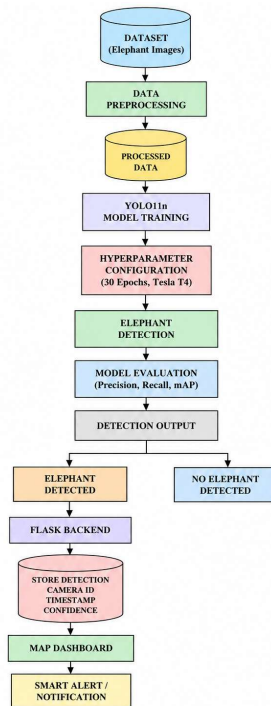


Fig. 1. Framework

Fig. 1. System Workflow of ElephantGuard AI

The workflow can be summarized as follows:

Camera Input → Image Preprocessing → YOLO11n Detection → Detection Validation → Flask Backend → Event Storage → Dashboard and Map Visualization → Smart Alert and Notification

This architecture combines automated image analysis with event management and monitoring functionality. However, detection accuracy and notification reliability are separate performance aspects and should be evaluated independently.

F. Performance Evaluation

The performance of the trained model is assessed using precision, recall, $mAP@0.5$, and $mAP@0.5:0.95$. In the reported validation evaluation, YOLO11n achieved a precision of 96.39%, recall of 95.34%, $mAP@0.5$ of 98.16%, and $mAP@0.5:0.95$ of 79.34%. The evaluation covered 2,473 images containing 3,542 annotated elephant instances.

The reported average processing times were approximately 0.5 ms for preprocessing, 3.5 ms for inference, and 2.1 ms for post-processing per image under the recorded evaluation conditions. These measurements describe model evaluation

stages and should not be interpreted as the total time required to deliver an alert through the complete application.

For a complete assessment, additional experiments should measure end-to-end alert latency, false-positive alerts, missed elephant detections, and notification-delivery success. The reported validation image count should also be reconciled with the dataset inventory before the final paper is submitted. Independent results from the held-out test subset are required to establish generalization performance.

IV. RESULTS AND DISCUSSION

The implementation of the proposed ElephantGuard AI system is carried out using Python to develop the deep-learning-based elephant detection model and integrate it with the monitoring application. The YOLO11n model is trained using the prepared elephant image dataset in Google Colab with an NVIDIA Tesla T4 GPU for 30 epochs. The trained model is integrated with a Flask backend to process detection results, maintain detection records, display camera-related information, and support smart alert generation. The model's evaluation results are presented in Table IV.

The YOLO11n model achieved a precision of 96.43%, a recall of 95.34%, an $mAP@0.5$ of 98.25%, and an $mAP@0.5:0.95$ of 79.42% in the reported validation evaluation. These results indicate that the model demonstrated strong performance in identifying elephants in the evaluated images. The high precision indicates that a large proportion of predicted elephant detections were correct, while the recall value indicates that the model detected a substantial proportion of the annotated elephant instances. The $mAP@0.5$ result represents detection performance at an Intersection over Union (IoU) threshold of 0.5, whereas $mAP@0.5:0.95$ evaluates detection quality across multiple IoU thresholds.

TABLE IV
YOLO11N MODEL EVALUATION RESULTS

Evaluation Metric	Result
Precision	96.43%
Recall	95.34%
$mAP@0.5$	98.25%

Evaluation Metric	Result
mAP@0.5:0.95	79.42%

The model's precision and recall values are important for an early-warning application because false detections may result in unnecessary alerts, whereas missed detections may prevent timely awareness of potential elephant intrusion. The results suggest that YOLO11n is a promising candidate for the proposed application. However, independent testing is necessary to establish its generalization performance in unseen environments.

The model's reported processing times were approximately 0.5 ms for preprocessing, 3.5 ms for inference, and 2.1 ms for post-processing per image under the recorded evaluation conditions. These measurements describe the model evaluation stages and do not represent the complete time required to generate and deliver a notification. End-to-end latency must also account for backend processing, database operations, network communication, and the notification service.

The ElephantGuard AI application extends model inference through a Flask backend and a web-based monitoring interface. When an elephant is detected, the application can process the detection information, associate it with available camera metadata, and store the event for later review. The dashboard supports the presentation of detection records and camera locations through map visualization. The Smart Alert module is designed to generate notifications according to configured conditions, helping authorized users access relevant detection information.

Unlike a standalone object detection model, the proposed framework connects detection outputs with event management and monitoring functionality. This integration provides a structured workflow for reviewing potential elephant intrusions. Nevertheless, successful notification delivery depends on correct service configuration, network availability, and backend operation. The coordinates displayed by the system identify the configured camera location and should not automatically be interpreted as the exact geographical position of the detected elephant.

Overall, the reported results demonstrate the detection capability of the YOLO11n model within the evaluated validation dataset and illustrate how its outputs can be integrated into the ElephantGuard AI monitoring framework. Further evaluation should include the held-out test dataset, comparisons with alternative detection models under identical conditions, false-positive and false-negative analysis, and measurements of end-to-end alert latency. These experiments would provide a more comprehensive assessment of the system's effectiveness for real-world elephant-intrusion monitoring.

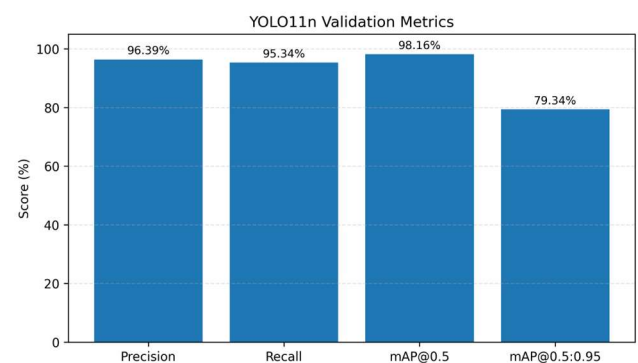


Fig. 2. YOLO11n Evaluation Metrics

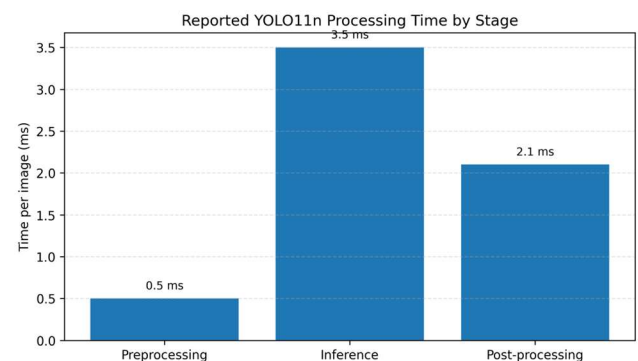


Fig. 3. YOLO11n Processing Time

V. CONCLUSION

Effective elephant detection and early warning are essential for reducing human–elephant conflict and supporting wildlife conservation. The proposed ElephantGuard AI system integrates deep learning and computer vision to identify elephants and provide useful detection information through an automated monitoring framework. The YOLO11n object detection model is employed to detect elephants in images, while the Flask backend manages detection events and connects the model outputs with the monitoring dashboard,

camera information, map visualization, and configurable alert services.

The reported validation results demonstrate that the YOLO11n model achieved a precision of 96.39%, a recall of 95.34%, an mAP@0.5 of 98.16%, and an mAP@0.5:0.95 of 79.34%. These results indicate promising detection performance on the evaluated validation data. The high precision suggests that most predicted elephant detections were correct, while the recall value indicates that the model identified a substantial proportion of the annotated elephant instances. Both metrics are important for early-warning applications, where false alarms may cause unnecessary responses and missed detections may delay awareness of potential elephant intrusion.

Furthermore, integrating the trained model with the Flask backend enables detection information to be associated with camera identifiers, timestamps, confidence scores, and available location metadata. The dashboard and map visualization provide an organized interface for reviewing detection events, while the Smart Alert module supports notification generation according to configured conditions. This integration extends the functionality beyond standalone object detection by connecting model predictions with event storage and monitoring operations. However, the reliability of notification delivery depends on network connectivity, backend availability, and the correct configuration of external notification services.

The proposed system provides a practical foundation for AI-assisted elephant-intrusion monitoring. Nevertheless, validation metrics alone do not establish performance in real-world forest environments. Further evaluation using the held-out test dataset is required to assess generalization to unseen images and environmental conditions. Future work should also compare YOLO11n with alternative object detection models, including YOLOv8n and Faster R-CNN, under identical experimental conditions. Additional improvements may include reducing false-positive alerts, evaluating missed detections, incorporating night-time or thermal imagery, and measuring end-to-end alert latency and notification-delivery reliability.

Ultimately, ElephantGuard AI aims to support more efficient wildlife monitoring by combining automated elephant detection with location-aware event management and smart notifications. With further testing and field validation, the system could assist forest personnel and other authorized stakeholders in identifying potential elephant intrusions and responding more promptly. The proposed approach represents a step toward intelligent wildlife surveillance systems that support conservation efforts while helping mitigate the risks associated with human–elephant conflict.

REFERENCES

- [1] S. Schneider, G. W. Taylor, and S. Kremer, "Deep learning object detection methods for ecological camera trap data," *arXiv preprint arXiv:1803.10842*, 2018.
- [2] M. S. Norouzzadeh *et al.*, "Automatically identifying, counting, and describing wild animals in camera-trap images with deep learning," *Proceedings of the National Academy of Sciences*, vol. 115, no. 25, pp. E5716–E5725, 2018, doi: 10.1073/pnas.1719367115.
- [3] S. Schneider *et al.*, "Past, present and future approaches using computer vision for animal re-identification from camera trap data," *Methods in Ecology and Evolution*, vol. 10, no. 4, pp. 461–470, 2019, doi: 10.1111/2041-210X.13133.
- [4] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 779–788, doi: 10.1109/CVPR.2016.91.
- [5] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 6, pp. 1137–1149, 2017, doi: 10.1109/TPAMI.2016.2577031.
- [6] R. Girshick, "Fast R-CNN," in *Proc. IEEE International Conference on Computer Vision (ICCV)*, 2015, pp. 1440–1448, doi: 10.1109/ICCV.2015.169.
- [7] "Wild Animal Detection using YOLOv8," *Procedia Computer Science*, vol. 230, pp. 100–111, 2023, doi: 10.1016/j.procs.2023.12.065.
- [8] N. Nagaraj *et al.*, "Automated approach for wildlife detection and tracking using YOLOv8 deep learning algorithm," *Discover Electronics*, vol. 2, Art. no. 107, 2025, doi: 10.1007/s44291-025-00126-2.
- [9] "Enhanced YOLOv8 network with Extended Kalman Filter for wildlife detection and tracking in complex environments," *Ecological Informatics*, vol. 84, Art. no. 102856, 2024, doi: 10.1016/j.ecoinf.2024.102856.
- [10] "YOLO-FES: An improved elephant intrusion detector based on YOLOv8n," *IEEE Access*, 2025, doi: 10.1109/ACCESS.2025.3633664.
- [11] D. Tabak *et al.*, "Machine learning to classify animal species in camera-trap images," *Methods in Ecology and Evolution*, 2019.
- [12] I. H. Sarker, "Deep learning: A comprehensive overview on techniques, taxonomy, applications and research directions," *SN Computer Science*, vol. 2, no. 6, Art. no. 420, 2021.
- [13] K. Y. Chan *et al.*, "Deep neural networks in the cloud: Review, applications, challenges and research directions," *Neurocomputing*, vol. 545, Art. no. 126327, 2023.
- [14] Ultralytics, "Ultralytics YOLO Documentation." [Online]. Available: <https://docs.ultralytics.com/>
- [15] "A framework for detecting and tracking elephants in drone videos," 2025, doi: 10.1139/dsa-2025-0032.