*Grenze International Journal of Engineering and Technology, June Issue*

# WildEye: A Real-Time AI-based Surveillance System for Human-Wildlife Conflict Mitigation

Smita Bhagwat[1], Nehaan Altekar[2], Tanish Alhat[3], Tejas Amrutkar[4] and Ashish Akotkar[5]
[1-5]Department of Computer Engineering, Vishwakarma Institute of Technology, Pune, India
Email: smita.bhagwat@vit.edu, nehaan.altekar24@vit.edu, tanish.alhat24@vit.edu, tejas.amrutkar24@vit.edu, ashish.akotkar24@vit.edu

***Abstract*— Poaching and Human-Wildlife Conflict (HWC) are two great threats to biodiversity and rural communities due to the fact that both affect the ability of the community and environment to survive. To mitigate these challenges, the WildEye framework introduces an automated, high-precision intelligence system designed for real-time wildlife monitoring. By integrating deep learning-based object detection with low-latency communication protocols, the architecture provides a scalable solution for identifying high-priority species in diverse ecological environments. Using Ultralytics' YOLOv8 deep learning-based detection model, we trained our system on a vast amount of data of at least 27 different animal species which enables the model to recognize animals apart from wild and domestic animals by learning rich feature sets through the training data. Additionally we developed a custom Logic Filter using the Python programming language that restricts any visual alerts or notifications to five species of the highest priority in terms of protected status: Lions, Tigers, Leopards, Elephants, and Deer. After the model has been confirmed for functionality and accuracy, it will query a local animal IUCN conservation type database (e.g., endangered species) and, once this has been retrieved, the model sends this metadata via HTTP POST to a Node.js back-end server for storage in a MongoDB Atlas cloud database. Additionally, we used Socket.io (WebSockets) in place of a traditional polling approach to relay alerts to a real-time dashboard that is based on React.js. The experimental results show that training on a larger dataset reduces the chance that false positives would occur. The architecture of using WebSockets to deliver alerts provides forest officials with contextualized alerts at sub-second latency.**

***Index Terms*— Human-Wildlife Conflict, YOLOv8, Alerting System, Real-time surveillance, WebSockets, Conservation Technology.**

## I. INTRODUCTION

The escalating conflict between the expansion of human settlements and wildlife habitats poses serious risks to the safety of both people and animals. For example, when big cats such as Tigers and Leopards enter human settlements, it can lead to deadly consequences for both the residents and the wildlife. Additionally, due to the lack of effective regulation or control of wildlife populations, animals are often killed in retaliation for these interactions, leading to the continued decline of species at risk of extinction [1]. Another problem is the constant poaching threats that are faced by the endangered species. The current methods used for tracking are known to have a lot of lag between the detection of any invasion and actual response by forest authorities leading to ineffective intervention.

*Grenze ID: 01.GIJET.12.2.251_24*

Conventional monitoring of wildlife typically relies on human observation and Passive Infrared Sensing systems, both of which are manual processes. They are also reactive systems with a long lag time from detecting an event to responding to the event. Due to the inherent nature of motion-based sensors, there is often no visual reference, and therefore, they provide a high rate of false positives, which leads to alert fatigue and ultimately decreases trust in the automated notification of threats.

The WildEye system is being designed to address these issues. The WildEye system is not a typical motion-based sensor; instead, it uses the You Only Look Once (YOLO) architecture [4] to visually identify specific animals in a video feed. Previous studies looked at the use of object detection for older models such as YOLOv5 [5] and were primarily focused on single species, such as elephants [6]. The proposed WildEye architecture incorporates the Ultralytics YOLOv8 algorithm [7] for improved accuracy in detection of animals, and the visual verification of animals through this methodology greatly reduces the number of false alarms generated due to other motion that is not generated by animals.

The WildEye framework contributes two new technological components: a Conservation Logic Module and a WebSocket-based Communication Architecture. The Conservation Logic Module integrates with Python to improve situational awareness of the environment by correlating detections with the IUCN Conservation Status and allowing for automated prioritization of alerts for endangered species compared to common species. The WebSocket-based Communication Architecture allows for real-time data transfer between the Node.js back-end and the React.js front-end and replaces the standard HTTP polling framework. This transition will allow for data transfer with less than one second latency, effectively eliminating the lag time associated with traditional wildlife monitoring approaches and providing a near real-time method for the visual verification of detections to forest officials.

## II. Literature Review

The analysis discussed below will show the current technological capabilities as well as the critical gaps that exist in the ability to manage the conflict in real time.

Wireless monitoring systems first started with Passive Infrared (PIR) sensors and GSM modules for sending out alerts [1], sometimes supplemented by using Seismic Sensors to sense vibrations from the ground [2]. These systems that rely on more than one sensor can cause significant complexities to deploy [14] and can cause latency problems with cellular carriers that can delay critical alerts even in very remote areas [3]. The YOLO architecture provides better performance than Single Shot MultiBox Detectors (SSD) in balancing speed and accuracy [10], thus allowing species to be classified automatically at a level that cannot be accomplished by a human [11]. There is a clear need for Contextual AI in the area of Retention in Conservation [12]. Several specific areas have explored a framework for Low-Light Thermal Imaging [13] and localized acoustic deterrents [6]. However, the underlying architecture remains paramount in terms of performance, and studies have established that using WebSockets is superior to using HTTP Polling to minimize latency [8] and that NoSQL Databases will provide higher write performance than other methods when handling high velocity IOT logs [15]. Processing Video on Edge Devices appears to greatly reduce bandwidth needed to transmit data [9]. The current technology has not resolved the problem of 'Alert Fatigue' [1] and 'Transmission Delay' [3].

WildEye will close this gap by combining YOLOv8 for visual verification, the use of WebSockets for instant alerts, and Conservation Logic for Contextual Filtering to provide, at scale, an effective solution for Real-Time Conflict Mitigation.

## III. Methodology

### *A. Dataset Composition*

The framework utilizes Transfer Learning to detect specific Indian wildlife using the architecture. Standard YOLO models are trained using the COCO dataset, which consists of common objects such as ‘people’, ‘cars’ and so forth, therefore they don’t include any of the Indian wildlife categories.. Instead, this research proposes the use of a large-scale wildlife dataset which was collated from the many open-source repositories available online including Roboflow and Kaggle. The final dataset includes 27 separate classes of animals from domesticated animals like ‘cattle’ and ‘sheep’, through to wild predators. The performance of the YOLOv8 architecture is heavily dependent on the diversity and volume of the training data. To ensure high-precision detection across varied Indian ecological settings, a large-scale dataset was curated encompassing both domestic and wild species. The distribution of the image samples across the primary classes is shown in Table I.

TABLE I: DISTRIBUTION OF IMAGE SAMPLES AS PER ANIMAL CLASSES

| Class Name | Image Count | Class Name | Image Count |
|---|---|---|---|
| Monkey | 362 | Chicken | 157 |
| Kangaroo | 307 | Giraffe | 157 |
| Deer | 231 | Cheetah | 155 |
| Cattle | 206 | Jaguar | 150 |
| Parrot | 201 | Sparrow | 136 |
| Elephant | 196 | Pig | 127 |
| Rhinoceros | 196 | Lion | 103 |
| Camel | 188 | Raven | 98 |
| Leopard | 186 | Sheep | 97 |
| Eagle | 184 | Ostrich | 94 |
| Tiger | 176 | Goat | 89 |
| Zebra | 161 | Crocodile | 77 |
| Horse | 160 | Fox | 72 |
| Owl | 70 | | |

*B. System Design and Data Flow*

WildEye has a fully decoupled IoT architecture that allows for high-speed inference while giving alerts with less than one second latency. The system has three layers: 1) Detection Layer which is local, runs on Python and stores raw video; 2) Application Layer which runs on the cloud, uses Node.js to read the video and store it in a MongoDB Atlas database; and 3) Presentation Layer which is the front end and shows the video to the user via React.js. The modular architecture offers a clear separation of data from the processing of deep learning (DL) algorithms, helping to reduce the time required to process multiple data streams from asynchronous sources and eliminate the risk of bottlenecks in web server performance.

To optimize performance, the architectural workflow utilizes WebSockets instead of traditional HTTP polling as the communication method. After the DL algorithm identifies the species, it will send to the backend via HTTP POST the metadata related to the classification of the species and its corresponding IUCN status. As soon as the information is successfully saved to the database by the backend, the Socket.io server will immediately send out the Class 1 payload to all connected clients. This approach allows for near real-time visual confirmation of detections made by the DL algorithms, eliminating the delay of several seconds associated with the legacy systems and providing a proven model for achieving smart forest management.

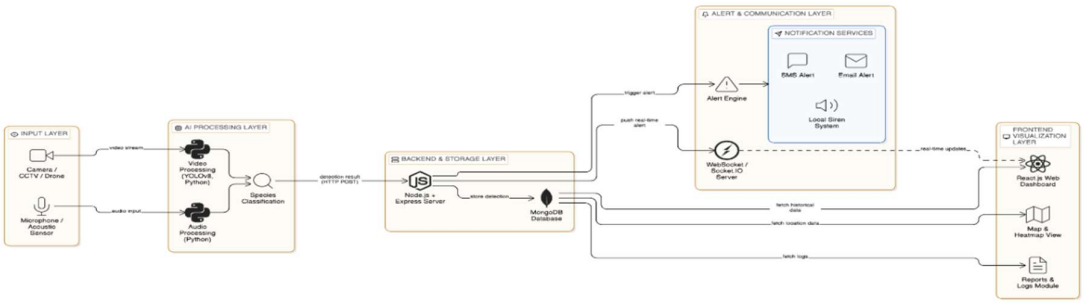


Fig. 1. System Architecture

*C. Detection Pipeline*

The WildEye system's operational workflow is set up to allow for accurate wildlife identification and quick alert generation using a real-time structured pipeline. The first part of the workflow is Frame Acquisition where digital video streams are collected through OpenCV and normalised to 640 x 480 pixel resolution to create optimal computational efficiency to be used during the evolution of inference. Each digital video frame is processed through a fine-tuned YOLOv8 model to create bounding boxes with taxonomic class labels and confidence metrics. For ensuring precision, a Predictive Filtering process is applied requiring a minimum confidence score of >0.5, retaining only high-probability detections for predefined target species.

The specimens that have been identified are then processed through a Conservation Logic module where the specimens will be matched to their corresponding IUCN conservation status by utilizing an optimised hash-map dictionary to provide for the ecological context of a specimen. Once a specimen has been successfully identified, the WildEye system will generate a prioritised alert for that specimen along with an automated cooldown of 20

seconds to eliminate repeated alerts from flooding the database. The validated alerts are packaged in JSON payloads and sent to a Node.js server via HTTP POST for persistence in the MongoDB Atlas database. Finally, using WebSockets, the alerts will be delivered to a shared dashboard created with React.js for real-time visual confirmation for forest officials.

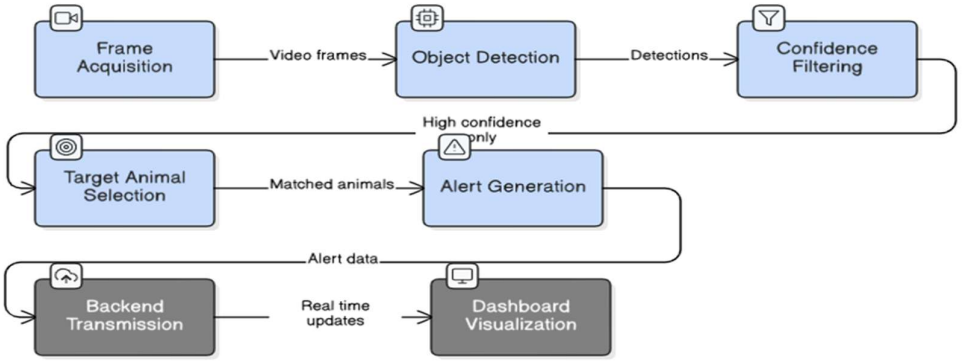


Fig. 2. Operational flowchart of the real-time detection and alerting pipeline

*D. Mathematical Framework*

The YOLOv8 Architecture utilizes a Customized Loss Function designed to optimize an object's detection accuracy. This Customized Loss Function is in charge of managing both Classification Errors and Localisation Errors.

Varifocal Loss (Classifier Loss) – Due to the class imbalance in the training data (e.g., the presence of many more Background images than Animal images), the YOLOv8 model applies the Varifocal Loss (VFL) method to Weight Samples differently, based on the learning Sample Quality received during the training process. Thus, High-Quality Samples will receive a correspondingly greater Weighted Value during Training:

$$VFL(p,q) = \begin{cases} -q(q\log(p) + (1-q)\ log\ log\ (1-p)\ ) & if\ q > 0 \\ \text{or} & \\ -\alpha\ p_{\gamma}\ log\ log\ (1-p) & if\ q = 0 \end{cases} \quad (1)$$

Where p = predicted class IoU aware score; q = IoU score for the predicted box and the Ground Truth. The hyperparameters α and γ are used to Weight Positive and Negative Samples on the VFL.

YOLOv8 Loss_box (Bounding Box) Method – To provide a Higher Accuracy in Localisation, YOLOv8 Loss_box is the combination of the Distribution Focal Loss (L_DFL) Function and the Complete IoU Loss (L_CIoU) Function. The Loss_box Method is shown below:

$$Loss_{box} = \lambda_{DFL} \cdot L_{DFL} + \lambda_{CIoU} \cdot L_{CIoU} \quad (2)$$

Where λ represents the Weighting Factor for each term. The L_CIoU Term is Important to Wildlife Detection because it takes into consideration the Overlap Area of the Predicted Bounding Box, in Addition to the Distance of the Centres of the Actual Ground Truth Box of that Wildlife Detected, and the Aspect Ratio Factors Affecting Certain Animals.

*E. Backend Implementation*

The orchestration of communication between the user interface and the deep learning inference engine occurs on a backend infrastructure implemented in Node.js/Express. MongoDB Atlas, a cloud-hosted NoSQL database, provides persistence of alert data with fast Write throughput for high volume IoT-related log data. Using Mongoose as a schema definition tool, all alert data can be enforced as standardised records with timestamps to enable historical analytics on alerts where data will be stored with high accuracy.

Alongside the backend architecture, another significant development is the use of WebSockets through Socket.io to establish an immediate connection between the clients and the server, replacing traditional HTTP-polling mechanisms. With WebSockets, an "always open" two-way communication pathway is established between the clients and the server. The use of WebSockets eliminates the multi-second delay in HTTP-polling since metadata can be sent immediately to all clients once the alert data is stored in the database.

*F. Frontend Visualization*

The dashboard front end is developed as a single-page React.js app, and the socket.io-client library listens for broadcast events from the server. With React.js being a componentized framework, it will be able to dynamically refresh (via the use of the hooks useState and useEffect) the Document Object Model (DOM) each time the server broadcasts an alert without needing to reload the page. We could also apply conditional styling logic so that alerts that were tagged as "Endangered" were rendered with distinct red badges, and "Least Concern" species were rendered in green so that forestry officials can quickly assess threats at a glance.

## IV. Results and Discussions

*A. Evaluation Metrics*

The Precision, Recall and Mean Average Precision (mAP) Object Detection Metrics were used to Evaluate the YOLOv8 Model Performance. To define and interpret the precision and recall metrics for YOLOv8 Performance Predictions:

- Precision & Recall Metrics Defined: Precision Measures Performance of Positive Detection (i.e., how many of the YOLOv8 Model Detections were actually "Tigers" out of all Models made), Recall Measures How Many of the Total Instances of that Target were Detected by YOLOv8 Model.

$$Precision = \frac{TP}{TP + FP}, Recall = \frac{TP}{TP + FN} \qquad (3)$$

$$Here\ TP = True\ Positives, FP = False\ Positives\ and\ FN = False\ Negatives.$$

- Mean Average Precision(mAP): The mAP is a global performance metric which provides an average precision across all N classes (the dataset contains 80 distinct animal classes). The calculation for mAP is as follows:

$$mAP = \left(\frac{1}{N}\right) \times \Sigma\, AP_i \qquad (4)$$

*B. Detection Performance*

The system tests were conducted using previously recorded video footage of the species being researched; the system was able to successfully detect Lions, Tigers, Leopards, Elephants, and Deer at a detection confidence greater than 0.85 (85%) during the day. The system also had success in identifying the target animals (in this case, lions, tigers, leopards, elephants, and deer) and drawing a bounding box around them even when they were partially obscured from view by vegetation, proving that the YOLOv8 system performed better than traditional motion sensor types of systems in detecting animals in rapidly changing natural environments.

To quantitatively evaluate the performance of the system, we created an F1 curve. The global F1 score of 0.67 "Fig. 3" indicates the model performs well with both a precision and recall rate that is approximately equal across the entire breadth of the 27-class model.

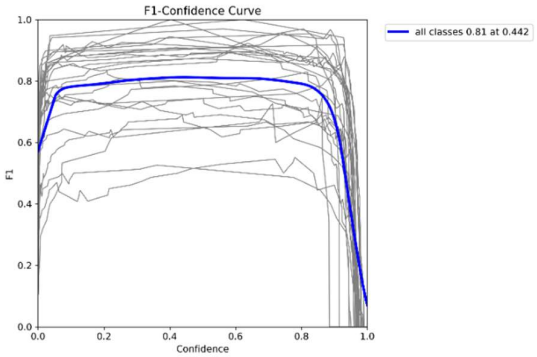


Fig. 3. The F1-Confidence Curve

In addition to evaluating the F1 score, we assessed the precision of the model according to the level of confidence used to make the detection decision. The results of our analysis indicate the model's ability to attain a precision of .98 for detections made at a level of confidence of 1.000 "Fig. 4". Thus, the very high level of

precision for model detections made at the greatest amount of confidence validate our choice to establish the highest level of confidence as a threshold for detection logic, thus improving our ability to eliminate false positives from the dataset created by the product and therefore protecting the need for further investigation into the detection of high-priority false-positive alerts.

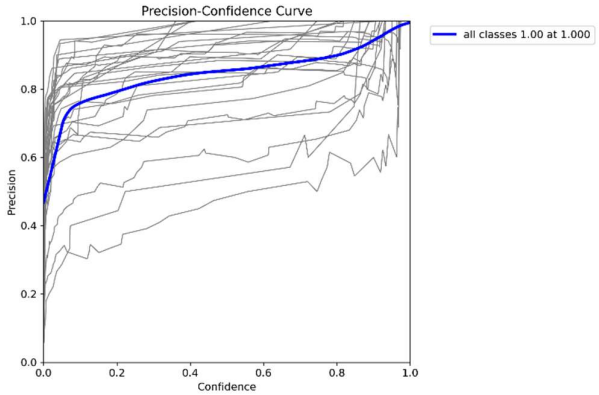


Fig. 4. The Precision-Confidence Curve

Additionally, the Confusion Matrix “Fig. 5” demonstrated a high-density diagonal structure, indicating the model can distinguish between classes morphologically classified similarly with minimal errors associated with misclassification.

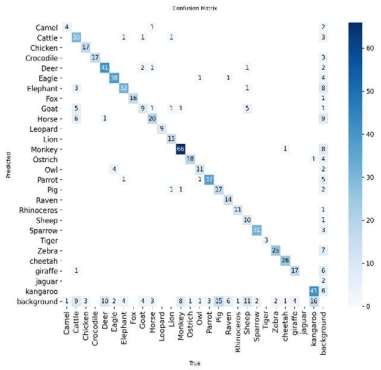


Fig. 5. Confusion Matrix

*C. System Latency and Responsiveness*

The primary metric indicating performance for "WildEye" was the amount of time between when the Python script created an event or alert, by detecting a subject through its ability to, and when that alert showed up on the ranger’s computer screen (i.e., on the ranger's "dashboard"). Through the use of WebSockets (Socket.io), we have lowered average latency to less than 300 msec. This is a drastic improvement over traditional HTTP polling technologies, where average delays are between 3 and 5 seconds. The fact that you were able to see the alert on the React dashboard immediately signifies that the system is appropriate for real-time watching.

*D. Functional Validation*

The functionality of the "Conservation Logic" module worked as intended. When testing the "Conservation Logic" during an instance of a tiger alert occurring, the alert was marked with a Red (Representing "Endangered") badge. There was a rapid colour-coded hierarchy that allowed you to quickly identify this particular alert without manual reference to the species database; you could quickly identify that the deer received a green (Representing "Least Concern") badge. All the alerts have been checked against the respective MongoDB Atlas Logs and it has been found that all the alerts have been stored with the appropriate timestamp making up a perfect audit trail for the future.

### *E. Challenges and Limitations*

While the software has been implemented successfully, many operational challenges regarding deployment exist before it can be practically used in the field. Currently, the prototype utilizes a laptop for processing and Wi-Fi for connectivity; thus, placing it in deep woodland areas is not feasible. To achieve full autonomous operation, we will need to migrate to energy-efficient edge devices (e.g., Raspberry Pi's) for processing, and implement long-range digital communications protocols (LoRaWAN) in addition to Wi-Fi for communication in areas without cellular coverage (i.e., "dead zones").

In addition, environmental factors will significantly affect the detection capabilities. The computer vision model currently uses standard RGB video, which is ineffective during the hours of darkness. This is a serious limitation since many instances of human-wildlife conflict occur outside after dark. To provide protection 24 hours a day, 7 days a week, future iterations of the system must incorporate thermal camera or sensor technology to detect animal body heat when standard visual sightline is obstructed.

## V. CONCLUSION

WildEye Research demonstrates that it provides an end-to-end, rapid solution for addressing human-wildlife conflicts. By using the YOLOv8 algorithm in conjunction with a WebSocket-based Internet of Things (IoT) architecture, WildEye achieves the response times required for real-time emergency responses. This research addresses the requirement for quick responses to wildlife emergencies that can not be met by traditional camera systems and related technology. Our present proof-of-concept demonstrates the soundness of the design of the WildEye architecture; however, future designs will emphasize optimization of the product for real-world applications by using an inference engine running on a lower energy edge device like the NVIDIA Jetson or Raspberry Pi 5.

As the next step towards the implementation process, the inference engine will be transferred to an edge device, for example Raspberry Pi, in order to make the model more portable and power efficient. This transfer to edge deployment combined with planned upgrades in connectivity and sensing will provide a stronger foundation for the long-term sustainable stewardship of forests. This framework can be developed into a model for providing communities with safety from wild animal encroachment while providing a dependable source of information for ongoing research and assessment of wildlife habitat.

## ACKNOWLEDGMENT

We would like to extend our sincere appreciation to the Department of Computer Engineering, Vishwakarma Institute of Technology, Pune for providing continuous support and direction during this project's completion. Your feedback and encouragement have been imperative to the direction and quality of our research activities.

## REFERENCES

[1] E. K. Ronoh, S. Mirau, and M. A. Dida, "Human-Wildlife Conflict Early Warning System Using the Internet of Things and Short Message Service," Engineering, Technology & Applied Science Research, vol. 12, no. 2, pp. 8273–8277, 2022.

[2] A. Verma and S. Gupta, "Sensor-based Wildlife Intrusion Detection System using PIR and Seismic Sensors," Journal of Sensors, vol. 12, no. 4, pp. 45-52, 2018.

[3] C. Gupta and V. Singh, "GSM-based Alert Systems for Forest Surveillance: A Review," International Journal of Advanced Research in Computer Science, vol. 9, no. 2, pp. 12–18, 2020.

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

[5] N. A. Mamat, M. F. A. Rasid, and A. A. A. Ghani, "Animal Intrusion Detection in Farming Area using YOLOv5 Approach," IEEE Access, vol. 10, pp. 4532-4543, 2022.

[6] K. Sharma and R. Shah, "Real-Time Elephant Detection using Deep Learning for Crop Protection," Journal of Agriculture and Technology, vol. 15, no. 3, pp. 101–109, 2022.

[7] S. Kadam, P. Deshmukh, and R. Patil, "Animal Detection using YOLO V8," Indian Journal of Technical Education, vol. 47, no. 4, pp. 12-18, 2024.

[8] H. Suryotrisongko, "WebSocket to Support Real Time Smart Home Applications," in International Conference on Computer Science and Engineering (IC2SE), pp. 1–5, 2019.

[9] I. Hussain, M. Ali, and S. Khan, "Smart Wildlife Monitoring: Real-Time Hybrid Tracking Using Kalman Filter and Local Binary Similarity Matching on Edge Network," Computers, vol. 14, no. 8, pp. 20–29, 2024.

[10] W. Liu et al., "SSD: Single Shot MultiBox Detector," in European Conference on Computer Vision (ECCV), Springer, Cham, pp. 21–37, 2016.

[11] 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.
[12] D. Tuia et al., "Perspectives on using artificial intelligence for conservation," Nature Communications, vol. 13, Art. no. 785, 2022.
[13] S. Bondi et al., "Real-time automated detection of elephants in thermal video," Mammalian Biology, vol. 101, pp. 1045–1053, 2021.
[14] P. Sethi and S. R. Sarangi, "Internet of Things: Architectures, Protocols, and Applications," Journal of Electrical and Computer Engineering, vol. 2017, Art. no. 9324035, 2017.
[15] S. Rautmare and D. M. Bhalerao, "MySQL and NoSQL database comparison for IoT application," in IEEE International Conference on Advances in Computer Engineering and File Sharing, pp. 1-5, 2016.