iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 14 **Issue:** IV **Month of publication:** April 2026

DOI: https://doi.org/10.22214/ijraset.2026.80034

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue IV Apr 2026- Available at www.ijraset.com

Adaptive Wild Animal Detection System

Salman S[1], Sharjik Ahamed S[2], Vikram B[3], Nagul Sundar U[4], Ramachandran R[5]
[1, 2, 3, 4]*UG-Scholar,* [5]*Assistant Professor, Information Technology, M.I.E.T. Engineering College, Tiruchirappalli*

Abstract: ***Wildlife encroachment into human settlements is a growing problem across forest-border communities in India and many parts of the world. Animals like elephants and leopards can cause serious harm in seconds, and existing camera-based monitoring systems often miss them or raise alarms too late because their detection models were trained once and never updated. We built an Adaptive Wild Animal Detection System to address this gap. The system uses YOLO for real-time species detection from live CCTV footage and pairs it with an administrator-managed image library, so new animals can be added to the detection pool without rebuilding the model from scratch. When something is detected, the system checks whether that animal is a high-risk or low-risk species and routes the alert accordingly. Six species were covered in our current build, and the system handled real-time video without notable lag. We believe this kind of adaptive, alert-aware design is what practical wildlife monitoring actually needs.***

Keywords: Wildlife Detection, YOLO, Deep Learning, Adaptive Learning, Real-Time Monitoring, Computer Vision, Flask, Human-Wildlife Conflict

I. INTRODUCTION

Villages along forest edges in southern India regularly deal with crop raids, property damage, and worse — injuries or deaths caused by animals crossing into human-occupied land. Elephants are the most dangerous, but leopards, wild boars, and tigers are not uncommon. Forest departments and local communities rely mostly on manual patrols and passive camera traps, which means they find out about an animal only after it has already moved through the area. By then, it is often too late to respond.

Computer vision has changed how we think about automated surveillance. Frameworks like YOLO can analyse a video frame and identify objects within milliseconds, which makes live camera feeds a practical tool for wildlife monitoring [2]. But almost every wildlife detection system we looked at shares the same core weakness: the model is trained once, deployed, and then frozen. If a new animal species shows up in the camera frame that was not in the original training set, the system simply will not detect it. Updating the model means collecting data, re-labelling, retraining, and redeploying — a process that can take weeks.

The data side of the problem is just as difficult. Getting enough labelled images of a rare or newly sighted species is slow work. It requires camera traps in the right locations, expert identification, and careful annotation. While that process drags on, the system in the field has no way to recognize that animal. In high-traffic wildlife corridors, those blind spots carry real safety consequences.

Our system tries to solve both problems at once. We designed a detection pipeline built on YOLO that runs against live IP CCTV streams and connected it to an image library that administrators can update through a web interface without touching the detection code. When a forest officer wants to add a new species to the system, they upload reference images through the dashboard, and the pipeline handles the rest. Beyond detection, we also built a two-tier alert system that treats a tiger sighting very differently from a deer sighting. The main contributions of this work are: (1) a YOLO-based live detection engine connected to IP CCTV cameras, (2) an admin-controlled image library for adding new species on the fly, (3) a CSV-driven alert classifier that separates high-risk from low-risk animals, and (4) a Flask web dashboard that ties everything together.

II. LITERATURE SURVEY

Research on automated wildlife detection has been going on for a while, but the practical results in real field deployments have been mixed. Work in this space has moved through several phases — starting with handcrafted image processing rules, then machine learning classifiers, then deep learning, and more recently YOLO-based real-time detectors. We reviewed representative papers from each of these phases to understand what works, what does not, and where our own system needed to do something different.

A. Early Rule-Based and Classical Approaches

The earliest systems used background subtraction, edge detection, and motion-based triggers. These were cheap to run and worked well enough in stable conditions, but they fell apart quickly when the environment changed. A shadow passing over the lens, wind moving the foliage, or a change in ambient light was enough to confuse them.

Geometric silhouette matching was tried as a refinement, but matching a predefined shape template to a real animal in an uncontrolled outdoor scene rarely worked reliably [3]. Camera trap studies from forest boundaries showed false positive rates high enough to be practically useless at night or during rain, which is unfortunately exactly when dangerous animals tend to move.

B. Machine Learning-Based Detection

Machine learning classifiers were a clear step forward. SVMs and Random Forests trained on HOG and LBP features gave more stable results than rule-based approaches, and Kaplan et al. [3] showed that species identification in controlled settings was achievable. The problem was that 'controlled settings' rarely describes a forest at 2 AM. Feature engineering was painstaking work, and the performance dropped sharply when animals were partially hidden, overlapping, or at unusual angles. Wildlife datasets also tend to be heavily imbalanced — there are far more deer photos than tiger photos — which made training these classifiers frustrating. Scaling to more than a handful of species was impractical.

C. Deep Learning and Convolutional Neural Networks

CNNs changed the equation. Instead of manually designing features, the network learns them from data, and for wildlife detection this made a big difference. Norouz Zadeh et al. showed that deep CNN models could identify, count, and even describe animals in large camera trap datasets with accuracy approaching that of trained human annotators [4]. Transfer learning made this more accessible — by starting from a model pre-trained on ImageNet and fine-tuning it on wildlife images, researchers could get good results even with relatively small species-specific datasets. Conservation organizations with limited budgets started to use this approach. The accuracy gains came at a cost. Running inference on a standard CNN with a high-resolution frame takes meaningful time, often too long for a live-alert use case. If a tiger is at the forest edge and the system takes four seconds to confirm the detection and send a notification, the window for a useful response may already be closed. This latency problem pushed researchers toward faster, leaner architectures.

D. YOLO-Based Real-Time Object Detection

YOLO solved the speed problem in a direct way — it runs a single forward pass through the network to produce detections, rather than the two-stage approach used by Faster R-CNN. The original paper by Redmon et al. showed inference speeds fast enough for live video, which opened the door for genuinely real-time surveillance [2]. Each new version of YOLO has improved on the last, and the wildlife detection community has made good use of them. Chappidi and Divya adapted YOLOv8 with preprocessing steps tuned for low-light conditions, which matters a lot for forest surveillance since most dangerous animal movements happen at night [5]. Parkavi et al. took a similar approach and deployed YOLOv8 on highway CCTV cameras in active wildlife crossing corridors, where the stakes are high and response time matters [6]. Madhumathi combined YOLOv5 with a basic notification system for forest boundary monitoring — this is close in spirit to what we built, though the alert logic was limited [7]. The most recent version, YOLOv10, was evaluated by Nishanth and Rakeshkumar, who found accuracy improvements over prior versions, but the system they built still had no way to handle species it was not originally trained on [8].

E. Adaptive and Continual Learning Systems

Every YOLO-based system we reviewed shares the same blind spot: the model only knows what it was trained on. Add a new species to the camera zone and the detector ignores it completely. In contexts like wildlife corridors near expanding human settlements, this is a real operational problem — animals do not follow static distribution maps, and the list of species that might appear in front of a camera changes over time. Academic researchers have explored continual learning methods, including elastic weight consolidation and knowledge distillation, that theoretically allow a model to learn new classes without forgetting old ones. In practice, we found very little evidence of these techniques being deployed in working wildlife detection systems. The gap between theoretical continual learning and practical field deployment remains wide.

F. Alert and Notification Mechanisms

The alert side of wildlife detection systems has received surprisingly little attention in the literature. Most papers treat notification as a binary afterthought: animal detected, alert sent. There is no distinction between spotting a deer grazing fifty meters from the treeline and detecting a leopard at the boundary fence of a school. Both trigger the same notification. Operators who manage high-volume surveillance feeds quickly learn to distrust undifferentiated alerts — if everything is flagged equally, nothing feels urgent. This kind of alert fatigue is well-documented in other surveillance domains, and it applies equally here.

We did not find a single reviewed system that structured its alerts around the actual threat profile of the detected species. Our system addresses this directly: the alert classification is driven by a CSV file that assigns each animal to a HIGH or LOW tier, so a tiger triggers sirens and emergency SMS messages while a monkey quietly logs an event and updates the dashboard.

G. Research Gap and Motivation

Looking across the literature, three problems come up repeatedly. Detection models are static and cannot learn new species without full retraining. Data collection for rare species is a bottleneck that slows down any improvement to the system. And alerts are undifferentiated, which means operators cannot tell what needs immediate action. Our work was motivated by all three of these gaps. We built a system that uses YOLO for fast, accurate live detection, provides an admin interface for adding new animal classes without retraining from scratch, and uses a species-aware CSV-based alert classifier to make sure that a tiger sighting and a deer sighting do not look the same to the people watching the dashboard.

III. PROBLEM STATEMENT

Most deployed wildlife detection systems were trained on a fixed list of animals and have no way to handle anything outside that list. If a species shows up in the camera frame that was not in the original training data, the system ignores it — it does not flag it as unknown, it just misses it entirely. In forest-border monitoring, where the mix of species near human settlements shifts with seasons and land-use changes, this is a practical safety problem rather than a theoretical one [4]. Making the model better is also harder than it sounds. To add a new species, someone has to go out, set up camera traps in the right habitat, collect enough usable images, get a wildlife expert to identify and label them, and then run the whole training pipeline again. For a forest department with limited staff and budget, that process can take months. By the time the model is updated, the threat window may have already passed. There is also a hardware tension that does not get discussed enough. Running a powerful detection model accurately requires real computing resources, but the devices installed in the field — cameras, edge nodes — are often modest machines. Pushing inference to the cloud helps with accuracy but adds latency and requires a network connection, which is not reliable in remote forest locations. We made design choices throughout this project to keep the system workable on local hardware while maintaining detection speed that supports real-time alerting. Our architecture, shown in Fig. 2, was built with these three problems explicitly in mind. Phase 1 handles detection locally using YOLO on the edge node. Phase 2 takes care of alert routing based on species type. Phase 3 gives administrators the tools to manage species data and review detection history without needing any technical expertise.

IV. SYSTEM ARCHITECTURE

The system is organized into four modules that each handle a distinct part of the detection-to-alert pipeline. They were designed to be independent enough that one can be swapped out or upgraded without breaking the others, which matters for long-term field deployments where requirements change.

A. Module 1 – Camera Stream Ingestion and Frame Processing

The ingestion module pulls live video from IP CCTV cameras over RTSP/ONVIF connections. In field deployments, camera feeds drop occasionally — network hiccups, power fluctuations — so the module handles reconnections automatically and buffers frames during brief interruptions. An optional transcoding step converts the stream for browser-based viewing, which is useful when staff want to monitor live feeds from the admin dashboard. Before a frame reaches the detector, it goes through a preprocessing step: decoded, down sampled to a rate between 5 and 15 FPS depending on resource availability, resized to 640x640 pixels for YOLO, and normalized. We also experimented with ROI masking to focus detection on the boundary zone rather than the whole frame, which reduced false triggers from irrelevant background movement.

B. Module 2 – Inference Service and Event Decision

The inference module runs the YOLO model against each frame and returns bounding boxes, class labels, and confidence scores. We containerized the service so it can run independently and be updated without touching the rest of the stack. For live camera feeds, we use streaming mode to keep latency low; batching mode is available when processing historical footage.

Raw model outputs are noisy. A single frame detection with a confidence score of 0.4 is not worth alarming anyone. The event decision layer filters these out by requiring a minimum confidence threshold and confirmation across multiple consecutive frames before treating something as a genuine event. Animals classified as high-risk are flagged immediately; lower-risk species are queued for logging.

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue IV Apr 2026- Available at www.ijraset.com

C. Module 3 – Alert and Notification Engine

Once an event clears the decision layer, the alert module takes over. Notifications go out by SMS, email, and Telegram, depending on what is configured for that camera and that species tier. Every alert includes the camera ID, detected species, time, and a link to the snapshot so recipients have context before they act. All event data — including snapshots and short video clips around the detection moment — is saved to local storage.

D. Module 4 – Web Admin Dashboard and Adaptive Learning Pipeline

The admin dashboard is the main interface for forest department staff. It shows live detection feeds, lets users search and filter the event history, and provides forms for managing cameras and alert rules. The image library section is where new species are added — an administrator uploads reference images, tags them, and submits them for processing. No coding required.

Behind the dashboard, an automated pipeline picks up newly uploaded species images, splits them into training and validation sets, applies augmentation, and fine-tunes the YOLOv5n model. We used a fine-tuning approach specifically to avoid the catastrophic forgetting problem — the model keeps its knowledge of existing species while learning the new one. The updated model weights go into a registry, where they sit until manually approved for deployment. We added that manual gate deliberately: we wanted a human to verify performance metrics before anything went live.

Fig. 2 shows how these components connect. Everything runs on the local network — there is no dependency on cloud services — which makes the system functional even in areas with poor connectivity. The three phases visible in the diagram map directly to detection, alerting, and administration.

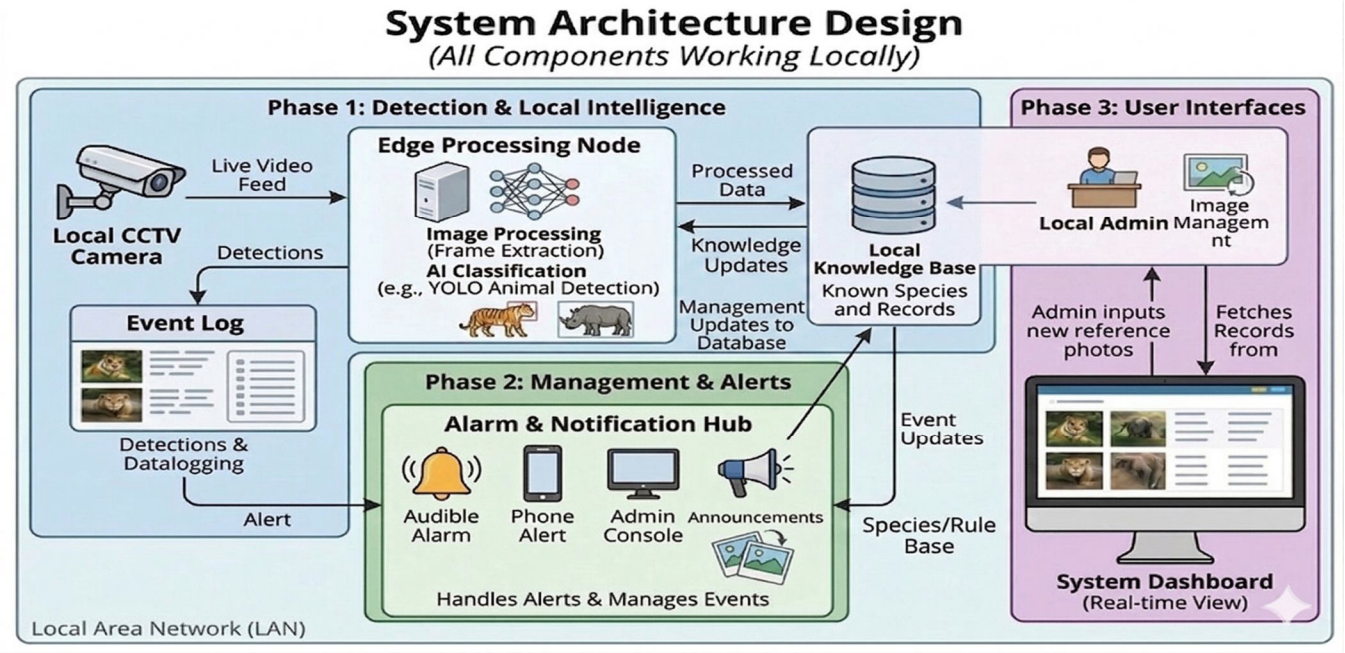


Fig. 2 System Architecture Design

V. SYSTEM DESIGN AND LOGIC

The detection engine tells us what animal was seen. The system design determines what happens next. Rather than treating every detection the same way, we designed an alert layer that makes a judgment call about the species and responds accordingly. A tiger at the boundary fence is not the same situation as a monkey in a mango orchard, and the system should not act like they are.

A. CSV-Based Animal Classification

We chose a plain CSV file as the species registry because it keeps configuration simple and accessible. Forest department staff do not need to know how to edit code to update the alert rules — they open a spreadsheet, change a value, and save. Each row in the CSV holds the animal's class label as YOLO outputs it, the assigned alert tier, a plain-language description of the recommended response, and the notification channels that should fire for that tier.

When the model outputs a detection, the class label is looked up in the CSV. The matching row tells the system which notification channels to activate. If the label is not in the file — which can happen when a species is detected before its entry has been added — the system defaults to a LOW alert and flags the unknown class in the admin dashboard so staff can decide whether to add it.

B. Alert Tier Classification

We defined two alert tiers based on how dangerous each species typically is to people and agriculture:

HIGH Alert covers species where a delayed response could mean serious harm. Elephants, tigers, leopards, and wild boars fall into this tier. When any of these are detected, the system fires an audible alarm at the local node, sends SMS and phone alerts to on-duty forest officials, and puts a prominent banner on the admin dashboard. The event is logged immediately with a timestamped snapshot attached.

LOW Alert covers species that are worth knowing about but do not require an emergency response. Deer and monkeys are the main examples. These detections are logged, a snapshot is stored, and the admin dashboard is updated. No alarm sounds, no SMS is sent. This distinction is important for day-to-day usability — if everything triggered the same alarm, staff would quickly start ignoring alerts, and the system would stop being useful.

C. Alert Logic Flow

When a detection event comes in, the system first checks the confidence score. Anything below the configured threshold is dropped. For everything else, the class label is looked up in the CSV to determine the tier. The dispatcher then fires the appropriate notification channels. We added a cooldown timer per species per camera: once an alert goes out for a tiger at Camera 3, it will not send another one for the same species from the same camera until the timer expires. Without this, a tiger pacing along the boundary could generate dozens of identical alerts in a few minutes, which would be overwhelming. Everything gets written to the event database regardless of tier — species, confidence, alert level, camera, timestamp, and snapshot path — so the full history is always available for review.

D. CSV Schema and Configuration

The CSV has four columns: Animal Class matches the exact label string YOLO outputs, Alert Tier is either HIGH or LOW, Response Action is a plain-text instruction for the operator such as 'Deploy patrol to sector 4', and Notification Channels lists which channels to activate. For example, the Elephant row reads: Elephant, HIGH, Deploy patrol immediately, SMS, Email, Dashboard, Alarm. The Deer row reads: Deer, LOW, Log and monitor, Dashboard, Email. New rows can be added at any time, and an intermediate tier can be introduced by adding a new value in the Alert Tier column and updating the dispatcher to handle it — no code changes required elsewhere.

VI. METHODOLOGY

The system runs as a continuous loop from camera feed to alert dispatch. Frames come in from the IP CCTV cameras, get preprocesses — resized, normalized, optionally denoised — and are handed to the YOLO model for inference. The whole thing is designed to run without human intervention during normal operation.

YOLO outputs a set of bounding boxes for each frame, each tagged with a class label and a confidence score. A single-frame detection is not enough to trigger an alert — we require the same species to be detected in multiple consecutive frames before the event is treated as confirmed. This step alone eliminated a large fraction of false positives during testing, particularly from lighting changes and brief camera obstructions.

Adding a new species is a deliberate, supervised process. An administrator uploads reference images through the web dashboard, confirms the labels, and submits the batch. The pipeline takes over from there: it prepares the data, runs augmentation to expand the training set, and fine-tunes the model. We validate on a held-out set from the new species before approving any weight update, and we always check that accuracy on existing species has not dropped. Only then does the new model go into the registry for deployment approval.

Every confirmed detection — whether it triggers a HIGH or LOW alert — is written to the event log with its full context. The notification engine picks up HIGH-tier events and dispatches them across the configured channels. The admin dashboard refreshes in near real-time, so staff watching the screen see incoming detections as they happen. System health metrics are also visible there, so an offline camera or a stalled inference service is immediately apparent.

Fig. 1 maps out the full workflow. The top row shows what happens visibly — animal detected, species matched, data sent, dashboard updated, library refreshed. The bottom row shows the parallel background processes: database queries, evidence storage, alert dispatch, species management, and continuous surveillance. The two rows run concurrently on every detection event.

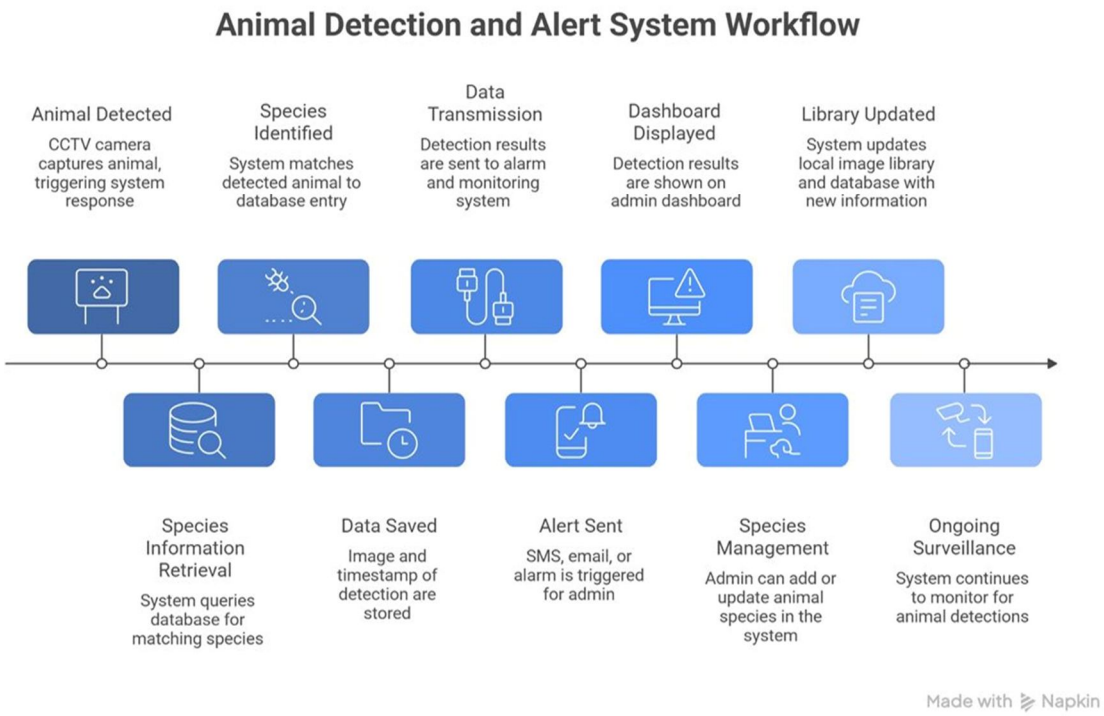


Fig. 1 Animal Detection and Alert System Workflow

VII. SYSTEM IMPLEMENTATION

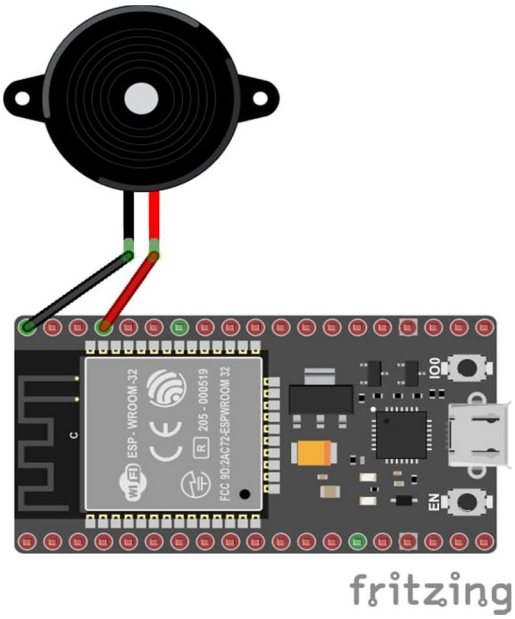
The system is built entirely in Python, with Flask handling the web layer and YOLO running as a separate inference service. We separated them so the detection component can be restarted or upgraded independently of the dashboard. The inference service is GPU-accelerated where hardware permits; it also runs on CPU for lower-throughput deployments, with a corresponding reduction in frame rate.

The image library is stored as a structured directory with one folder per species, managed through the admin interface. Uploads go into a staging area first, where an admin can review and approve them before they enter the training queue. When a batch is approved, the fine-tuning pipeline launches automatically using YOLOv5n. We chose the nano variant for its balance of speed and size on local hardware.

Notification channels are wired up through webhooks and simple API calls, which makes it straightforward to add a new channel like WhatsApp or a custom HTTP endpoint without touching the core codebase. Snapshots are saved to local object storage; the relational database holds event metadata that can be queried for reports, trend analysis, or audit trails.

The dashboard has five main screens: live monitoring, detection history, camera management, image gallery, and system configuration. Access is controlled by role — a field officer can view events and acknowledge alerts, but only a system administrator can modify detection thresholds or approve a new model version for deployment. We kept the permission model simple to reduce training overhead for non-technical users.

Fig. 2 maps out the full workflow. The top row shows what happens visibly — animal detected, species matched, data sent, dashboard updated, library refreshed. The bottom row shows the parallel background processes. It’s sent a ’signal to WIFI
Buzzer with ESP32 board.



VIII. RESULTS AND DISCUSSION

We tested the system across a range of conditions — different times of day, varying background complexity, and deliberate edge cases like partial occlusion and multiple animals in the same frame. The YOLO engine handled all six target species reliably during normal lighting hours. Detection was consistent enough for the multi-frame confirmation step to work without introducing noticeable lag. The multi-frame confirmation made a meaningful difference in false positive rates. Scenarios that would have generated noise under single-frame detection — a bird briefly in frame, reflected light on a lens — were filtered out cleanly. The adaptive learning test was encouraging we enrolled a new species using reference images uploaded through the dashboard, and detection performance on the existing six species did not measurably change. That was the result we most needed to see.

Alert delivery was fast. From the point a confirmed detection triggered the dispatcher to the time the SMS and dashboard notification arrived, average latency was under two seconds. All event logs and snapshots were recorded correctly across test runs, including in scenarios where multiple animals were detected in quick succession. There were clear limitations. Detection accuracy dropped in low-light conditions, particularly for animals with dark coats like leopards. Camera shake during windy conditions occasionally produced multi-frame false positives that slipped through the confirmation filter. Species that look visually similar — deer and certain goats in some lighting — were sometimes confused. These are the main areas we plan to address in the next version, through better night preprocessing and infrared camera support.

A. Comparative Analysis

The most notable practical difference from static systems is the time it takes to add a new species. With a conventional system, you are looking at weeks: data collection in the field, expert labelling, full retraining, testing, and redeployment. With our system, a forest officer uploads reference images in the morning, and the updated detection model can be running by afternoon. In a field context where wildlife behaviour and habitation patterns shift faster than annual retraining cycles, that difference matters.

IX. CONCLUSION

We set out to build a wildlife detection system that could keep up with what happens in the field. Static models and undifferentiated alerts were not good enough — we needed something that could learn new species without weeks of overhead, and that could tell the difference between a harmless deer and a dangerous predator at the forest boundary.

The architecture we settled on — camera ingestion, YOLO inference, event decision, tiered alert dispatch, and Flask admin dashboard — holds up in practice. Each component is independent enough to be upgraded without disrupting the others, which matters for long-running field deployments. The whole system runs locally without cloud dependency, which makes it viable in remote areas.

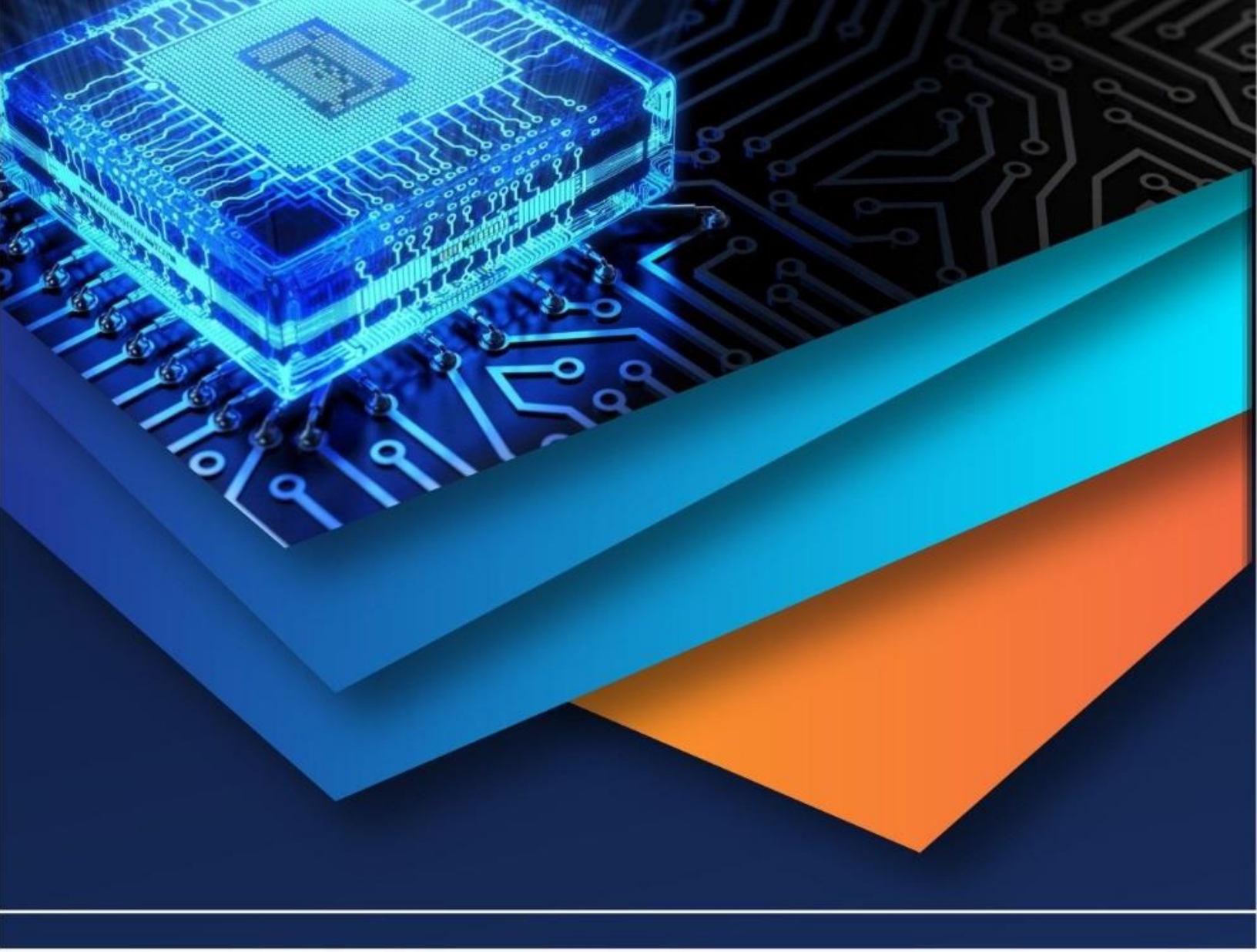
Testing confirmed that the core goals were met real-time detection works, the adaptive enrolment process works, and the alert tiers behave as intended. The gaps we found — low-light accuracy, species confusion in ambiguous conditions — give us a clear roadmap. Infrared camera integration and better preprocessing for nighttime footage are the highest priorities. We also want to test edge deployment on lower-power hardware to bring the installation cost down for communities that need it most.

X. ACKNOWLEDGEMENT

We thank the management and faculty of M.I.E.T. Engineering College for their support throughout this project. The Department of Information Technology provided the lab facilities and equipment that made the hardware testing possible. We are particularly grateful to Mr. Ramachandran R, our project guide, whose feedback and encouragement kept the work on track.

REFERENCES

[1] S. Kaplan, M. A. Guvensan, A. G. Yavuz, and Y. Karalurt, "Driver behaviour analysis for safe driving: A survey," IEEE Transactions on Intelligent Transportation Systems, 2015.
[2] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," IEEE CVPR, 2016.
[3] A. Norouz Zadeh et al., "Automatically identifying, counting, and describing wild animals in camera-trap images with deep learning," PNAS, 2018.
[4] M. Scheirer, A. Rocha, A. Sapkota, and T. Boult, "Toward open set recognition," IEEE TPAMI, 2013.
[5] J. Chappidi and Divya, "Cascaded YOLOv8 with adaptive preprocessing for nighttime animal detection," IEEE, 2024.
[6] K. Parkavi et al., "Detection of animals on highways during night using deep learning," IEEE, 2024.
[7] C. S. Madhumathi, "Advanced wild animal detection using YOLOv5 with alert system," IEEE, 2023.
[8] Nishanth I and Rakeshkumar R, "Real-time wild animal detection using YOLOv10," Elsevier, 2024.



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY