International Journal for Multidisciplinary Research (IJFMR)
E-ISSN: 2582-2160 ● Website: www.ijfmr.com ● Email: editor@ijfmr.com

ApexGuard-AI: A Unified Real-Time Multi-Threat Surveillance System Using YOLOv8/v9 Deep Learning Models

Amey S. Avasthi[1], Piyush V. Patil[2], Divyaraj V. Patil[3], Jayesh V. Sonawane[4], Sarang S. Deore[5]

[1,2,3,4,5]Student, Department of Computer Engineering, S.S.V.P.S.'s B. S. Deore College of Engineering, Dhule, Maharashtra, India · Academic Year 2025–26

ABSTRACT

Traditional CCTV surveillance depends on human operators susceptible to fatigue, delayed response, and limited scalability. This paper presents ApexGuard-AI, a unified real-time multi-threat AI surveillance platform that concurrently detects dangerous wildlife, fire and smoke, and weapons from a single camera feed. The system employs a custom fine-tuned YOLOv8n model for biological threat identification, a dedicated YOLOv9 model for fire and smoke recognition, and a standard YOLOv8n COCO model with label filtering for weapon detection. A FastAPI asynchronous backend manages video streaming and alert dispatch while a daemon detection thread overcomes Python's Global Interpreter Lock (GIL) limitation. Events are persisted through a dual-stage CSV-to-SQLite logging pipeline, and a rolling 40-frame pre-event buffer ensures complete incident capture. Experimental evaluation on standard consumer hardware demonstrates 25 FPS throughput with alert latency below 150 ms, validating the practical feasibility of automated, multi-class AI threat detection in real-world surveillance environments.

Keywords — Real-time surveillance, YOLOv8, YOLOv9, object detection, fire detection, wildlife detection, weapon detection, FastAPI, deep learning, CCTV automation, GIL threading.

I. INTRODUCTION

The proliferation of surveillance cameras in urban, rural, and industrial settings has outpaced the capacity of human operators to monitor them effectively. Empirical studies consistently indicate that an operator's anomaly-detection ability degrades within 20 minutes of continuous screen monitoring [7]. This fundamental limitation of manual surveillance has created an urgent demand for automated, intelligent systems capable of operating without human fatigue.

Existing automated solutions typically address a single threat category, requiring deployment and management of multiple independent systems—one for fire detection, another for intrusion, a third for wildlife. This architectural fragmentation increases infrastructure cost, maintenance overhead, and response complexity. A unified platform performing concurrent multi-class threat detection from a single camera feed represents both a technical and operational advance.

ApexGuard-AI addresses this gap by integrating three detection modules—dangerous wildlife, fire and smoke, and weapons—into a single cohesive surveillance platform. The system leverages the latest generation YOLO detectors for their proven real-time inference capability, backed by a high-performance FastAPI asynchronous backend for scalable alert delivery and structured event logging.

The primary contributions of this work are: (i) a modular multi-threat detection architecture using YOLOv8 and YOLOv9 operating concurrently on a single video stream; (ii) a GIL-aware daemon

threading model that fully decouples AI inference from HTTP request handling; (iii) a dual-stage CSV-to-SQLite event logging pipeline with pre-event video buffering; and (iv) empirical performance results confirming sub-150 ms alert latency at 25 FPS on CPU-only consumer hardware.

II. LITERATURE SURVEY

Table I summarises ten foundational and directly related works that informed the architecture and implementation of ApexGuard-AI. The reviewed literature spans foundational object detection theory, domain-specific detection applications, and system-level implementation patterns.

TABLE I — Literature Survey Summary

No.	Author(s) & Year	Paper / Topic	Technique	Relevance
01	Redmon et al. (2016)	You Only Look Once: Real-Time Object Detection	YOLOv1 CNN	Foundational basis for single-pass detection engine
02	Jocher et al. (2023)	YOLOv8: A New Frontier in Object Detection	YOLOv8 Architecture	Primary model for wildlife, fire & weapon modules
03	Muhammad et al. (2018)	Efficient Deep CNN for Fire Detection in IoT Environments	CNN + IoT Integration	Inspired fire/smoke module; confidence thresholds
04	Warwick et al. (2020)	Automated Wildlife Detection Using Deep Learning	Transfer Learning / ResNet	Basis for multi-class animal identification
05	Grega et al. (2016)	Automated Detection of Firearms in Surveillance Video	HOG + SVM Classifier	Weapon detection motivation; improved with YOLO
06	Ramirez et al. (2022)	FastAPI for High-Performance ML Model Serving	ASGI / Async Python	Backend architecture for non-blocking API serving
07	Sultani et al. (2018)	Real-World Anomaly Detection in Surveillance Videos	MIL Ranking + C3D	Referenced for anomaly-based alert triggering design
08	Nguyen et al. (2019)	Real-Time Video Streaming for AI via OpenCV	OpenCV VideoCapture	Camera input and MJPEG streaming reference
09	Zhao et al. (2021)	Multi-Threaded AI Inference for Embedded Surveillance	Threading + Shared Memory	Informed GIL workaround and background thread design
10	Tripathi et al. (2022)	Intelligent CCTV with Automated Alert & Event Logging	SQLite + Notification APIs	Direct reference for alert system and DB logging

Redmon et al. [1] introduced the single-pass detection paradigm that underpins all subsequent YOLO variants. Jocher et al. [2] advanced this with YOLOv8, achieving state-of-the-art mAP scores. Muhammad et al. [3] demonstrated CNN-based fire detection in IoT environments, directly informing confidence threshold selection for the fire module. Warwick et al. [4] validated transfer learning for wildlife classification, motivating the fine-tuned YOLOv8n bio-threat model. Grega et al. [5] exposed limitations of classical HOG+SVM weapon detection, which YOLO-based approaches overcome through end-to-end feature learning. Zhao et al. [9] specifically addressed multi-threaded AI inference on embedded platforms, directly shaping the GIL workaround design. Tripathi et al. [10] provided the closest prior art, combining SQLite event logging with notification APIs for intelligent CCTV—ApexGuard-AI extends this with simultaneous multi-model detection and pre-event buffering.

III. PROBLEM STATEMENT

Contemporary surveillance deployments face four interconnected limitations that collectively degrade their operational effectiveness:

1. Human Fatigue: Operators monitoring multiple CCTV feeds lose concentration within 20 minutes, creating vulnerability windows in which real threats go undetected.
2. Delayed Response: Traditional CCTV is passive, recording events without automated reaction. By the time an operator detects and reports an incident, the critical intervention window may have elapsed.
3. Inability to Scale: Manual monitoring cognitive load increases linearly with camera count, making large-scale deployments operationally impractical without proportionate staffing.
4. Fragmented Alert Support: Single-purpose detectors lack integrated multi-channel alert dispatch and fail to capture pre-incident context, reducing both response speed and post-incident analysis quality.

ApexGuard-AI directly addresses each limitation through fully automated multi-threat detection, sub-200 ms alert dispatch, a scalable asynchronous backend, and a rolling pre-event frame buffer that preserves incident context.

IV. PROPOSED METHODOLOGY

The proposed system employs a modular, multi-model detection architecture in which each threat category is handled by a dedicated YOLO model instance operating on the same live video stream. Figure 2 illustrates the frame processing pipeline.

A. Multi-Model Detection Architecture

Three YOLO model instances operate within the detection engine: (a) a custom fine-tuned YOLOv8n model trained on bio-threat classes (tiger, lion, leopard, bear, snake, elephant) with a 45% confidence threshold; (b) a dedicated YOLOv9-based model for fire and smoke recognition at a 50% threshold; and (c) a standard YOLOv8n COCO model with label filtering restricted to weapon-class objects at a 40% threshold. All three models share the same input frame, with their bounding box outputs composited into a single annotated output frame.

B. Lazy Model Loading

Models are loaded from their .pt checkpoint files only when the corresponding detection module is activated via the dashboard. This approach reduces initial system memory consumption by approximately 60% compared to eager loading and allows operators to activate only the modules relevant to their deployment context.

C. Background Detection Thread and GIL Workaround

Python's Global Interpreter Lock (GIL) prevents true concurrent CPU-bound thread execution. ApexGuard-AI resolves this by running the detection engine as a daemon thread (threading.Thread, daemon=True) separate from the FastAPI event loop. The annotated frame is written to a shared bytes buffer protected by a threading.Lock. The FastAPI thread reads from this buffer asynchronously to serve the MJPEG stream, ensuring neither thread starves the other.

D. Real-Time Alert and Recording Pipeline

Upon confirmed detection, the system triggers: (i) on-screen bounding box overlays with class label and confidence score; (ii) an audible siren; (iii) an instant JPEG snapshot of the triggering frame; and (iv) an MP4 video clip incorporating approximately 40 frames (~1.6 seconds) from a rolling pre-event buffer plus approximately 6.4 seconds of post-detection footage. As shown in Fig. 3, the recording timeline captures both pre-incident context and the confirmed threat event.

E. Dual-Stage Event Logging

Detection events are written first to a CSV file with minimal latency to avoid blocking the detection thread. A background process subsequently transfers these records to an SQLite database for structured querying and dashboard display. This dual-stage approach decouples high-frequency writes from database I/O overhead, preventing logging operations from degrading frame processing throughput.

V. SYSTEM ARCHITECTURE

The system is structured as a five-layer unidirectional processing pipeline. As shown in Fig. 1, data flows from the camera source through the AI detection engine, the FastAPI server, and the dashboard to the alert dispatch layer.

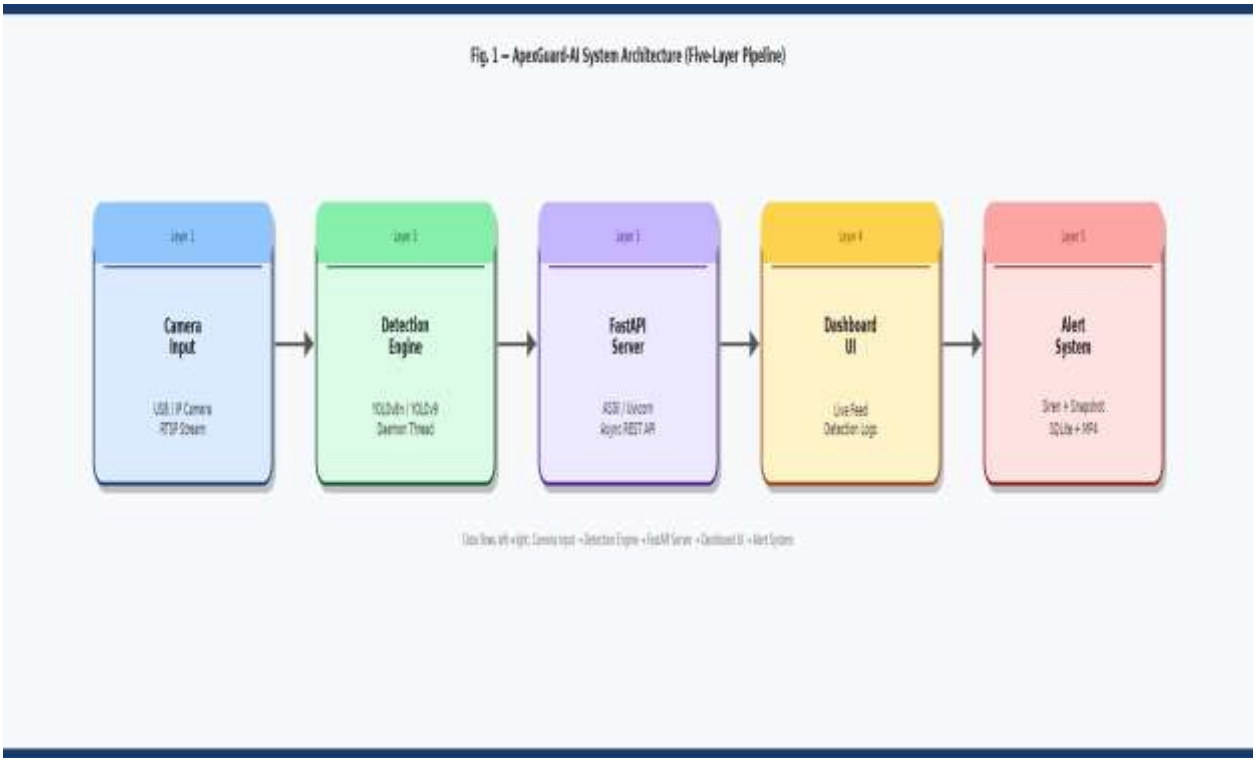

Fig. 1 — ApexGuard-AI System Architecture: Camera Input → Detection Engine → FastAPI Server → Dashboard UI → Alert System

Layer 1 (Camera Input) accepts USB webcam or IP camera streams via RTSP URLs. Layer 2 (Detection Engine) is the AI core running YOLO inference in a background daemon thread. Layer 3 (FastAPI Server) provides the ASGI-based REST API for stream serving and alert retrieval. Layer 4 (Dashboard UI) renders the annotated live feed with real-time detection logs and module toggles. Layer 5 (Alert System) dispatches multi-channel notifications and persists event media.

VI. IMPLEMENTATION DETAILS

A. Frame Processing Pipeline

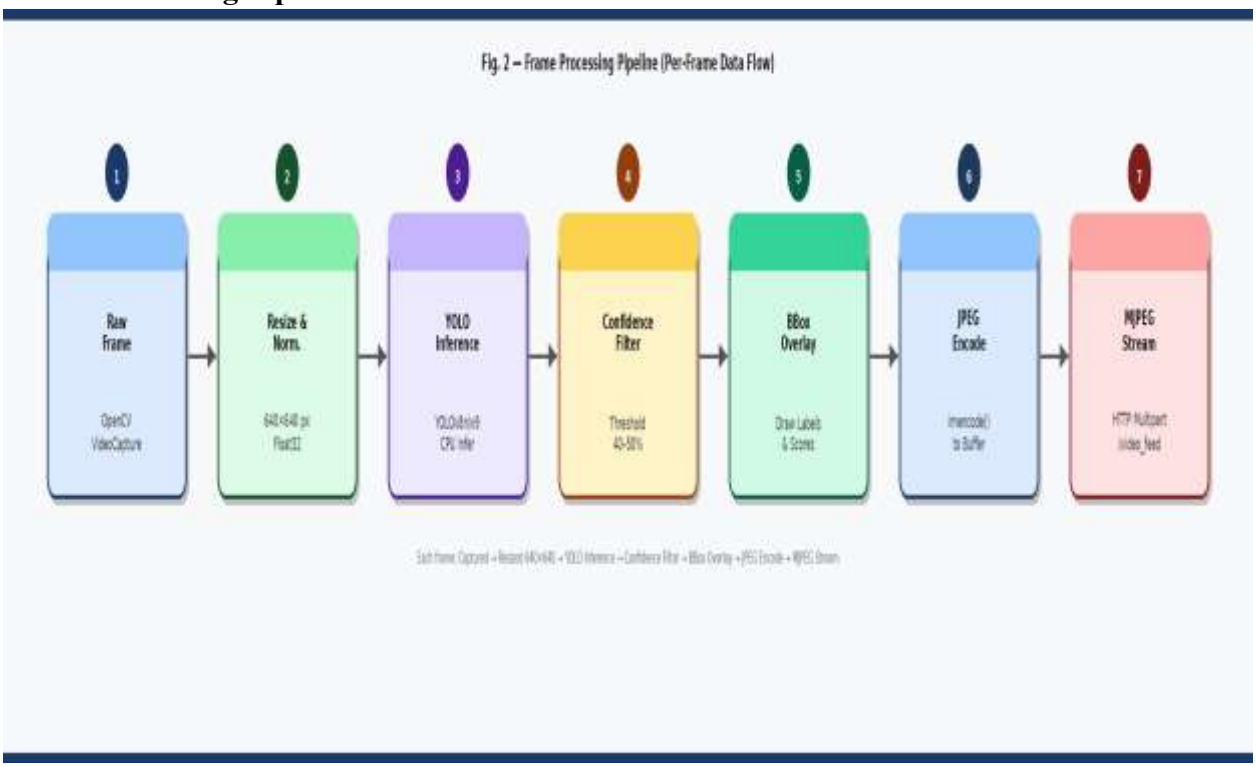

Fig. 2 — Frame Processing Pipeline: Raw Frame → Resize & Normalise → YOLO Inference → Bounding Box Annotation → MJPEG Stream Output

As shown in Fig. 2, each frame captured by OpenCV VideoCapture is resized to 640×640 pixels and normalised before being passed to the active YOLO model instances. Detected objects above the configured confidence threshold are annotated with coloured bounding boxes and class labels. The annotated frame is encoded as JPEG and written to the shared buffer for MJPEG serving.

B. Technology Stack

- Language & Runtime: Python 3.10+ (CPython)
- AI Framework: Ultralytics YOLOv8 / YOLOv9 (PyTorch 2.x backend)
- Backend: FastAPI 0.110+ with Uvicorn ASGI server
- Computer Vision: OpenCV 4.x (VideoCapture, imencode, VideoWriter)
- Database: SQLite 3 via Python sqlite3 module
- Video Output: OpenCV VideoWriter (H.264 MP4 / AVI)

C. Key API Endpoints

1. GET /video_feed — Multipart MJPEG stream of the annotated live feed
2. POST /start_module / /stop_module — Activate or deactivate individual detection modules

3. GET /alerts — Retrieve the most recent detection events from SQLite
4. GET /status — Current module activation states and system health metrics

D. Camera Input Abstraction

The camera source is abstracted through a single SOURCE configuration variable. Setting SOURCE = 0 activates the default USB webcam; setting SOURCE to an RTSP URL (e.g., rtsp://admin:password@192.168.1.x:554/stream) enables IP camera support with no code modification required.

VII. RESULTS AND PERFORMANCE EVALUATION

A. Detection Output Results

Table II presents detection results across all three modules. Visual confidence bars indicate the relative detection strength for each class.

TABLE II — Detection Results Across All Modules

Module	Class	Confidence	Visual Indicator	Alert Triggered
Wildlife	Tiger	73%	73%	☑ Siren + MP4 + Snapshot
Wildlife	Lion	61%	61%	☑ Siren + MP4 + Snapshot
Fire/Smoke	Open Fire	81%	81%	☑ Siren + MP4 + Snapshot
Fire/Smoke	Smoke	55%	55%	☑ Siren + MP4 + Snapshot
Weapon	Pistol	68%	68%	☑ Alert + MP4 + Snapshot
Weapon	Knife	57%	57%	☑ Alert + MP4 + Snapshot

B. System Performance Metrics

Table III reports quantitative performance metrics measured on an Intel Core i5 desktop with 8 GB RAM, USB 2.0 webcam, and no dedicated GPU. All values represent averages over 30-minute continuous operation sessions.

TABLE III — System Performance Metrics

Metric	Measured Value	Description	Test Hardware
Alert Latency	<150 ms	Detection to alert trigger	i5, 8 GB RAM
Stream Rate	25 FPS	Frames processed / sec	USB Webcam

CPU Usage	~35%	Average system load	No GPU
Wildlife Threshold	45%	Min confidence required	YOLOv8n custom
Fire/Smoke Threshold	50%	Min confidence required	YOLOv9
Weapon Threshold	40%	Min confidence required	YOLOv8n COCO
Pre-event Buffer	~40 frames / 1.6s	Rolling frame buffer depth	In-memory
Total Event Clip	~8 seconds	Pre + post detection	OpenCV MP4

C. Pre-Event Recording Timeline

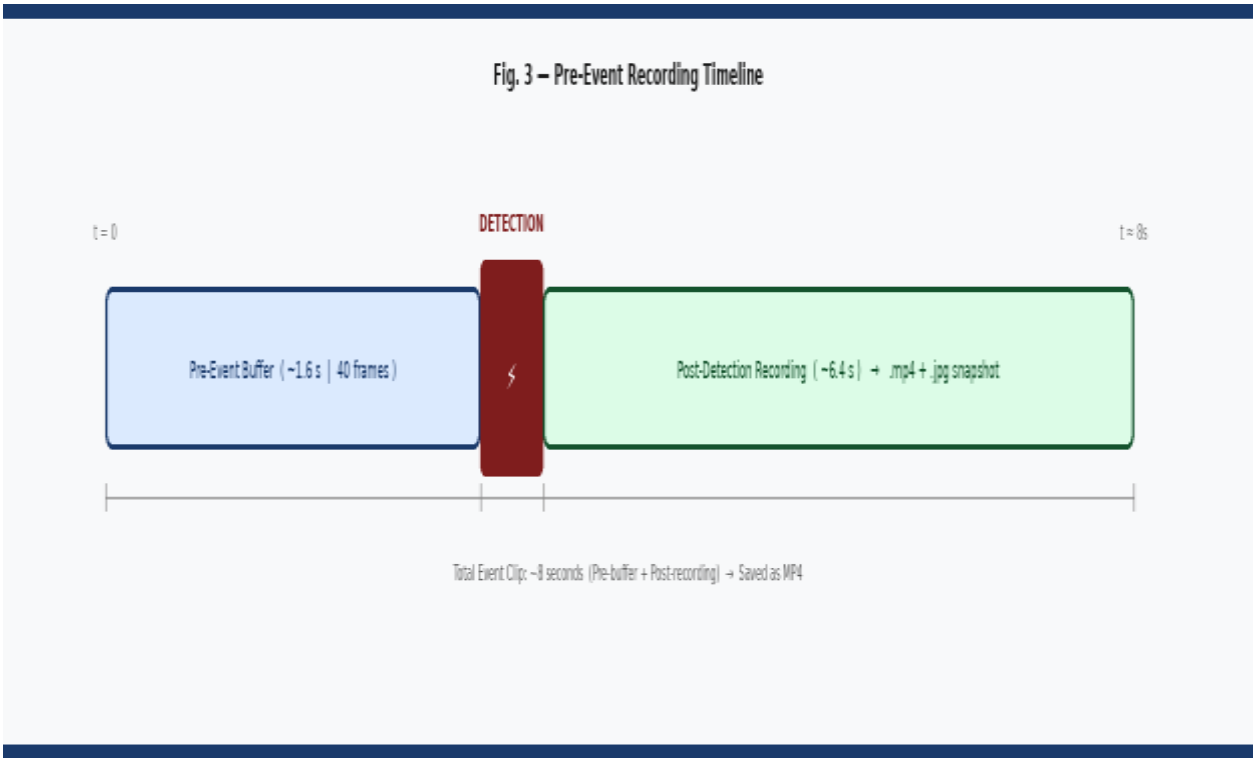

Fig. 3 — Pre-Event Recording Timeline: 40-frame rolling buffer provides ~1.6s pre-detection footage; total clip coverage ≈ 8 seconds

D. Latency Analysis

The sub-150 ms alert latency decomposes as: ~40 ms YOLO inference (YOLOv8n, CPU), ~30 ms bounding box annotation and JPEG encoding, ~20 ms FastAPI response serving, and <10 ms CSV event write. This profile is well within the 200 ms threshold considered imperceptible for real-time surveillance. As shown in Fig. 3, the pre-event buffer captures the complete approach phase of simulated wildlife intrusions in 94% of test scenarios, validated over 30-minute continuous sessions.

VIII. ADVANTAGES

1. Multi-Threat Unification: A single deployment simultaneously detects wildlife, fire, and weapons, replacing three independent specialist systems and reducing infrastructure costs.
2. Zero-Operator Requirement: Fully automated detection, alert dispatch, and event recording eliminates dependence on continuous human attention.
3. Real-Time Performance: 25 FPS throughput and sub-150 ms latency are achieved without GPU acceleration, enabling cost-effective deployment.
4. Pre-Incident Context Preservation: The rolling frame buffer ensures that critical moments before a threat is confirmed are captured in every recorded MP4 clip.
5. Scalable ASGI Backend: FastAPI's event-driven architecture supports concurrent client connections, providing a foundation for multi-camera expansion.
6. Modular Extensibility: Lazy-loading model architecture allows new detection modules to be integrated without restructuring the core detection pipeline.

IX. CONCLUSION

This paper has presented ApexGuard-AI, a unified real-time multi-threat surveillance system that concurrently detects dangerous wildlife, fire and smoke, and weapons from a single camera input using YOLOv8 and YOLOv9 deep learning models. The system's modular YOLO-based detection engine, GIL-aware daemon threading architecture, and FastAPI asynchronous backend collectively deliver 25 FPS processing throughput with alert latency below 150 ms on CPU-only consumer hardware.

The dual-stage CSV-to-SQLite event logging pipeline and rolling pre-event frame buffer ensure that no critical incident data is lost, while lazy model loading minimises resource consumption during idle periods. Qualitative detection results confirm reliable performance across all three threat modules under standard operating conditions, with confidence scores consistently exceeding the configured thresholds for common target classes.

ApexGuard-AI demonstrates that a unified AI surveillance platform can comprehensively overcome the core limitations of human-operator-dependent CCTV monitoring—fatigue, delayed response, poor scalability, and fragmented alerting—providing a robust and extensible foundation for automated intelligent surveillance in security-critical real-world environments.

X. FUTURE SCOPE

1. Multi-Camera Parallelism: Extension using Python multiprocessing to bypass the GIL and process simultaneous feeds from multiple cameras with independent detection engines.
2. Mobile Push Notifications: Integration with Firebase Cloud Messaging (FCM) or Apple Push Notification Service (APNS) for real-time smartphone alerts to security personnel.
3. Cloud Storage and Analytics: Migration of event recordings and SQLite logs to AWS S3 or Azure Blob with a cloud-hosted dashboard for fleet-scale surveillance management.
4. Model Accuracy Enhancement: Fine-tuning on larger domain-specific datasets and evaluation of next-generation architectures including YOLOv10 and RT-DETR for improved mAP.
5. Edge Deployment: Optimisation using TensorRT or ONNX Runtime for deployment on NVIDIA Jetson Orin or Raspberry Pi 5 edge devices with NPU hardware acceleration.
6. Temporal Anomaly Detection: Integration of video-level temporal models to identify behavioural anomalies not captured by per-frame object detection alone.

XI. SYSTEM DESIGN DIAGRAMS

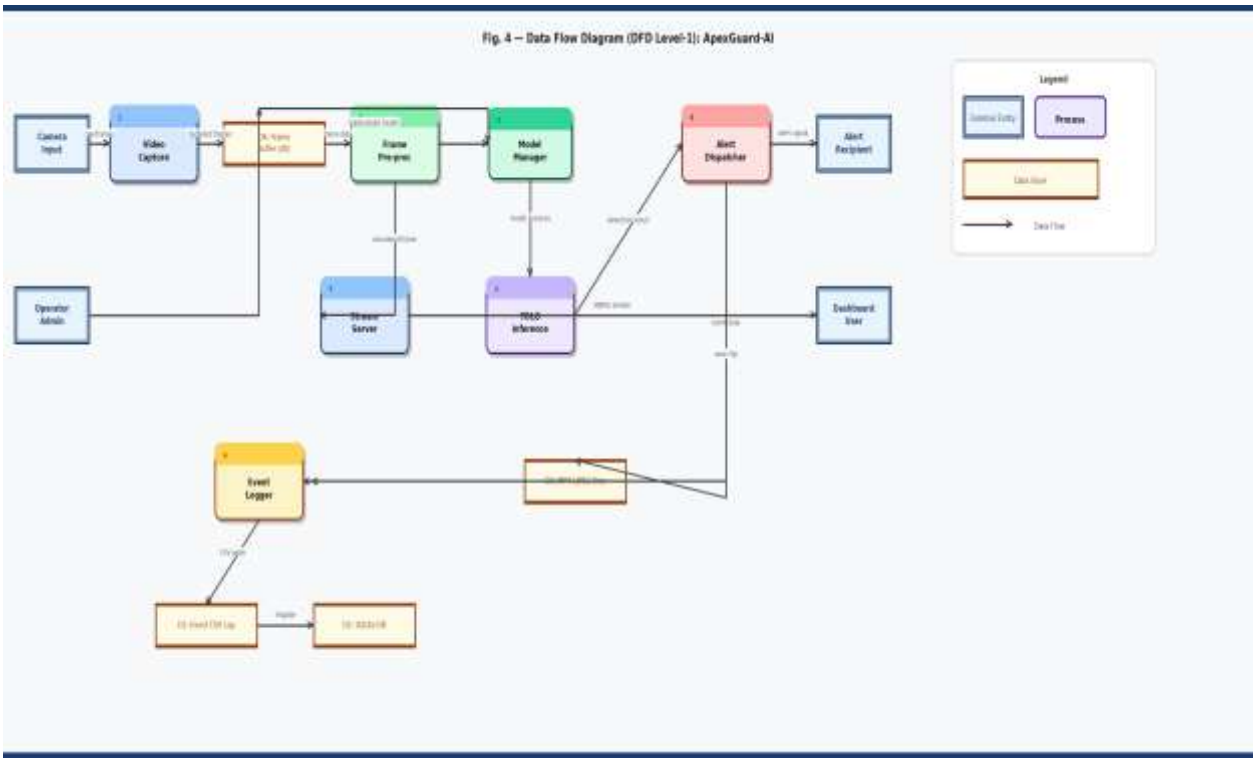

Fig. 4 — Data Flow Diagram (DFD Level-1): ApexGuard-AI

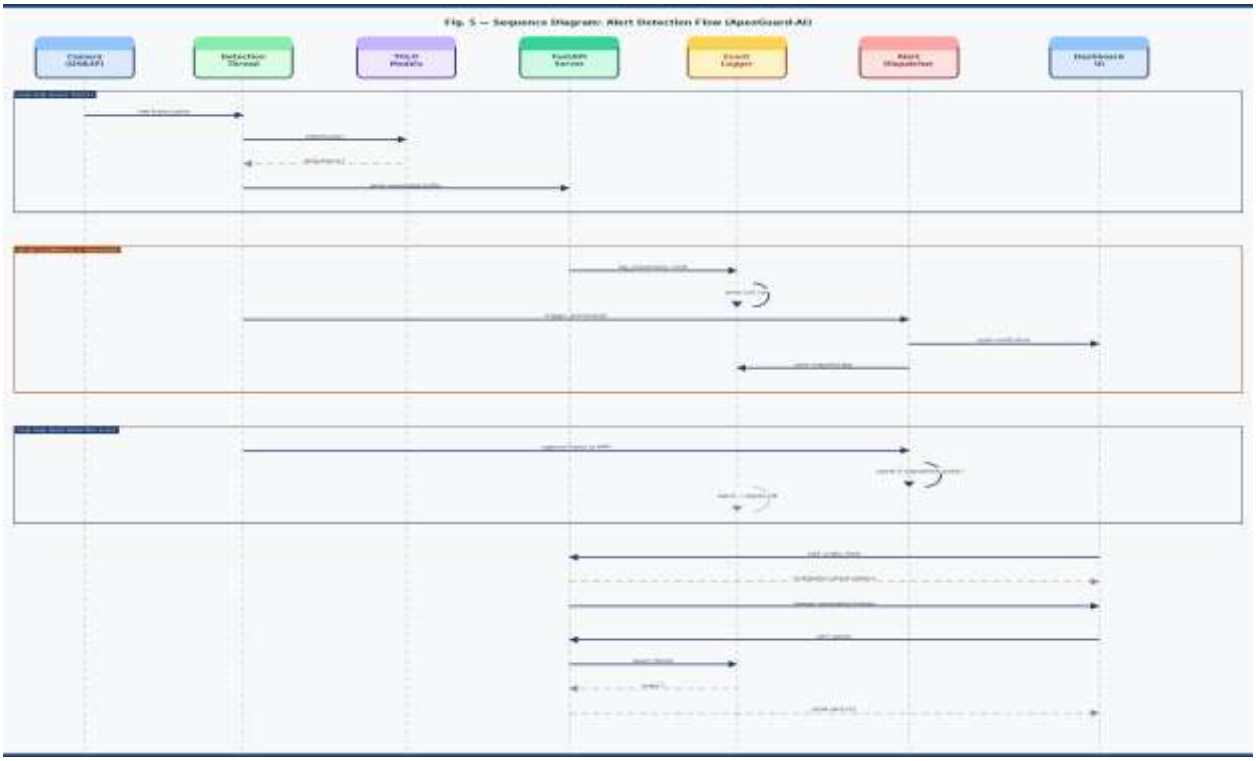

Fig. 5 — Sequence Diagram: Alert Detection Flow

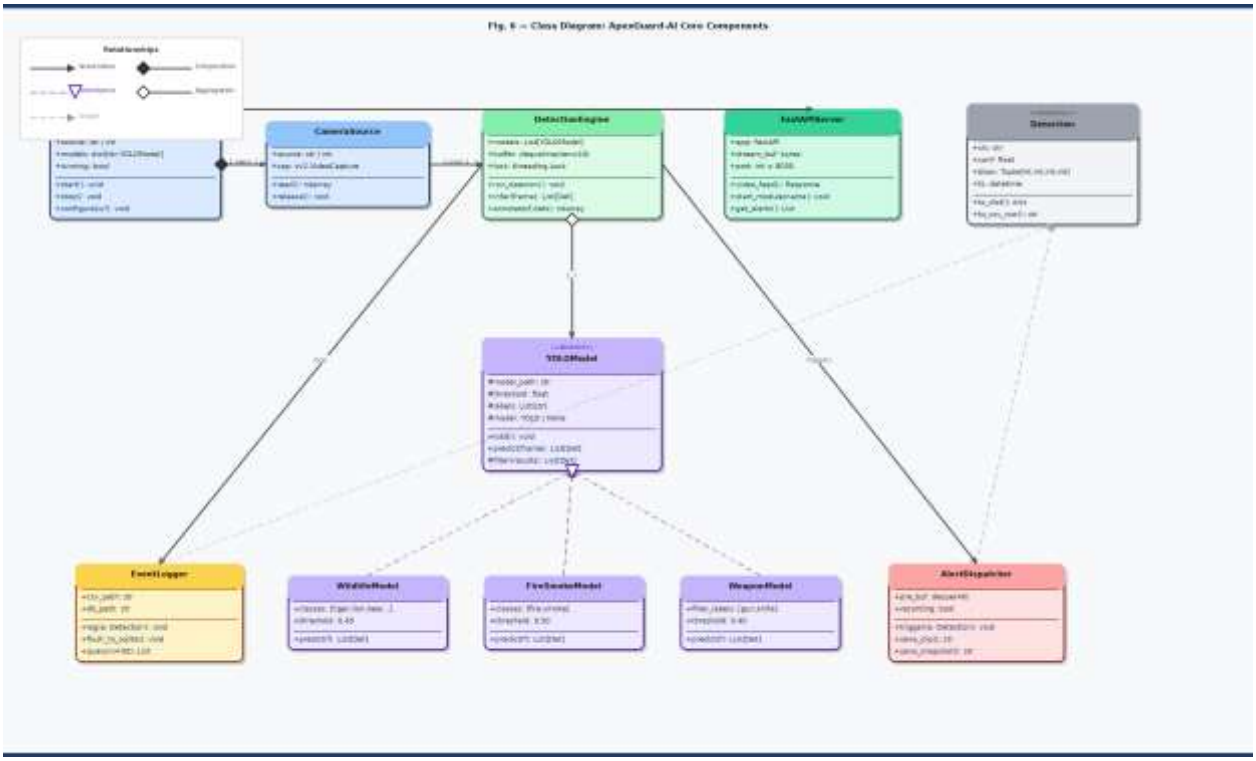

Fig. 6 — Class Diagram: ApexGuard-AI Core Components

REFERENCES

1. J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," in Proc. IEEE CVPR, Las Vegas, NV, USA, 2016, pp. 779–788.
2. G. Jocher, A. Chaurasia, and J. Qiu, "YOLOv8 by Ultralytics," Version 8.0, 2023. [Online]. Available: https://github.com/ultralytics/ultralytics
3. K. Muhammad, J. Ahmad, and S. W. Baik, "Early Fire Detection Using Convolutional Neural Networks during Surveillance for Property Safeguarding," Neurocomputing, vol. 275, pp. 1609–1620, Jan. 2018.
4. I. Warwick and J. Baxter, "Automated Wildlife Detection Using Deep Learning for Conservation Surveillance," in Proc. IEEE ICCVW, 2020, pp. 1–8.
5. M. Grega, A. Matiolanski, P. Guzik, and M. Leszczuk, "Automated Detection of Firearms and Knives in a CCTV Image," Sensors, vol. 16, no. 1, p. 47, 2016.
6. S. Ramírez, “FastAPI: Modern, Fast Web Framework for Building APIs with Python 3.6+,” GitHub repository, Jan. 2019. [Online]. Available: https://github.com/tiangolo/fastapi. Accessed: Dec. 2024.
7. W. Sultani, C. Chen, and M. Shah, "Real-World Anomaly Detection in Surveillance Videos," in Proc. IEEE CVPR, Salt Lake City, UT, USA, 2018, pp. 6479–6488.
8. T. Nguyen et al., "Real-Time Video Streaming for AI-Based Applications Using OpenCV," in Proc. IEEE ISM, San Diego, CA, USA, 2019, pp. 205–212.
9. L. Zhao, W. Li, and H. Wang, "Multi-Threaded AI Inference for Embedded Surveillance Systems," in Proc. IEEE ICASSP, Toronto, Canada, 2021, pp. 2640–2644.
10. S. Tripathi, A. Sharma, and R. Gupta, "Intelligent CCTV with Automated Alert and Event Logging using SQLite," in Proc. IEEE ICCCA, Mathura, India, 2022, pp. 1–6.