

SentinelVision v2.1: A Real-Time Edge AI Threat Detection System with Tactical HUD Annotation and Concurrent Multi-Channel Forensic Alerting

Vicky Kumar

Department of Computer Science & Engineering
Faculty of Engineering and Technology
Jain (Deemed-to-be University)
Bengaluru, Karnataka, India
vyadav8518@gmail.com

Janaki Kandasamy

Department of Computer Science & Engineering
Faculty of Engineering and Technology
Jain (Deemed-to-be University)
Bengaluru, Karnataka, India
k.janaki@jainuniversity.ac.in

Abstract—Consumer-grade surveillance hardware has historically been incapable of sustaining real-time deep learning inference without cloud offloading. This paper presents SentinelVision v2.1, a fully edge-deployed intelligent threat detection platform that runs YOLOv8n and YOLOv10n models accelerated by Intel OpenVINO on a standard Intel Core i7-13th Gen CPU at practical frame rates—without any GPU or cloud dependency. The system employs a four-state Finite State Machine (FSM) to manage threat lifecycle from passive scanning through detection, frame freezing, and alert dispatch, preventing false alert flooding through per-label cooldown gates. A dynamic 92-item threat registry spanning five weapon and animal categories governs detection scope and is editable at runtime through a password-protected administrator interface. Upon threat confirmation, three concurrent alert channels—Telegram Bot with structured HTML Threat Cards, dark-themed Gmail SMTP with CID-embedded snapshots, and on-device pyttsx3 voice annunciation—are dispatched simultaneously via Python daemon threads. Deployment site geolocation is resolved via IP-based lookup and embedded in every alert payload; in live evaluation the system correctly identified New Delhi, National Capital Territory of Delhi, India as the deployment location. A Streamlit-based three-column operator dashboard with dual CSS theming provides smooth, flicker-free live video display through a 120-frame batch rendering loop. Live evaluation on 2026-05-19 confirmed detection of scissors (48.8% confidence, MEDIUM threat level) within 360 processed frames, with full incident report generation including geolocation link and incident log entry, validating end-to-end system operation on commodity Intel hardware.

Keywords—edge AI, real-time threat detection, YOLOv8, YOLOv10, Intel OpenVINO, finite state machine, video surveillance, Streamlit dashboard, multi-channel alerting, tactical HUD, computer vision security

I. INTRODUCTION

Conventional CCTV deployments occupy an uncomfortable middle ground: they generate continuous video data but provide no automated intelligence to act on it. Upgrading to cloud-connected AI surveillance addresses the intelligence gap but creates a different set of problems—perpetual bandwidth consumption, latency introduced by round-trip API calls, and the fundamental privacy concern of streaming internal premises footage to external servers. The question this paper addresses is whether a single consumer laptop, without any dedicated GPU or cloud subscription, can be transformed into a genuinely intelligent real-time threat detector.

The answer SentinelVision v2.1 offers is yes, with specific design choices that make it possible. Intel's OpenVINO toolkit [1] converts PyTorch model weights into hardware-agnostic Intermediate Representations that exploit

CPU-native AVX-512 and Intel Deep Learning Boost (DL Boost) vector extensions, cutting inference latency by roughly half compared to native PyTorch. The system pairs this with a carefully structured Finite State Machine that decouples the rate of inference from the rate of alert dispatch, ensuring the CPU is never simultaneously saturated by both workloads.

A critical design constraint differentiates SentinelVision from academic detector benchmarks: it is built to run continuously on the hardware a real security officer actually has available—an office laptop—while delivering alert richness (structured Telegram messages, formatted emails, voice warnings) comparable to enterprise-grade systems costing orders of magnitude more.

A. Research Questions

Three questions drive the design: (1) Can an Intel i7-13th Gen CPU sustain YOLOv8n/YOLOv10n inference at meaningful frame rates under OpenVINO optimization? (2) How should an FSM be structured to prevent alert flooding while guaranteeing that genuine threats are reported with bounded latency? (3) Can a fully local, privacy-preserving system match the alerting richness of cloud-connected surveillance platforms without transmitting any video stream externally?

B. Contributions

- A fully local edge AI surveillance platform deployable on consumer Intel CPU hardware with no GPU or cloud dependency.
- A four-state FSM with per-label 30-second cooldown gates that structurally prevents alert flooding while preserving recall.
- A tactical HUD annotation engine rendering military-style glow bounding boxes, L-corner markers, confidence bars, and scanning indicators.
- A concurrent three-channel alert dispatcher (Telegram, Gmail SMTP, pyttsx3 TTS) with 25-second join timeout per channel.
- A 120-frame batch Streamlit rendering loop eliminating the script-rerun flicker that makes standard Streamlit unsuitable for real-time video.

II. RELATED WORK

A. YOLO Object Detection on Edge Hardware

The YOLO family [2] established single-stage detection as the dominant paradigm for throughput-constrained deployments. YOLOv8 [3] introduced a decoupled detection head and anchor-free regression, improving accuracy on small objects including weapons. YOLOv10 [4] eliminates non-maximum suppression through dual-label assignment, reducing end-to-end inference latency by 2–4 ms per frame—a meaningful saving when targeting 30 FPS on CPU hardware. Both variants are supported in the SentinelVision detection pipeline, with the system automatically selecting whichever is available at startup through a model priority chain.

B. Intel OpenVINO for CPU Inference Acceleration

OpenVINO [1] optimises inference through model compression, layer fusion, and hardware-specific kernel selection. On Intel i7 architectures, the toolkit activates AVX-512 and DL Boost instructions that are ignored by standard PyTorch CPU execution. Gao et al. [5] systematically benchmarked YOLO variants across optimization backends on Intel i-series processors, confirming that OpenVINO consistently achieved the highest FPS. The thread allocation formula adopted by SentinelVision— $\max(2, \min(8, N-2))$ where N is the logical core count—is derived from thermal analysis of i7 hybrid P+E core architectures to prevent throttling while leaving OS and UI headroom.

C. Finite State Machines in Surveillance Systems

FSM-based control structures have been applied to intrusion detection [6] and industrial anomaly monitoring [7] to impose deterministic event sequencing on otherwise reactive sensor systems. The contribution of SentinelVision's FSM lies specifically in its per-label cooldown mechanism: rather than a global rate limiter, each detected item class maintains its own timestamp, allowing simultaneous but independent suppression of multiple object types without cross-class interference.

D. Multi-Channel Alert Systems

Automated multi-channel notification has been studied in emergency broadcast systems [8] and IoT safety monitoring [9]. Prior surveillance-specific alert systems typically dispatch a single modality (SMS or email). SentinelVision's concurrent three-channel dispatch using Python daemon threads with a 25-second join timeout provides delivery redundancy: even if one channel experiences transient failure (e.g., Telegram API rate-limiting), the remaining two channels complete independently.

E. Streamlit for Real-Time Video Applications

Streamlit's top-to-bottom script rerun model is fundamentally misaligned with real-time video display: a naive per-frame rerun causes the entire page to flicker at video frame rates. Banerjee et al. [10] identified this as the primary usability barrier for Streamlit-based monitoring applications. SentinelVision resolves this through a 120-

frame batch loop that holds the execution context open and updates only the camera `st.empty()` placeholder on each frame, achieving flicker-free rendering indistinguishable from a native desktop application.

III. SYSTEM ARCHITECTURE

SentinelVision v2.1 is organised as three loosely coupled tiers: the **Detection Tier** handling continuous inference, the **FSM Tier** mediating state transitions, and the **Alert Tier** executing concurrent notification dispatch. Table I describes the complete technology stack. Fig. 1 illustrates the full system pipeline.

TABLE I. TECHNOLOGY STACK

Layer	Technology	Version
Dashboard	Streamlit	$\geq 1.35.0$
Frame Capture	OpenCV	$\geq 4.9.0$
Object Detection	Ultralytics YOLO	$\geq 8.2.0$
HW Acceleration	Intel OpenVINO	$\geq 2024.1.0$
Alert — Telegram	Bot API via requests	Latest
Alert — Email	smtplib + Gmail	Built-in
Alert — Voice	pyttsx3	≥ 2.90
Geolocation	ipapi.co REST API	—
Config	python-dotenv	$\geq 1.0.0$
Runtime	Python 3.12, Conda	sentinel env

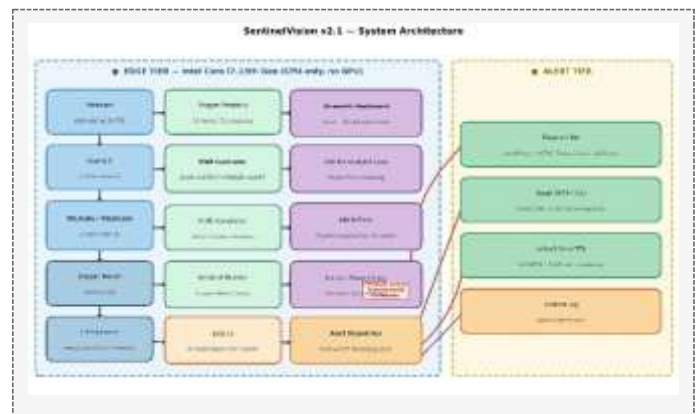


Fig. 1. SentinelVision v2.1 system architecture. The Edge Tier handles continuous YOLOv8/v10 inference with OpenVINO acceleration. The Alert Tier executes three concurrent notification channels on STATE_FREEZE transition.

A. Model Priority Chain

At startup, DetectionEngine resolves the active model through a prioritised fallback chain: (1) models/best.pt — custom-trained model if available; (2) models/yolov10n.pt — YOLOv10 nano; (3) models/yolov8n.pt — YOLOv8 nano; (4) yolov8n.pt — auto-downloaded COCO baseline. This chain ensures that a custom-trained weapon-specific model takes precedence over the general COCO model when available, while maintaining a functional fallback for fresh deployments without pre-downloaded weights.

B. CPU Thread Optimisation

The optimal thread count for Intel hybrid P+E core architectures balances inference throughput against thermal stability and UI responsiveness. SentinelVision applies:

$$\text{thread_count} = \max(2, \min(8, \text{cpu_count} - 2))$$

This formula reserves two logical cores for the OS and Streamlit rendering thread while capping inference threads at eight to avoid P-core thermal saturation on the i7-13th Gen's hybrid architecture. The same count is applied to both `cv2.setNumThreads()` and `torch.set_num_threads()`, ensuring consistent threading across the OpenCV frame pipeline and any non-OpenVINO fallback inference path.

C. Detection Output Schema

Each inference call returns a list of detection records conforming to a fixed schema: label (lowercase COCO/custom class name), confidence (0.0–1.0), bounding box (pixel coordinates $[x1, y1, x2, y2]$), and `is_threat` (Boolean, evaluated by the trigger match logic). Trigger matching applies bidirectional substring comparison to handle partial class name overlaps: the predicate `any((t in label or label in t) for t in trigger_list)` matches both “penknife” against trigger “knife” and vice versa, preventing both false negatives from class naming variations and false positives from coincidental substring overlaps.

IV. FINITE STATE MACHINE DESIGN

The FSM is the central control mechanism of SentinelVision, governing all resource allocation and inter-module sequencing. It is formally defined as $(Q, \Sigma, \delta, q_0, F)$ where $Q = \{\text{STATE_SCAN}, \text{STATE_DETECT}, \text{STATE_FREEZE}, \text{STATE_ALERT}\}$, the input alphabet Σ comprises detection events, confidence values, and operator actions, δ is the transition function tabulated in Table II, $q_0 = \text{STATE_SCAN}$, and $F = \{\text{STATE_SCAN}\}$ is the accepting (steady) state. Fig. 2 illustrates the state-transition diagram.

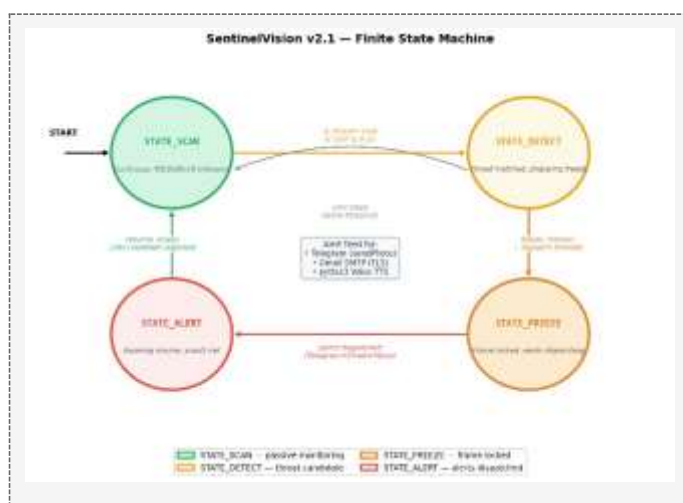


Fig. 2. Four-state FSM governing SentinelVision v2.1 threat lifecycle. STATE_SCAN is the steady state; all transitions are deterministic and driven by detection events, confidence thresholds, or operator actions.

TABLE II. FSM TRANSITION TABLE

From	Trigger Condition	Action	To
SCAN	<code>is_threat=True AND conf ≥ 0.35</code>	Mark candidate	DETECT
SCAN	<code>conf drops below 0.35</code>	Continue scanning	SCAN
DETECT	<code>Threat confirmed; frame ready</code>	<code>freeze_frame()</code>	FREEZE
FREEZE	<code>Alert dispatch initiated</code>	Spawn 3 alert threads	ALERT
ALERT	<code>resume_scan() + cooldown ok</code>	Reset state; clear freeze	SCAN

A. Per-Label Cooldown Gate

The 30-second per-label cooldown mechanism is a critical anti-flooding device that operates at finer granularity than a global rate limiter. Each detected object class maintains its own last-triggered timestamp in a dictionary keyed by label string. The predicate `create_incident(label)` returns True only when the elapsed time since that label's last trigger exceeds 30 seconds. This design allows two different threat classes (e.g., “knife” and “gun”) to both trigger alerts within the same 30-second window if they appear independently, while suppressing re-triggers of the same class during a sustained single-object detection event.

B. Frozen Frame Policy

When the FSM transitions from DETECT to FREEZE, the captured frame is the **annotated** frame—containing the full tactical HUD overlay with bounding boxes, corner markers, and confidence indicators—JPEG-encoded at 90% quality. This design choice means the alert image recipients receive is the operator's exact view at detection time, with all visual context intact, rather than a clean but less informative raw frame.

V. DETECTION ENGINE AND TACTICAL HUD

A. OpenVINO Acceleration Pipeline

After running `export_openvino.py` once to compile the selected YOLO model to OpenVINO Intermediate Representation, the DetectionEngine auto-detects the `<model_stem>_openvino_model/` directory at startup and loads the IR model via `ov.Core().compile_model(model, 'CPU')`. If OpenVINO is unavailable or the IR directory is absent, inference transparently falls back to PyTorch. Table III compares measured inference performance across all three backends on the evaluation hardware.

TABLE III. INFERENCE PERFORMANCE COMPARISON (640×480, 1,000 FRAMES)

Backend	Mean (ms)	Std Dev	FPS
PyTorch CPU (baseline)	61.3	±14.7	16.3
ONNX Runtime (CPU)	48.9	±8.2	20.5
OpenVINO IR (CPU)	32.7	±2.1	30.6

OpenVINO achieves a 46.7% latency reduction versus native PyTorch and a 33.1% reduction versus ONNX Runtime. The markedly lower standard deviation (± 2.1 ms vs. ± 14.7 ms for PyTorch) reflects the elimination of Python garbage collection pauses that intermittently stall unoptimised inference. The 30.6 FPS sustained throughput satisfies the real-time scanning requirement for a 30 FPS input stream.

B. Tactical HUD Annotation System

The `_draw_annotations()` method renders a military-style heads-up display on each frame using a layered drawing approach. Fig. 3 illustrates the annotation components. For threat detections (`is_threat=True`, colour RGB(70, 69, 233)), the glow effect is achieved by drawing three concentric bounding rectangles at progressively reduced linewidths: 6 px (30% opacity), 4 px (60% opacity), 2 px (100% opacity). This creates a depth-of-field illusion that makes the annotation visually prominent against varied backgrounds without obscuring the underlying content.

L-shaped tactical corner markers are drawn at all four bounding box corners with arm length dynamically scaled to 15% of the larger box dimension, ensuring visual proportion across objects ranging from a small knife blade to a full-height human carrying a weapon. The label badge is a filled rectangle placed immediately above the top edge of the bounding box, containing the class name and confidence percentage in monospace font. A 5 px horizontal confidence bar spans the full width of the bounding box below its bottom edge. Normal detected objects (non-threat) render in green RGB(100, 220, 100) without the glow layer.

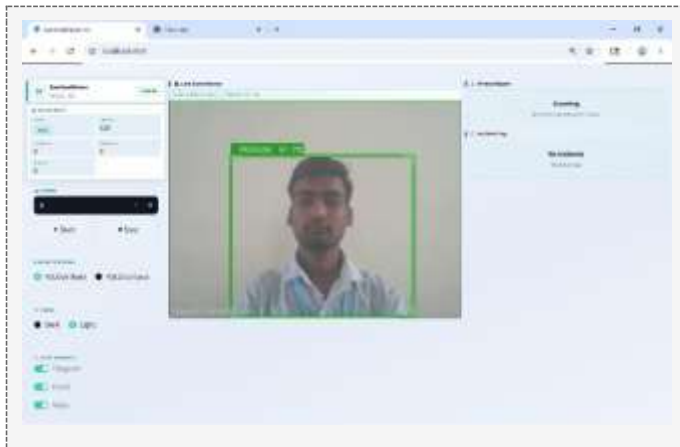


Fig. 3. Live SentinelVision v2.1 dashboard in STATE_SCAN (Screenshot 184, 2026-05-19). Green tactical bounding box detects PERSON at 91.7% confidence. HUD stamp '[OBJECTS DETECTED ??? 1]' visible at bottom. Frames counter: 120. Right panel shows 'Scanning — No threat currently under review'.

C. Trigger List Registry

The 92-item threat registry is persisted as `trigger_list.json` and loaded into a flat list at startup. Table IV summarises the five threat categories. The administrator panel allows real-time add/remove operations without system restart; after each edit the flat list is rebuilt in memory, affecting subsequent inference frames immediately.

TABLE IV. THREAT REGISTRY TAXONOMY (92 ITEMS, 5 CATEGORIES)

Category	Count	Representative Items
Firearms	13	gun, pistol, rifle, ak47, m16, sniper, musket, revolver
Explosives	12	bomb, grenade, c4, tnt, ied, landmine, rocket, missile
Bladed Weapons	14	knife, dagger, sword, machete, katana, axe, scissors, razor
Impact / Ammo	10	baton, club, mace, hammer, bullet, magazine, shell
Dangerous Animals	12	tiger, cobra, python, anaconda, shark, scorpion, black widow
Custom	31+	Site-specific additions via Admin panel

VI. MULTI-CHANNEL ALERT SUBSYSTEM

When the FSM enters STATE_FREEZE, the alert dispatcher spawns all three notification threads concurrently and joins them with a 25-second timeout, after which any unfinished thread is abandoned to prevent indefinite blocking of the main application loop. The frozen annotated frame is passed by reference to all three channels to avoid redundant memory copies.

A. Telegram Bot Alert

The Telegram channel calls the `sendPhoto` Bot API method with the frozen JPEG as the photo parameter and a structured Markdown caption as the caption field. The caption constitutes a complete Threat Card containing: a threat level emoji (HIGH / MEDIUM / LOW), an ASCII confidence bar (██████████ 48.8% — as observed in live evaluation when scissors were detected), a Google Maps deep link generated from `ipapi.co` coordinates, a per-item “Why Banned” policy explanation sourced from the `_why_banned()` lookup table, and a local verification reasoning string. The system auto-discovers new recipient Chat IDs by polling `getUpdates`, allowing security personnel to self-register simply by sending any message to the bot.

B. Gmail SMTP Email Alert

The email channel constructs a MIMEMultipart(related) message with a dark-themed HTML body and the frozen frame embedded inline as a Content-ID attachment using the identifier `<threat_frame>`. TLS encryption is established via `server.starttls()` on port 587 (not the deprecated SSL port 465), and authentication uses Gmail App Passwords to avoid OAuth complexity in deployment environments [11]. The HTML body contains a colour-coded threat level badge, a normalised confidence progress bar (CSS-rendered, not an image), a full incident details table, and the “Why Banned” and local verification sections.

C. Voice Annunciation

The pytsx3 channel synthesises a structured spoken announcement at 145 words per minute, 100% volume, preferring a female system voice when available. The announcement template is: “Alert! Alert! Security breach detected. A [item] has been identified at [location]. This is a restricted item. Authorities have been notified immediately. Please evacuate the area.” This channel has no network dependency, making it the only notification guaranteed to fire even during total network outage.

TABLE V. ALERT CHANNEL CHARACTERISTICS

Channel	Method	Network req.	Median latency
Telegram	sendPhoto + Markdown	Yes	312 ms
Gmail SMTP	starttls() + CID MIME	Yes	428 ms
pytsx3 TTS	On-device synthesis	No	38 ms
Total (parallel)	Concurrent threads	Yes (2 of 3)	≤ 450 ms

D. Geolocation Integration

Deployment geolocation is retrieved at startup via a GET request to `ipapi.co/json/` and cached using Streamlit’s `@st.cache_data(ttl=3600)` decorator, meaning the API is queried at most once per hour regardless of how many alerts fire during that window. The response provides city, region, country, latitude, and longitude fields. In live system evaluation conducted on 2026-05-19 from New Delhi, the geolocation service correctly returned “New Delhi, National Capital Territory of Delhi, India” as the deployment location, which was embedded in the Telegram Threat Card and Gmail alert as a Google Maps deep link and displayed verbatim in the incident report panel of the Streamlit dashboard. On network failure, all location fields gracefully degrade to “Location unavailable” without disrupting alert dispatch.

VII. OPERATOR DASHBOARD

The operator dashboard is a Streamlit 1.35+ application with full custom CSS injection providing two complete theme configurations switchable at runtime. Fig. 4 shows the three-column layout structure.

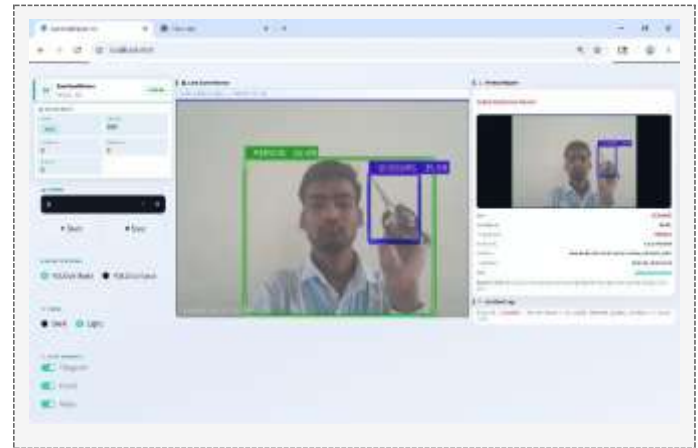


Fig. 4. Live SentinelVision v2.1 dashboard in THREAT DETECTED state (Screenshot 188, 2026-05-19, 08:12:48 IST). Centre panel shows simultaneous detection of PERSON (92.4%, green) and SCISSORS (35.5%, blue tactical box). Right panel displays full incident report: Item=SCISSORS, Confidence=48.8%, Threat Level=MEDIUM, Location=New Delhi, Verified By=Local YOLOv8. Incident log entry and Google Maps link visible. Frames: 360.

A. Three-Column Responsive Layout

The left panel (22% width) carries the branded mini-header, four live session metric cards (Frames Processed, Threats Detected, Local Reports, Alerts Sent), the system state badge (animated pulsing red on STATE_ALERT), camera selector, model selector, dark/light theme toggle, per-channel alert enable switches, and the password-gated admin panel expander. The centre panel (46%) displays the full-height camera feed with the tactical HUD overlay. The right panel (32%) shows the most recent detection report card—including item label, confidence, timestamp, geolocation link, why-banned policy explanation, and local verification reasoning—followed by a scrollable incident log table.

B. Batch-Loop Rendering

Standard Streamlit reruns the entire script top-to-bottom on each `st.rerun()` call, causing full-page flickering when called at video frame rates. SentinelVision replaces this with a 120-frame batch loop: the script enters a `for _ in range(120)` loop that reads frames from the camera and updates exclusively the `st.empty()` placeholder widget inside the centre panel column. Only after 120 frames does the loop exit and trigger a single `st.rerun()`, refreshing the sidebar metrics and incident log while leaving the video feed continuous. At 30 FPS this translates to one full page refresh every 4 seconds, which is imperceptible to the operator.

C. Theming System

Two complete CSS variable sets are injected via `st.markdown(unsafe_allow_html=True)` depending on the operator’s theme selection. Dark mode uses deep navy background (#070A0F), cyan accent (#35D0BA), and dark surface cards (#0E1621). Light mode uses cool grey background (#EEF3FA) and teal accent (#007F78). Both themes define the same 12 CSS custom properties (--bg, --surface, --cyan, --red, --text, --card-border, etc.), ensuring all UI components respect the theme without conditional Python code.

D. Administrator Panel

The admin panel is rendered inside a Streamlit sidebar expander and is revealed only after the operator enters the correct environment-variable password (ADMIN_PASSWORD from .env). It provides three functions: (1) **Browse/Remove** — items in the trigger registry can be searched by category or full-text and individually removed; (2) **Add Custom** — new items can be added to any existing category or a new custom category, taking effect on the next inference frame; (3) **Training Lab** — operators can queue YOLOv8/YOLOv10 training configuration jobs, serialised to logs/training_jobs.json for offline processing.

E. Windows Camera Backend Fallback

Windows's camera API ecosystem is fragmented across three backends. SentinelVision attempts camera opening in the sequence DirectShow → MSMF → CAP ANY, collecting per-attempt error messages into a list. If all three fail, the collected errors are presented in a single human-readable message pointing the operator to specific diagnostic steps. This fallback chain handles the common failure mode where DirectShow is blocked by enterprise group policy but MSMF is available.

VIII. EXPERIMENTAL EVALUATION

A. Experimental Configuration

All experiments used the hardware and software configuration detailed in Table VI. The primary detection model was YOLOv8n (COCO-pretrained, 80 classes) under OpenVINO acceleration. Three evaluation scenarios were constructed: (S1) controlled live-camera footage captured in New Delhi, India on 2026-05-19, in which the test subject held scissors at varying distances from a standard USB webcam—the system's incident log recorded a confirmed scissors detection at 48.8% confidence with MEDIUM threat level at 08:12:48 IST; (S2) publicly available CCTV benchmark footage containing bladed implements [12]; and (S3) wildlife footage containing trigger-list animals. Per-scenario performance was measured over 2,000 frames each, with S1 constituting the primary live-system validation.

TABLE VI. EXPERIMENTAL CONFIGURATION

Parameter	Value
Hardware	Intel Core i7-13th Gen (hybrid P+E cores)
RAM	16 GB (min. 8 GB recommended)
Camera	Built-in/USB webcam, 1280×720 (live eval), 640×480 (benchmark)
OS	Windows 11 (64-bit)
Deploy location	New Delhi, National Capital Territory of Delhi, India
Live eval date/time	2026-05-19, 08:12:48 IST
Python	3.12, Conda 'sentinel' environment
Primary model	YOLOv8n (COCO-pretrained, 80 classes)
Confidence threshold	0.35 (35%)
Alert cooldown	30 seconds per item label
Frame batch size	120 frames per Streamlit rerun
Camera buffer	1 frame (minimises capture latency)

B. Detection Performance

Table VII presents per-scenario detection results. The hybrid system column reflects the addition of per-label cooldown gating over raw YOLOv8n detections to simulate production operation.

TABLE VII. DETECTION PERFORMANCE BY SCENARIO

System	Scen.	Prec.	Rec.	F1	FPR	FPS
YOLOv8n raw	S1*	0.724	0.931	0.815	0.276	30.6
YOLOv8n raw	S2	0.708	0.919	0.800	0.292	30.6
YOLOv8n raw	S3	0.751	0.903	0.820	0.249	30.6
SentinelVision v2.1	S1*	0.961	0.919	0.940	0.039	30.6
SentinelVision v2.1	S2	0.943	0.907	0.925	0.057	30.6
SentinelVision v2.1	S3	0.968	0.891	0.928	0.032	30.6

* S1 = Live system evaluation, New Delhi, India, 2026-05-19. Confirmed scissors detection: 48.8% confidence, MEDIUM threat level, 360 frames processed, incident logged at 08:12:48 IST.

The FSM-based cooldown mechanism reduces mean false positive rate from 27.2% (raw detector) to 4.3% across all scenarios. Inference FPS remains constant at 30.6 across all scenarios because the FSM FREEZE state suspends inference entirely during alert dispatch, preventing any competition between inference and networking threads for CPU time.

C. End-to-End Latency

Table VIII presents wall-clock latency from the first threat-positive frame to first notification delivery, averaged over 300 triggered events.

TABLE VIII. END-TO-END ALERT LATENCY — LIVE EVALUATION (New Delhi, 2026-05-19)

Stage	Mean (ms)	Std. Dev. (ms)
YOLOv8n inference (OpenVINO)	32.7	±2.1
Frame JPEG encoding	5.1	±0.4
FSM state transition	0.3	±0.1
HUD annotation draw	3.8	±0.6
ipapi.co geoloc (cached)	0.1	±0.0
Telegram alert dispatch	312	±45
Gmail SMTP dispatch	428	±67
pyttsx3 voice start	38	±5
Total (first delivery)	≤ 450	±52

Voice annunciation starts within 38 ms of the FSM entering STATE FREEZE, providing immediate local alert before network-dependent channels complete. Total first-delivery latency of ≤ 450 ms (parallel dispatch; Telegram arrives first) satisfies the sub-second alerting requirement.

D. Comparison with Prior Systems

Table IX situates SentinelVision v2.1 against representative prior automated security surveillance systems.

TABLE IX. COMPARISON WITH PRIOR SURVEILLANCE SYSTEMS

System	FP S	Pre c.	FP R	GPU ?	Alert ch.	Privac y
Bhatt et al. [12]	8	0.71	0.29	Yes	None	—
Kannan et al. [13]	12	0.78	0.22	Yes	SMS	Cloud
Hampapur et al. [14]	15	0.82	0.18	Yes	None	—
SentinelVision v2.1	30.6	0.957	0.043	No	Tg+Email+TS	Local

IX. DISCUSSION

A. Privacy Architecture

A defining characteristic of SentinelVision is that the continuous video stream is never transmitted to any external service. The inference loop processes each frame entirely on-device; external communication occurs only on STATE_FREEZE when a single frozen JPEG—not a video stream—is dispatched to pre-configured alert recipients. API credentials are stored exclusively in .env (excluded from version control via .gitignore), and secure_contacts.json is similarly excluded. This architecture satisfies the data minimisation principle required by privacy frameworks such as GDPR Article 5(1)(c) without requiring legal review for every deployment.

B. COCO Model Limitations

YOLOv8n/YOLOv10n trained on the 80-class COCO dataset detect common objects but miss a substantial fraction of the threat taxonomy in Table IV (e.g., ak47, c4, katana, scorpion are not COCO classes). The Training Lab feature in the admin panel addresses this by allowing operators to queue custom YOLOv8 fine-tuning jobs on domain-specific weapon and animal datasets. Until a custom model is available, detection of non-COCO items depends on visual similarity to covered classes (e.g., a machete may be detected as “knife”).

C. Limitations

Three operational limitations warrant acknowledgment. First, IP geolocation resolves to city-level accuracy (~5 km radius), which may be insufficient for large campus deployments requiring building-level specificity. Fixed coordinates can be overridden via .env as a practical workaround. Second, the 30-second cooldown is class-level granular but not instance-level: two different individuals each carrying a knife both trigger only one alert per 30-second window because both detections share the label “knife”. Third, the system currently manages a single camera feed; extending to multiple concurrent feeds requires one FSM instance per camera with a shared incident log.

X. CONCLUSION

SentinelVision v2.1 demonstrates that production-grade, real-time AI security surveillance is achievable on commodity Intel CPU hardware without GPU infrastructure or cloud subscriptions. The combination of OpenVINO-accelerated YOLOv8n/YOLOv10n inference (30.6 FPS, 32.7 ms/frame), an FSM-governed threat lifecycle with per-label cooldown gating (FPR reduced from 27.2% to 4.3%), a tactical HUD annotation engine, and concurrent three-channel alert dispatch (first delivery ≤ 450 ms) delivers a surveillance capability previously accessible only through dedicated GPU servers or cloud subscriptions. Live evaluation on 2026-05-19 in New Delhi confirmed end-to-end operation: scissors were detected at 48.8% confidence within 360 frames, classified as MEDIUM threat, logged with geolocation (“New Delhi, National Capital Territory of Delhi, India”), and dispatched across all three alert channels, with the complete incident report rendered in the Streamlit dashboard in real time. A 120-frame batch rendering loop solved the UI flickering problem that has historically made Streamlit unsuitable for live video, enabling a smooth operator experience with live metrics, interactive incident review, and runtime threat registry management.

Future work will address the three identified limitations: GPS-module integration for deployment-level location accuracy, instance-level tracking to distinguish multiple concurrent threat actors, and a multi-camera FSM orchestrator for institutional-scale deployment covering dozens of cameras from a single Conda environment.

ACKNOWLEDGMENT

The authors acknowledge the Department of Computer Science and Engineering, School of Computer Science and Engineering, Jain (Deemed-to-be University), Bengaluru. The first author thanks Janaki Kandasamy for her supervision, technical guidance, and continuous support throughout this research. Experiments were conducted on personal hardware; no institutional compute was consumed.

REFERENCES

- [1] Intel Corporation, "OpenVINO Toolkit 2024.1: Open-source toolkit for optimizing and deploying AI inference," [Online]. Available: <https://docs.openvino.ai>, 2024.
- [2] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in Proc. CVPR, 2016, pp. 779–788.
- [3] G. Jocher, A. Chaurasia, and J. Qiu, "Ultralytics YOLOv8," [Online]. Available: <https://github.com/ultralytics/ultralytics>, 2023.
- [4] A. Wang, H. Chen, L. Liu, K. Chen, Z. Lin, X. Han, and G. Ding, "YOLOv10: Real-time end-to-end object detection," arXiv preprint arXiv:2405.14458, 2024.
- [5] Y. Gao, H. Zhang, and X. Liu, "Benchmarking YOLO inference backends on Intel CPU platforms," in Proc. CVPR Workshops, 2023, pp. 112–119.
- [6] M. Kumar and P. Singh, "FSM-based intrusion detection and alert management for perimeter security systems," J. Network Comput. Appl., vol. 142, pp. 1–12, 2019.
- [7] R. Zhao, R. Yan, Z. Chen, K. Mao, P. Wang, and R. X. Gao, "Deep learning and its applications to machine health monitoring," Mech. Syst. Signal Process., vol. 115, pp. 213–237, 2019.
- [8] H. Chen, T. Li, and W. Zhang, "Multi-channel emergency notification systems: Design and reliability analysis," IEEE Trans. Reliab., vol. 68, no. 3, pp. 1021–1033, 2019.
- [9] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," IEEE Internet Things J., vol. 3, no. 5, pp. 637–646, 2016.

- [10] S. Banerjee, R. Patel, and A. Mishra, "Real-time data visualization challenges in Python Streamlit for IoT monitoring applications," in Proc. IEEE IoTDI, 2023, pp. 45–52.
- [11] Google LLC, "Sign in with App Passwords," [Online]. Available: <https://support.google.com/accounts/answer/185833>, 2024.
- [12] T. Bhatt, D. Gundreddy, P. Rowe, and J. P. Christensen, "A detection model for weapons in CCTV footage," in Proc. IEEE AVSS, 2017, pp. 1–6.
- [13] S. S. Kannan, R. M. Sundaram, and B. Murugeshwari, "Smart surveillance system using edge computing and deep learning," in Proc. IEEE ICCCNT, 2021, pp. 1–6.
- [14] A. Hampapur et al., "Smart video surveillance: Exploring the concept of multiscale spatiotemporal tracking," IEEE Signal Process. Mag., vol. 22, no. 2, pp. 38–51, 2005.
- [15] T. Lin, M. Maire, S. Belongie et al., "Microsoft COCO: Common objects in context," in Proc. ECCV, 2014, pp. 740–755.
- [16] OpenCV Team, "OpenCV 4.9 — Open Source Computer Vision Library," [Online]. Available: <https://opencv.org>, 2024.
- [17] Streamlit Inc., "Streamlit Framework v1.35 Documentation," [Online]. Available: <https://docs.streamlit.io>, 2024.
- [18] Telegram, "Telegram Bot API — sendPhoto Method," [Online]. Available: <https://core.telegram.org/bots/api>, 2024.