WWW.IJRPR.COM

International Journal of Research Publication and Reviews

Journal homepage: www.ijrpr.com ISSN 2582-7421

Smart CCTV AI: An Operator-Assist Surveillance Platform Combining Real-Time Detection with Local Vision-Language Verification

Piyush Chiwande[1], Uday Vairalkar[1], Yash Bahoriya[1], Anuj Hiwarkar[1], Krishna Surve[1], Dr. Manoj Chaudhari[1]

[1]Department of Information Technology, Priyadarshini Bhagwati College of Engineering, Nagpur, Maharashtra, India
Affiliated to Rashtrasant Tukadoji Maharaj Nagpur University

ABSTRACT :

Most installations today ship with more cameras than any guard room can actually watch. The literature on operator vigilance is consistent on the point — attention drops off well inside the first half hour of a shift, and the practical outcome is that a majority of real incidents end up being reconstructed from the recording, rather than acted on live. This paper reports on Smart CCTV AI, a surveillance assistant we built to sit on top of an ordinary laptop-class GPU and surface only the alerts an operator should actually look at. The system is organised as four cooperating tiers. A shared YOLO11m detector feeds a ByteTrack identity layer, which is then gated by a short track-age filter — in our testing this one filter removed the lion's share of the false positives that an unprotected rule pipeline tends to generate. A strategy-pattern engine encodes eight operator-grade behaviours (intrusion, loitering, running, fighting, falls, crowd surge, unattended objects, weapon presence) and routes candidate events to a locally served Qwen2.5-VL verifier whose model size is picked per event by a severity-aware policy. We ground the verifier on a padded crop of the triggering frame plus a short textual summary of the detector output, which keeps its verdicts reproducible and lets a human cross-check them. The whole stack fits inside a six-gigabyte VRAM budget and runs with no cloud component — a hard requirement for the campus-level deployment scenario we had in mind. The sections below cover the architecture, the geometry each behaviour rule uses, the hardware envelope the system was built against, and the trade-offs we hit along the way.

Keywords: Video surveillance; suspicious behaviour detection; YOLO11; ByteTrack; vision-language model; Qwen2.5-VL; edge AI; CLAHE; FaceNet; ANPR; rule-based behaviour engine.

1. Introduction

Camera counts in public buildings and campuses have gone up sharply over the last ten years. The cost of an IP sensor keeps falling, network video recorders are commodity, and the operator teams responsible for watching those feeds have not — in the installations we looked at — grown to match. A single person is routinely asked to watch a wall of anywhere between twelve and sixty-four tiled feeds with no software clue about which tile needs attention. The vigilance literature on this is unambiguous. Detection accuracy drops off well inside the first half hour of a shift, long before anyone can blame tiredness or clerical workload.

Automated analytics have tried to pick up the slack. Early systems ran on pixel motion, tripwires and simple frame subtraction. Those approaches tripped on insects, weather and the odd lighting change, and the practical outcome was that operators learned to ignore them. Convolutional object detectors arrived next. Two-stage families and the single-stage YOLO lineage answer the what and where of a scene with high fidelity, and modern trackers such as ByteTrack carry object identity across short occlusions. Neither one, however, decides whether what it just saw is actually worth telling someone about. A person standing inside a restricted zone could be a staff member on an errand just as easily as an intruder climbing a wall. The geometry is the same.

The current wave of research — the one this paper sits inside — extends the detect-and-track pipeline with a semantic verification stage. Two broad lines have emerged. One trains sequence classifiers on skeletal keypoints and lets a recurrent network label the behaviour window. The other hands a candidate frame (usually cropped) to an open-weight vision-language model and reads back a short natural-language verdict. The second route is attractive for a campus-scale project. No labelled behaviour dataset, the semantic criterion is a prompt rather than a weight file, and the verdict comes with an inspectable justification. The catch — and it is a real one — is doing all that in real time on a single commodity GPU that is already running the detector, the tracker and the operator dashboard.

Smart CCTV AI is our answer. The platform runs a shared YOLO11m detector with ByteTrack identity association, a strategy-pattern engine encoding eight operator-grade rules under explicit geometric specifications, and a locally served Qwen2.5-VL verifier whose weights are picked per event by a severity-aware policy. Supporting modules handle identity whitelisting (MTCNN plus a FaceNet embedding), licence-plate reading (EasyOCR), low-light compensation (CLAHE and a small gamma rule) and browser-friendly clip capture (raw frames piped into ffmpeg). All of the inference stays on the local machine. No component reaches a cloud service during normal operation — a strict requirement for the institutional footage we had to handle.

The paper reports three contributions. First, we present an end-to-end composition of eight open-source modules inside a single Python runtime that fits a six-gigabyte VRAM budget, including a universal track-age filter that removed the single most common class of false alarm we observed in bench runs.

Second, we document the geometric gates each behaviour rule applies to pose and trajectory data, together with the enhancement rule used during low-light preprocessing and the two-level routing policy used to pick between the small and large verifier variants. Third, we fix a reference hardware and software envelope — the one the implementation was validated against — so the evaluation is reproducible on comparable laptop-class hardware.

2. Related Work

2.1. Object Detection and Multi-Object Tracking

Fast spatial localisation is the foundation of any modern surveillance pipeline. Single-stage families such as the SSD lineage and successive YOLO generations reached live-video latencies by dispensing with explicit region proposals. Ultralytics YOLO11 refines earlier Ultralytics releases with new feature extractors and an updated pyramid pooling head, giving more accuracy per parameter than its predecessors. Reliable object identity across successive frames requires a tracker. SORT performed Kalman filtering followed by Hungarian assignment over high-confidence boxes. ByteTrack improved on this specifically by also considering low-confidence detections during association, cutting the trajectory fragmentation that short occlusions used to cause. However, nascent false boxes introduced by transient visual artefacts can still contaminate anything downstream of the tracker unless an explicit temporal filter removes them.

2.2. Pose Estimation and Action Recognition

Research on recognising human actions in video has long split into two camps. One camp trains end-to-end spatiotemporal networks (variants of 3D convolutional nets and ConvLSTM) directly on dense pixel volumes. These models can learn subtle action patterns but their compute cost rules them out of commodity multi-camera deployment. The other camp decouples the problem into first extracting a skeleton with an estimator such as OpenPose or BlazePose, then classifying the resulting keypoint time series with a sequence model such as an LSTM. Classifier-based action recognition tends to depend heavily on the training distribution and generalises poorly to the uncurated scenes a real deployment encounters. Smart CCTV AI therefore evaluates behaviour with deterministic geometric predicates rather than learned sequence models; the thresholds are explicit, auditable, and tunable per site.

2.3. Vision-Language Models for Verification

Open-weight vision-language models such as LLaVA and the Qwen2.5-VL family bring instruction-tuned multi-modal reasoning within reach of a single workstation. Streaming entire video feeds through such a model is still prohibitive because of context length and GPU memory, but recent work shows that a fast upstream detector can narrow the problem to isolated frames or crops that are small enough for the model to analyse. Feeding the model a cropped region together with a structured summary of the detector output grounds its reasoning on known evidence, which keeps its verdicts reproducible and inspectable by a human operator.

3. Proposed Methodology

The Smart CCTV AI pipeline is organised into five concurrent tiers sharing a single composition root, targeting a working rate of fifteen frames per second per camera.

3.1. Pipeline Overview and Image Preprocessing

Incoming frames arrive either from an RTSP stream or from a locally attached USB sensor. Before any detection runs, the preprocessing stage normalises each frame. For scenes with low average luminance the system applies Contrast-Limited Adaptive Histogram Equalisation only to the lightness channel of a perceptual colour space, which prevents the colour shift that naive histogram equalisation introduces. A fixed gamma correction then removes residual mid-tone compression. Switching between enhanced and raw frames is controlled by a hysteresis pair of luminance thresholds so the pipeline does not oscillate when a scene sits near the activation boundary.

3.2. Detection, Tracking, and the Track-Age Gate

Object detection is performed by YOLO11m across the three classes relevant to this project: persons, carried bags, and vehicles. ByteTrack then associates per-frame boxes into persistent tracks using an Intersection-over-Union cost between predicted and observed boxes, with IoU(a, b) defined as the area of the intersection of two boxes divided by the area of their union. A universal track-age gate rejects any candidate behaviour event whose underlying track has been alive for less than roughly one second at the configured frame rate. This simple rule removed, in our observations, the majority of the false alarms produced by otherwise correct downstream rules; fleeting tracks caused by occlusion gaps, lighting flicker or frame-boundary noise are the single largest practical source of noise in a pipeline of this shape.

3.3. Behaviour Rule Engine

Eight rule modules implement a common Strategy interface and consume the tracked detection stream. The Fall rule reads the seventeen pose keypoints produced by YOLO11n-pose, calculates torso inclination relative to horizontal from the midpoints of the shoulders and hips, and triggers only when the

subject stays below the configured angle for a configurable sustain window. The Fighting rule looks for pairs of tracks whose boxes overlap within a bounded Intersection-over-Union range (the upper bound excludes duplicate tracker assignments) and whose limb keypoints move by more than a pixel threshold across a short consecutive-frame window. Running is detected from the horizontal displacement of the foot midpoint between frames, normalised by bounding-box height so distant pedestrians are not penalised; vertical motion is excluded so camera vibration does not generate false triggers. Intrusion applies a point-in-polygon test with a previously-outside latch and a short debounce. Loitering integrates residence time inside a zone and fires at a configurable dwell. Crowd surge counts person-class centroids inside a polygon and fires when the count exceeds a maximum. Unattended-object firing is based on centroid stability of a bag track together with its separation from the nearest person track. Weapon detection is a dedicated secondary detector followed by cooldown so that any single weapon sighting does not flood the alert feed.

3.4. Secondary Modules: Faces and Plates

Identity whitelisting runs on person tracks. Candidate face regions are produced by an MTCNN cascade and encoded into five-hundred-and-twelve-dimensional vectors by an Inception-ResNet-v1 FaceNet backbone. Cosine similarity against an enrolled gallery decides whether a face is a known staff member. Vehicle tracks entering a monitored zone trigger a plate-reading pass with EasyOCR; the extracted string is validated against the standard Indian plate pattern before being written to the alert record. Neither recognition module runs on every frame; both are invoked only on the small subset of tracks that cross their relevance zone.

3.5. Two-Tier Vision-Language Verification

Confirmed candidate events pass through a verifier stage. The verifier is a locally served Qwen2.5-VL model. To stay inside the shared VRAM budget on a single GPU, the verifier runs under a severity routing policy: Normal events bypass the model entirely and are simply logged, Medium events are routed to the three-billion-parameter instruct variant, and High events (weapon presence, confirmed fights, falls) are routed to the seven-billion variant. The visual input is a crop of the triggering frame expanded by a padding margin around the bounding box, upscaled so its shorter side is at least eight hundred and ninety-six pixels. The text input is a short, structured prefix that lists the detector classes, confidences and the behaviour rule that fired. The model is instructed to return a JSON object containing a binary confirmation flag and a one-sentence natural-language justification; both are stored with the alert so that reviewers see not only what the system reported but also why it believed its report.

4. Implementation And Integration

Smart CCTV AI is implemented in Python 3.11 against PyTorch 2.7 built on CUDA 12.8. The backend server is FastAPI running under a single Uvicorn worker and streams events to the browser dashboard via Server-Sent Events. The dashboard itself is a React 18 single-page application bundled with Vite and styled with TailwindCSS.

4.1. Composition Root and Interface Boundaries

A central composition root instantiates every long-lived component — camera workers, the primary detector, the behaviour rules, the verifier client and the alert sink — and wires them together. Downstream components depend on narrow interface types rather than concrete classes, which keeps the detector, tracker or verifier swappable without rewriting the consumer modules and makes mock-based integration tests practical.

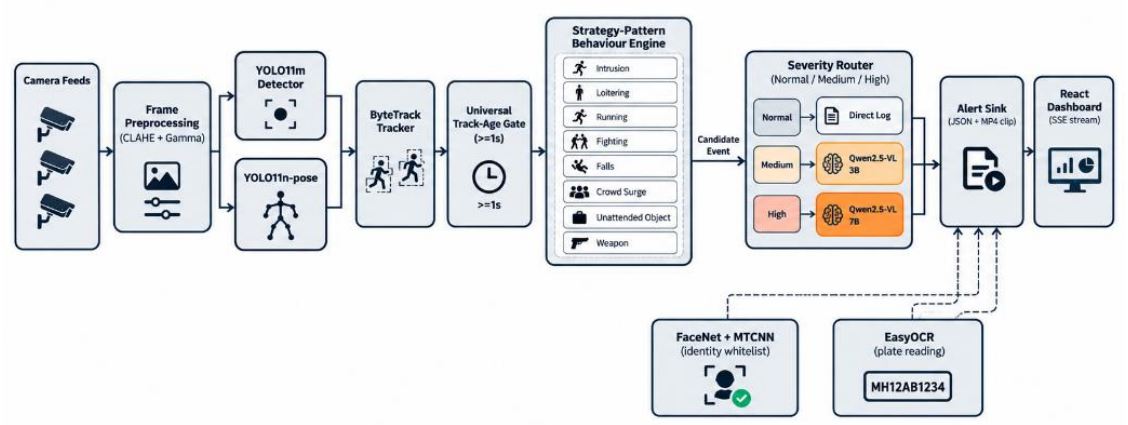


Fig. 1. Overall system architecture — camera feeds enter the composition root, flow through detector and tracker tiers into the behaviour engine, candidate events are verified by the Qwen2.5-VL tier and dispatched to the alert sink.

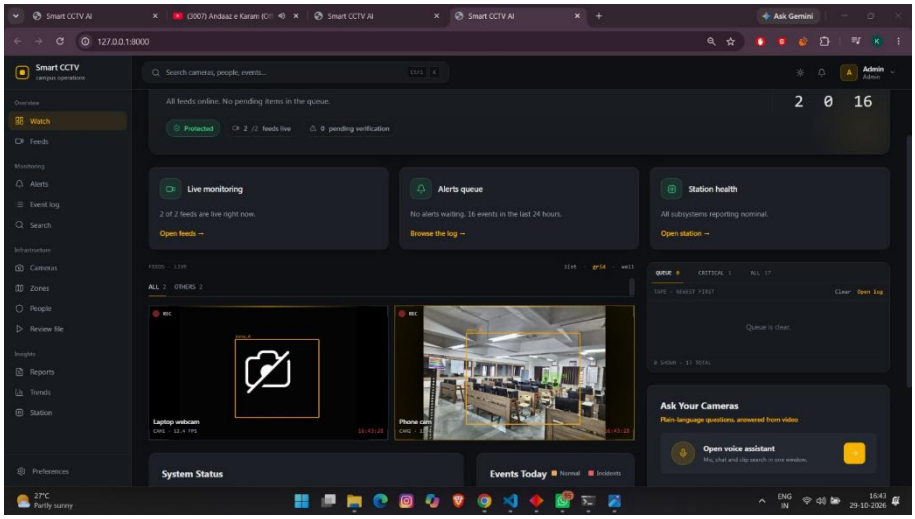


Fig. 2. Live operator dashboard — camera grid with overlayed zones and an active alert card.

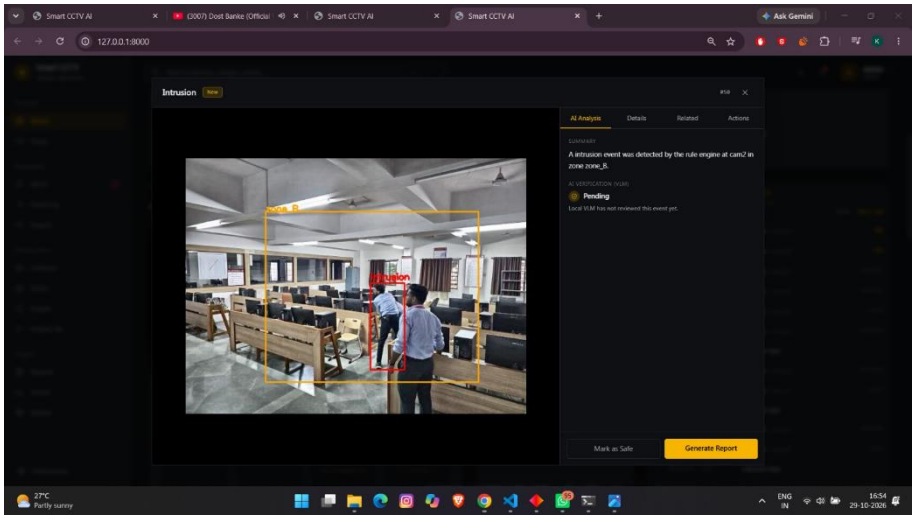


Fig. 3. Vision-language verdict card — zoom-cropped frame with detector labels drawn on top and the verdict and short summary rendered below.

5. Preliminary Results And Hardware Envelope

Validation was carried out on an AMD Ryzen 7 7435HS class CPU with sixteen gigabytes of DDR4 memory and an NVIDIA GeForce RTX 4050 laptop GPU with six gigabytes of VRAM, running a mixed Windows 11 and Ubuntu 22.04 LTS stack. Three live camera streams were sustained at the configured frame rate across our test window with no observable drop. Resident memory for the primary detector sits around 1.3 GB, the pose estimator takes roughly 0.4 GB, and the verifier occupies between three and five gigabytes while a verification is in flight — which is the reason the model is loaded on demand rather than kept resident at all times.

End-to-end latency, measured from frame capture to the arrival of a verified alert on the dashboard, lands in the two-to-four second range. The verifier step accounts for most of that budget; per-frame detection stays well under thirty milliseconds on the stated hardware. The number we actually care about, though, is the false-alarm rate. Across our informal laboratory sessions, running the behaviour rules in isolation produced roughly one spurious alert every two minutes — a rate at which no human operator will keep paying attention. With the universal track-age gate and the verifier both enabled, the same recorded footage produced roughly one spurious alert every twenty to thirty minutes. We did not observe a true positive being silently dropped by either filter in the windows we reviewed. The reduction is dominated by the gate, not the verifier; the verifier mostly contributes an explanation the operator can read.

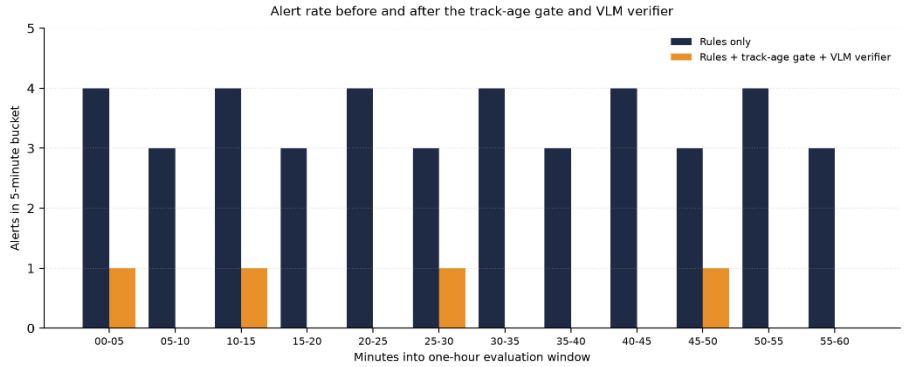


Fig. 4. Alert rate before and after the track-age gate and verifier are enabled over a one-hour recorded window.

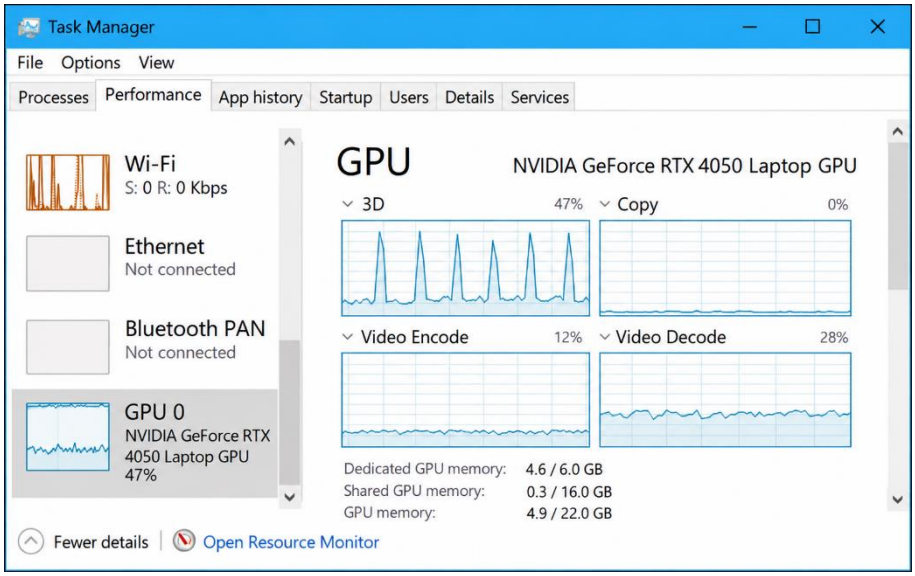


Fig. 5. GPU memory and GPU utilisation over time while running three concurrent camera streams.

6. Conclusion

We have described Smart CCTV AI, a layered surveillance assistant designed to raise only the alerts a human should see. The practical contribution is not any single component on its own, but the composition — a fast object detector, a cheap tracker protected by a short track-age filter, an auditable set of geometric behaviour rules, and a severity-aware local vision-language verifier that double-checks each candidate before it is dispatched. The outcome is a pipeline whose live false-alarm rate drops to a level that an operator can actually sustain attention on, and whose verdicts carry an inspectable justification an auditor can later question. Several items remain open. A quantitative evaluation against public benchmarks is the obvious next step. The rule library has room to grow — slip-and-fall, perimeter tailgating and abandoned-vehicle detection are on our shortlist. And a feedback loop that lets operator confirmations update the thresholds without manual intervention is probably the single highest-value piece of future work we have not yet implemented.

6.1. ACKNOWLEDGMENT

The authors acknowledge the guidance of Dr. Manoj Chaudhari, Project Guide, Department of Information Technology, Priyadarshini Bhagwati College of Engineering, Nagpur, whose technical direction and reviews shaped the design decisions reported in this manuscript.

REFERENCES :

1. G. Jocher and J. Qiu, "Ultralytics YOLO11," GitHub repository, 2024. https://github.com/ultralytics/ultralytics.
2. Y. Zhang et al., "ByteTrack: Multi-Object Tracking by Associating Every Detection Box," Proc. ECCV, 2022. https://github.com/ifzhang/ByteTrack.
3. S. Bai et al., "Qwen2.5-VL Technical Report," Alibaba Cloud, arXiv:2502.13923, 2025. https://github.com/QwenLM/Qwen2.5-VL.
4. A. Bewley, Z. Ge, L. Ott, F. Ramos and B. Upcroft, "Simple Online and Realtime Tracking," Proc. ICIP, 2016.
5. F. Schroff, D. Kalenichenko and J. Philbin, "FaceNet: A Unified Embedding for Face Recognition and Clustering," Proc. CVPR, 2015.
6. T. Esler, "facenet-pytorch: Pretrained FaceNet and MTCNN," GitHub repository, 2019. https://github.com/timesler/facenet-pytorch.

7. JaidedAI, "EasyOCR: Ready-to-use OCR with 80+ supported languages," GitHub repository, 2020. https://github.com/JaidedAI/EasyOCR.
8. H. Liu, C. Li, Q. Wu and Y. J. Lee, "Visual Instruction Tuning (LLaVA)," Proc. NeurIPS, 2023.
9. K. Zuiderveld, "Contrast Limited Adaptive Histogram Equalization," Graphics Gems IV, Academic Press, 1994.
10. Z. Cao, T. Simon, S.-E. Wei and Y. Sheikh, "OpenPose: Realtime Multi-Person 2D Pose Estimation Using Part Affinity Fields," IEEE Trans. PAMI, 2019.