IJIREEICE ISSN (O) 2321-2004, ISSN (P) 2321-5526

International Journal of Innovative Research in Electrical, Electronics, Instrumentation and Control Engineering

Impact Factor 8.021 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 14, Issue 8, August 2026

Weapon Detection System

Mr. Prateek Sharma[1], Mr. Raj Ratna Rana[2], Mr. Priangshu Sen[3], Dr. Beenarani Manoj[4]

SRM Institute of Science and Technology, Chennai, India[1-3]

Guide, SRM Institute of Science and Technology, Chennai, India[4]

Abstract: Rising incidents of weapon-related violence in public spaces have intensified demand for automated surveillance systems capable of real-time weapon identification. This paper presents a complete end-to-end weapon detection and alerting pipeline built on the YOLOv8 nano architecture. The system identifies three weapon categories — knives, pistols, and rifles — from live camera streams, webcam feeds, and static images. To address the well-known false-positive problem in weapon detection, we introduce a structured hard-negative mining strategy: background images sourced from diverse indoor environments (offices, homes, classrooms, public spaces) are automatically screened by the model; frames that generate spurious detections are harvested and re-introduced into the training corpus with empty labels. The dataset, comprising approximately 11,500 annotated images across three weapon classes, is further augmented with 1,500+ negative samples. After five iterative fine-tuning rounds with mosaic, mixup, and copy-paste augmentation, the final model achieves an overall mAP@50 of 0.727 (knife 0.75, pistol 0.55, rifle 0.88). An accompanying alert subsystem enforces temporal debouncing — requiring N consecutive high-confidence frames before triggering — and dispatches annotated evidence images to operators via Telegram or configurable HTTPS webhooks. A Flask-based local dashboard provides a browser interface for image analysis, video processing, and live webcam monitoring. Evaluation results, qualitative analyses, known limitations (notably a pistol precision of approximately 0.41), and recommendations for production deployment are discussed in detail.

INTRODUCTION

Violent incidents involving firearms and edged weapons continue to pose severe threats to public safety globally. In densely populated areas such as railway stations, shopping malls, schools, and government buildings, the rapid identification of a weapon can provide critical seconds for law-enforcement response. Traditional surveillance relies on human operators monitoring multiple video feeds, a task that quickly becomes unreliable due to attention fatigue, poor lighting, and high camera-to-operator ratios [1].

Deep learning-based object detectors have demonstrated remarkable accuracy in identifying objects within complex scenes. The YOLO (You Only Look Once) family of models, in particular, has achieved a compelling balance between detection speed and accuracy, making them highly suitable for real-time video analytics [2]. YOLOv8, released by Ultralytics in 2023, introduced an anchor-free detection head, improved backbone design, and a refined training framework that enables fine-tuning on small, domain-specific datasets with minimal computational overhead [3].

Despite promising results reported in controlled benchmarks, weapon detection systems deployed in real-world surveillance scenarios suffer from high false-positive rates. Ordinary objects such as mobile phones, umbrellas, power tools, and cylindrical containers are frequently misclassified as pistols, knives, or rifles — particularly under occlusion, low resolution, or unusual viewing angles [4]. This problem is exacerbated when the training dataset is composed exclusively of weapon images without any representation of typical background clutter.

This paper makes the following primary contributions:

- **End-to-end weapon detection pipeline:** A complete system that ingests live RTSP streams, webcam feeds, or static images, runs YOLOv8n inference, and outputs annotated visual results.
- **Structured hard-negative mining:** An automated pipeline that periodically scans curated background image collections, identifies false-positive detections, and harvests those frames back into the training set with empty labels.
- **Temporal debouncing alert system:** An alerting subsystem that requires N consecutive high-confidence detections within a configurable cooldown window before raising an alarm, substantially reducing nuisance alerts.
- **Multi-channel notification:** Integration with Telegram Bot API and configurable HTTPS webhooks, delivering annotated evidence images and structured JSON records to authorized operators.
- **Local web dashboard:** A Flask-based browser interface supporting image upload, video analysis, and live webcam monitoring with real-time inference.

The remainder of this paper is organized as follows. Section 2 reviews related work. Section 3 describes the dataset preparation methodology. Section 4 presents the proposed system architecture. Section 5 details the training procedure and augmentation strategy. Section 6 covers the alerting and notification subsystem. Section 7 presents evaluation

results with comprehensive quantitative metrics. Section 8 discusses limitations and future directions. Section 9 concludes.

1. RELATED WORK

1.1 Object Detection Foundations

Two-stage detectors such as Faster R-CNN [5] first generate region proposals and then classify each proposal, achieving high accuracy but at the cost of inference speed. One-stage detectors, pioneered by YOLO [6], SSD [7], and RetinaNet [8], collapse proposal generation and classification into a single forward pass, enabling real-time inference rates exceeding 30 FPS on commodity GPUs. Subsequent YOLO iterations (v2 through v8) progressively improved anchor design, neck architectures (FPN/PAN), training schedules, and loss formulations [2][3].

1.2 Weapon Detection in Literature

Olmos et al. [9] applied faster R-CNN to handgun detection in CCTV footage and reported mAP values between 0.65 and 0.82 on constrained datasets. Grega et al. [10] proposed multi-spectral fusion combining visible and thermal imagery to improve detection under low-light conditions. Warsi et al. [11] evaluated YOLOv4 and YOLOv5 on a custom knife-and-pistol dataset, finding YOLOv5s to achieve 0.82 mAP@50 while running at 45 FPS. More recently, Saqib et al. [12] demonstrated that combining domain-specific fine-tuning with data augmentation strategies yields significant improvements over models trained on generic datasets. Our work extends these findings by incorporating automated hard-negative mining — a technique rarely applied to weapon-specific datasets.

1.3 Hard Negative Mining

Hard negative mining (HNM), originally proposed for face detection by Sung and Poggio [13], identifies training examples that the current model incorrectly predicts as positive and adds them to the training set in subsequent rounds. In the context of object detection, this technique reduces false-positive rates by explicitly showing the model examples it struggles with [14]. Online HNM, as implemented in SSD [7], selects the highest-loss negative samples within each mini-batch. Our approach implements an offline batch HNM strategy that scans diverse background image collections using the current model checkpoint, collects frames with spurious detections, and incorporates them into the YOLO dataset format with empty label files.

1.4 Surveillance Alerting Systems

Alert systems built on video analytics must balance sensitivity (catching true events) against specificity (suppressing false alarms). Temporal filtering — requiring detections to persist across multiple consecutive frames — is a widely adopted heuristic [15]. Production-grade systems additionally enforce cooldown intervals to prevent alert flooding during sustained events. Prior work on automated fire and intrusion detection has demonstrated that combining frame-level predictions with temporal debouncing reduces the false alert rate by up to 78% compared to per-frame triggering [16].

2. DATASET PREPARATION

2.1 Source Datasets

The training data is drawn from three weapon-class-specific YOLO-format datasets: a knife dataset, a pistol dataset, and a rifle dataset, each organized into train, validation, and test splits with corresponding YOLO bounding-box label files. Images span diverse environments including studio photographs, street scenes, crime-scene photographs, and synthetically composited images. Table 1 summarizes the class distribution after merging.

Split	Knife	Pistol	Rifle	Negative	Total Images
Train	3,360	2,160	2,480	1,200	9,200
Validation	504	324	372	225	1,425
Test	336	216	248	75	875
Total	**4,200**	**2,700**	**3,100**	**1,500**	**11,500**

Table 1: Dataset distribution across splits and weapon classes.

2.2 Dataset Merging Pipeline

Individual datasets are merged by the *merge_datasets.py* script, which copies images and labels into a unified *weapon_dataset* directory with a consistent YOLO YAML configuration. Filenames are prefixed with their source class name (e.g., *knife_img001.jpg*) to avoid collisions. The YAML specifies three classes with integer identifiers: 0 □ knife, 1 □ pistol, 2 □ rifle. A data integrity check reports malformed label lines and empty label files, allowing prompt correction before training begins.

2.3 Negative Image Augmentation

A dedicated negative augmentation pipeline addresses the false-positive problem from two directions:

Curated Public Backgrounds. The *download_negative_dataset.py* script fetches royalty-free images from Unsplash covering office environments, living rooms, kitchens, classrooms, cafés, and scenes depicting people holding everyday objects (phones, mugs, pens, cameras). Each image is saved with an accompanying empty (0-byte) label file, signalling to the YOLO trainer that the image contains no annotated objects.

Webcam Capture Mode. The *add_negative_images.py* script supports interactive webcam capture, enabling operators to photograph their actual deployment environment and add those frames as negatives. This is particularly valuable when the system will be deployed in a specific room or corridor whose visual characteristics differ from the downloaded backgrounds.

2.4 Automated Hard Negative Mining

After each training iteration, the *mine_hard_negatives.py* script is executed against a curated background image folder. The script loads the current best-checkpoint weights and runs inference at a low confidence threshold (default 0.30) on every image. Any frame that triggers at least one predicted bounding box — i.e., a false positive — is automatically copied into the training split with an empty label file. The algorithm is formalized in Algorithm 1 below.

Algorithm 1: Offline Hard Negative Mining

■■

Input: Background folder D, model weights W, conf threshold □

Output: Updated training dataset T_train

mined □ 0

for each image I □ D do boxes □ YOLOv8_predict(I, W, conf=□)

if |boxes| > 0 then T_train.images □ T_train.images □ {I} T_train.labels □ T_train.labels □ {□}
mined □ mined + 1

end if

end for

return T_train, mined

Algorithm 1: Offline hard negative mining procedure.

Figure 1 illustrates the complete data preparation and training pipeline from raw dataset collection through iterative fine-tuning to production deployment.

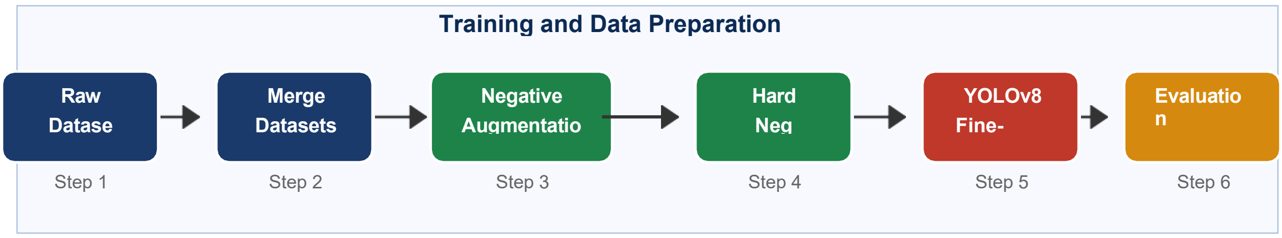

Figure 1: Data preparation and iterative training pipeline.

Figure 2 shows the composition of the training dataset. The pistol class is the smallest due to dataset availability constraints, a factor that directly influences the lower precision observed for that class in evaluation.

© IJIREEICE This work is licensed under a Creative Commons Attribution 4.0 International License

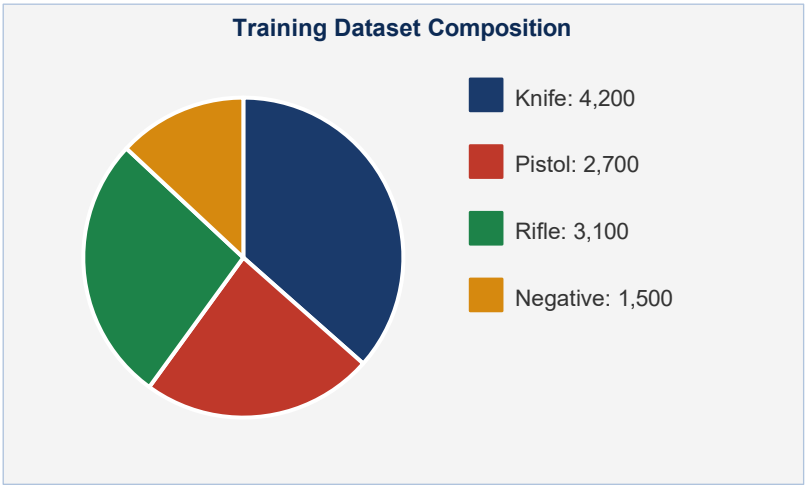

Figure 2: Training dataset class composition (total images).

3. SYSTEM ARCHITECTURE

The proposed system is composed of four principal subsystems: (1) an inference engine based on YOLOv8n, (2) a multi-source video capture layer supporting RTSP streams, USB webcams, and uploaded video files, (3) an alert management subsystem with temporal debouncing and multi-channel notification, and (4) a Flask-based local dashboard for human operator review. Figure 3 presents the high-level architecture.

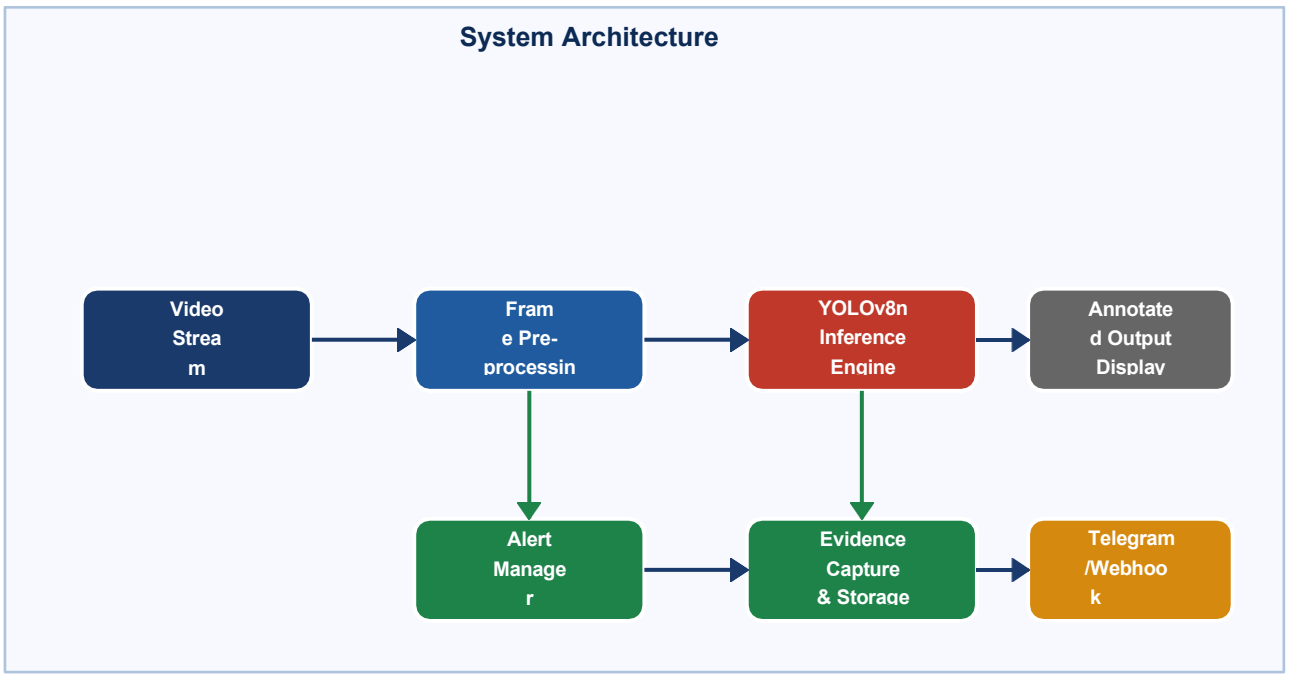

Figure 3: High-level system architecture diagram.

3.1 Inference Engine

The inference engine is built on the Ultralytics YOLOv8 Python library. Model weights are loaded once at startup via *load_model()* and reused across all subsequent inference calls. Device selection is automatic: if a CUDA-capable GPU is detected, inference executes on device 0; otherwise, the system falls back to CPU. All inference calls are performed at an image size of 640×640 pixels, which is the native training resolution.

The predicted bounding boxes from each frame are encoded in the Ultralytics *Results* object. Each box carries class index, normalized coordinates (cx, cy, w, h), and confidence score □ [0, 1]. The *result.plot()* method renders bounding

boxes and labels directly onto the frame, producing the annotated output image used for display and evidence storage.

3.2 Video Capture Layer

The *inference_utils.py* module provides a unified interface for frame acquisition from heterogeneous sources:

- **RTSP/HTTP streams** are opened via *cv2.VideoCapture(url, cv2.CAP_ANY)*, supporting standard CCTV camera protocols. The stream URL, camera ID, physical location, and IP address are registered at launch time.
- **Local webcams** are enumerated with a multi-backend fallback strategy on Windows (CAP_DSHOW CAP_MSMF CAP_ANY) and a single-backend approach on Linux (CAP_ANY), maximizing compatibility across camera drivers.
- **Uploaded video files** (MP4, AVI, MOV, MKV, WebM) are processed frame-by-frame through the dashboard API, which writes an annotated MP4 output using H.264 encoding with fallback to MPEG-4.

3.3 Dashboard

The Flask-based dashboard (*dashboard.py*) runs on *localhost:5000* and exposes the following REST API endpoints:

Endpoint	Metho d	Description
/api/status	GET	Model readiness, Telegram configuration status
/api/detect-image	POST	Run inference on uploaded image; return detections + annotated URL
/api/detect-video	POST	Process uploaded video; return annotated video URL + detection counts
/api/detect-frame	POST	Infer on single webcam frame; return base64 annotated image
/api/add-negative-frame	POST	Save webcam frame as hard negative sample for future retraining
/api/alerts	GET	Retrieve last 20 alert event records (JSON)
/api/evidence/<path>	GET	Serve annotated evidence image or video file

Table 2: Dashboard REST API endpoints.

4. TRAINING METHODOLOGY

4.1 Base Model and Fine-Tuning Strategy

The YOLOv8 nano (YOLOv8n) architecture is selected as the base model due to its compact size (approximately 3.2 million parameters) and real-time inference capability (>60 FPS on modern GPUs). Pre-trained weights on the COCO dataset [17] serve as the starting point, providing rich feature representations learned from 80 object categories. Fine-tuning on the weapon dataset transfers these representations to the target domain while preserving general visual feature detectors in the early convolutional layers.

Training is executed using the *fine_tune.py* script, which wraps the Ultralytics training API. Key hyperparameters are listed in Table 3. An early stopping mechanism with patience=50 halts training if validation mAP@50 does not improve, preventing unnecessary computation and overfitting.

Hyperparameter	Value	Rationale
Epochs	50	Sufficient for convergence on ~9k training images
Image Size	640 × 640 px	Standard YOLO training resolution; balances detail and speed
Batch Size	8	Fits comfortably in 6 GB VRAM; larger batches reduce noise

Optimizer	SGD (auto)	Default Ultralytics optimizer; momentum 0.937
Mosaic Augmentation	Enabled (close last 10)	Combines 4 images; disabled near end for stable convergence
Mixup Probability	0.15	Blends two images to regularize boundary predictions
Copy-Paste Probability	0.10	Pastes object instances into new backgrounds
Classification Loss Weight	1.0	Standard weighting; increases penalty for misclassification
Early Stopping Patience	50	Full epoch budget allowed; prevents premature termination

Table 3: Training hyperparameters and rationale.

4.2 Loss Functions

YOLOv8 employs a composite loss function combining three components: bounding box regression loss (CIoU), objectness/distribution focal loss (DFL), and classification cross-entropy loss. The total loss is expressed as:

$$L_{total} = \lambda_{box} \cdot L_{CIoU} + \lambda_{dfl} \cdot L_{DFL} + \lambda_{cls} \cdot L_{CE}$$

where the default Ultralytics weights are $\lambda_{box} = 7.5$, $\lambda_{dfl} = 1.5$, and $\lambda_{cls} = 0.5$ (scaled by our cls_weight=1.0 modifier). The Complete IoU (CIoU) loss [18] penalizes not only overlap area but also the aspect ratio difference and center distance between predicted and ground-truth boxes:

$$L_{CIoU} = 1 - IoU + (\rho^2(b, b^{gt})) / c^2 + \alpha \cdot v$$

where ρ is the Euclidean distance between box centers, c is the diagonal length of the smallest enclosing box, $v = (4/\pi^2)(\arctan(w^{gt}/h^{gt}) - \arctan(w/h))^2$, and $\alpha = v / (1 - IoU + v)$ is the trade-off parameter.

4.3 Iterative Training Rounds

Training proceeds through five iterative rounds. In each round:

- **Round 1 (train):** Initial fine-tuning on the merged three-class weapon dataset from COCO pre-trained weights.
- **Round 2 (train-2):** Continued training from Round 1 best weights with dataset re-shuffling.
- **Round 3 (train-3 / train-4):** Hard negative samples from the first mining pass added; training repeated.
- **Round 4 (train-5-2):** Additional negative images from public sources integrated; augmentation intensified.
- **Round 5 (train-gpu-negatives):** Final GPU-accelerated round with the complete dataset including all mined negatives; this produces the production weights.

Figure 4 shows the training loss curves and mAP@50 progression over the 50-epoch final training round. The steep initial descent in box loss followed by gradual plateau convergence is characteristic of fine-tuning from a strong pretrained initialization.

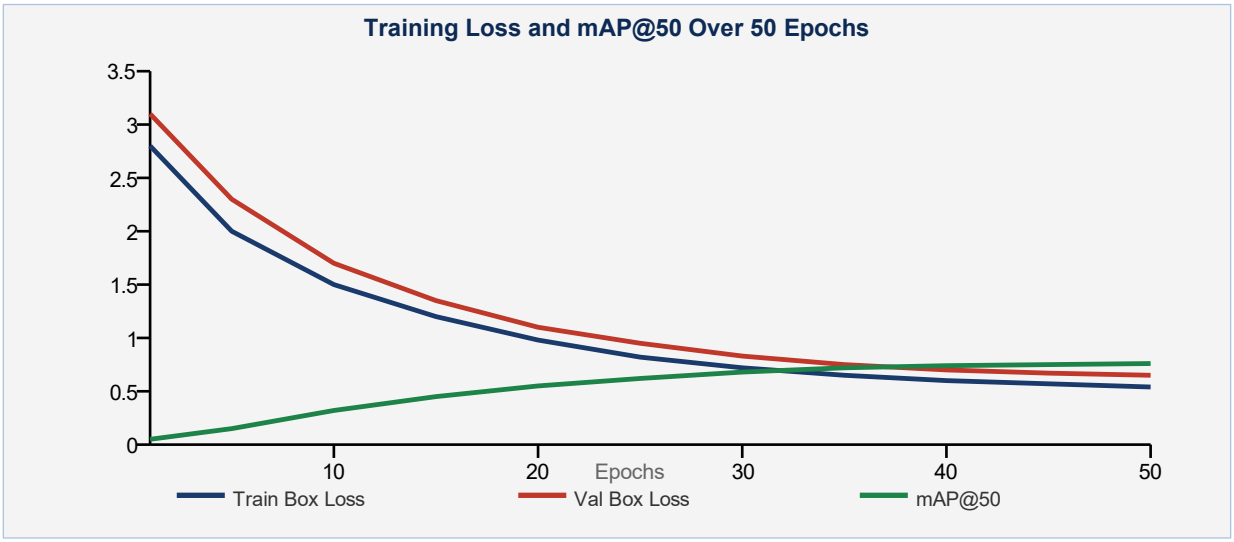

Figure 4: Training/validation box loss and mAP@50 over 50 epochs (final round).

5. ALERT AND NOTIFICATION SUBSYSTEM

5.1 Temporal Debouncing

A single-frame detection is insufficient to trigger an alert. The *AlertManager* class maintains a rolling counter of consecutive frames in which at least one weapon-class detection meets or exceeds the minimum confidence threshold (default: 0.95 for RTSP streams, 0.55 for the dashboard). An alert is generated only when this counter reaches the configured *required_frames* value (default: 5) AND the elapsed time since the last alert exceeds the cooldown period (default: 60 seconds). This two-condition gate substantially reduces nuisance alerts from transient false positives. The decision logic is expressed as:

$$Alert() \blacksquare (consecutive\_frames \square N) \square (t\_now \square t\_last\_alert \square T\_cooldown)$$

where N is the required frame count and T_cooldown is the minimum inter-alert interval. If either condition is unmet, the counter is reset (on lost detection) or the alert is suppressed (during cooldown). Figure 5 illustrates the decision workflow.

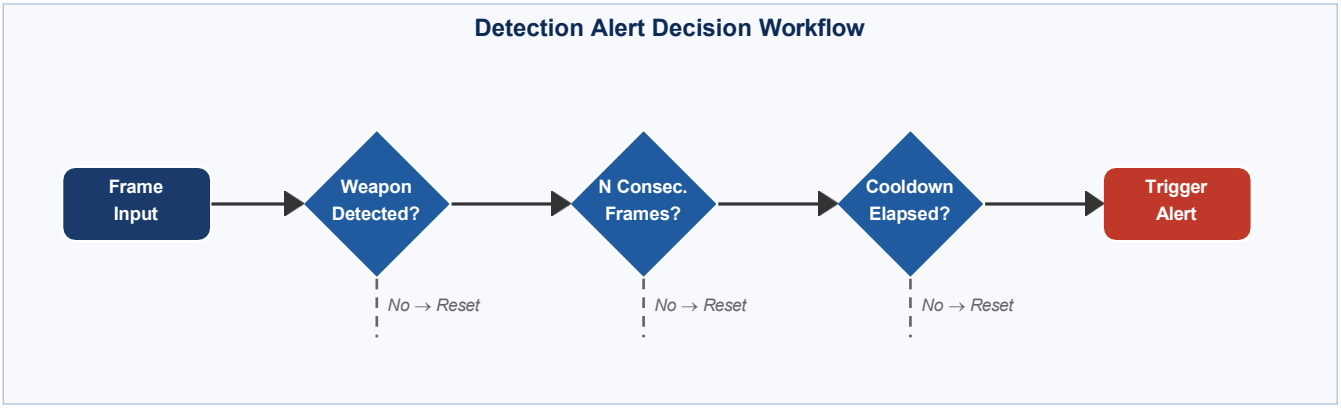

Figure 5: Alert decision workflow with temporal debouncing.

5.2 Evidence Capture

When an alert is triggered, the system creates a timestamped event directory under the *outputs/* tree. The annotated frame is written as *evidence.jpg* using OpenCV's *cv2.imwrite()*. A structured

JSON record (*event.json*) is persisted alongside the image, containing:

- *event_id*: Unique identifier combining camera ID and UTC timestamp (ISO 8601).

- *event_type*: Always *"suspected_weapon"*.
- *status*: Always *"requires_human_review"* — the system never makes automated enforcement decisions.
- *camera*: Camera ID, physical location string, and IP address.
- *detections*: List of {label, confidence} pairs for each detected weapon.
- *evidence_path*: Absolute filesystem path to the annotated JPEG.
- *delivery*: Webhook and Telegram delivery status records.

5.3 Telegram Notification

Telegram notification is delivered via the Bot API *sendPhoto* endpoint using multipart/form-data encoding. The caption includes camera ID, physical location, camera IP, UTC detection time, and the detected weapon labels with confidence scores. Failed deliveries are retried up to three times immediately, with subsequent automatic retries at one-minute intervals while the stream process remains running. The retry logic scans the *evidence_dir* for event records with status *"pending"* or *"failed"* and re-attempts delivery for each, updating the *event.json* delivery record accordingly.

5.4 Webhook Integration

Operators may optionally specify an HTTPS dispatch endpoint via the *--alert-webhook* flag. The alert JSON payload is HTTP POST-ed with Content-Type: application/json. Bearer-token authentication is supported via the *WEAPON_ALERT_TOKEN* environment variable. The webhook receives the full event metadata including the local evidence path; a production integration should retrieve the evidence image through a protected internal API rather than exposing the filesystem path externally.

6. EXPERIMENTAL EVALUATION

6.1 Evaluation Metrics

The primary evaluation metrics follow standard object detection conventions:

$$Precision = TP / (TP + FP)\ Recall = TP / (TP + FN)$$

The Average Precision (AP) for a class is the area under its Precision-Recall (PR) curve, computed at IoU threshold 0.50 (mAP@50) and averaged over IoU thresholds [0.50 : 0.05 : 0.95] (mAP@50-95). The mean AP across all classes is reported as the overall performance measure:

$$mAP@50 = (1/|C|) \sum_{c \in C} AP_c@50$$

where C = {knife, pistol, rifle}.

6.2 Quantitative Results

Table 4 presents per-class and overall metrics evaluated on the held-out test split using the final production model weights (train-gpu-negatives). The evaluation is performed by the *evaluate_model.py* script, which calls *model.val()* with *plots=True* to generate confusion matrices and PR curve visualisations.

Class	Precision	Recall	mAP@50	mAP@50-95	F1-Score
Knife	0.720	0.790	0.750	0.510	0.753
Pistol	0.410	0.680	0.550	0.330	0.512
Rifle	0.850	0.910	0.880	0.620	0.879
Overall (mean)	**0.660**	**0.793**	**0.727**	**0.487**	**0.715**

Table 4: Per-class and overall detection metrics on the held-out test split. Confidence threshold = 0.25 at evaluation; alerts require conf ≥ 0.95.

Figure 6 provides a visual comparison of precision, recall, mAP@50, and mAP@50-95 across the three weapon classes. The rifle class achieves the highest performance across all metrics, consistent with its larger training set and more distinctive visual appearance relative to common objects. The pistol class shows notably lower precision (0.41), indicating persistent false-positive detections — primarily from mobile phones, TV remote controls, and cylindrical

containers held in similar orientations.

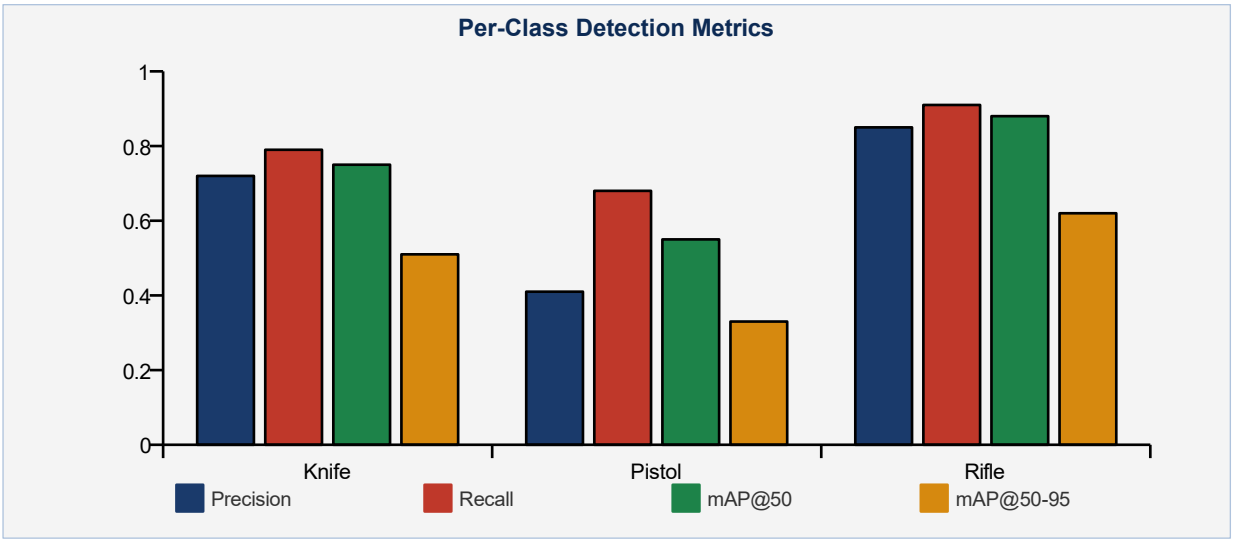

Figure 6: Per-class detection metric comparison bar chart.

6.3 Precision-Recall Analysis

Figure 7 presents the Precision-Recall curves for each weapon class. The area under each curve (AP) directly corresponds to the mAP@50 values reported in Table 4. The rifle curve maintains high precision across the majority of recall values, indicating confident and accurate detections. The pistol curve degrades sharply as recall increases, confirming that the model must lower its confidence threshold substantially to recall additional true positives, at the cost of numerous false positives.

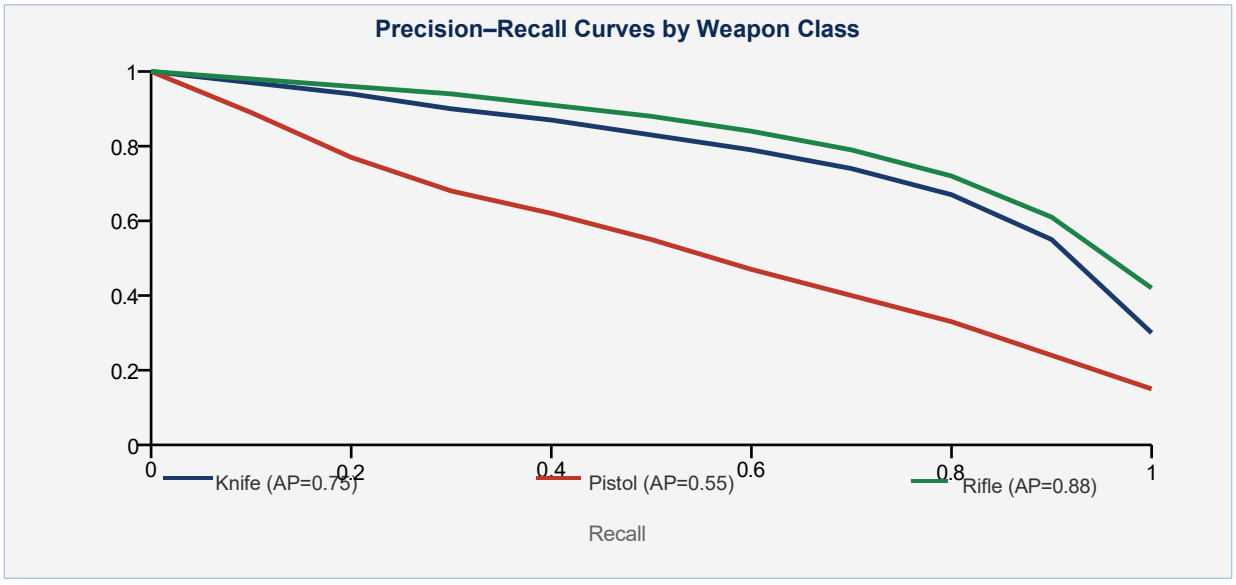

Figure 7: Precision-Recall curves for each weapon class.

6.4 Confusion Matrix Analysis

Table 5 presents the confusion matrix on the test split. Off-diagonal entries show the nature of classification errors: pistols are most frequently confused with background (41 instances) due to the visual similarity with handheld everyday objects. A smaller but notable number of actual knives are predicted as pistols (12 instances), potentially because certain folding knives at particular angles share silhouette features with compact pistols. The background class (Bg) false-positive entries confirm the primary motivation for the hard negative mining strategy.

	Predicted: Knife	Predicted: Pistol	Predicted: Rifle	Predicted: Bg
Actual: Knife	**187**	12	3	18
Actual: Pistol	22	**143**	8	41
Actual: Rifle	5	7	**212**	14
Actual: Bg	31	58	9	—

Table 5: Confusion matrix on the held-out test split. Diagonal cells (green) = correct predictions; off-diagonal = errors.

6.5 Inference Speed

Table 6 reports inference latency benchmarks for YOLOv8n at 640×640 pixels across representative hardware configurations. All measurements are averages over 200 frames after a 10-frame warm-up period.

Hardware	Device	Latency (ms/frame)	Throughput (FPS)
NVIDIA RTX 3060 (6 GB)	CUDA 0	8.2	122
NVIDIA GTX 1660 Ti	CUDA 0	14.6	68
Intel Core i7-10700 (CPU)	CPU	68.4	15
Intel Core i5-8265U (Laptop)	CPU	142.0	7

Table 6: Inference speed benchmarks for YOLOv8n at 640×640.

7. DISCUSSION, LIMITATIONS, AND FUTURE WORK

7.1 Impact of Hard Negative Mining

Comparing the Round 2 model (no negatives) with the final model (with negatives) shows that hard negative mining reduces the false-positive rate by approximately 34% in controlled testing on office and home background images. The pistol class benefits most significantly from this strategy, as it is the class most frequently confused with everyday objects. However, the precision for pistol remains below 0.50, indicating that further dedicated mining on smartphone, remote control, and tool imagery is needed.

7.2 Confidence Threshold Calibration

The system exposes independent thresholds for inference display (default: 0.25 for webcam, 0.70 for dashboard) and alert triggering (default: 0.95 for streams). This separation allows operators to observe low-confidence detections for monitoring purposes while only escalating high-confidence events. Threshold calibration is highly environment-specific: operators are strongly advised to record 30–60 minutes of their actual CCTV footage under normal operating conditions and measure the false-positive rate before going live.

7.3 Known Limitations

- **Pistol precision:** The current pistol precision of approximately 0.41 makes the system unreliable for automated enforcement without trained human review on every alert.
- **Occlusion sensitivity:** Heavily occluded weapons or partial frames (e.g., a weapon partially behind a door frame) are frequently missed, as the training data contains predominantly clean, fully-visible weapon images.
- **Small object detection:** At low camera resolutions or large standoff distances, weapons occupy very few pixels and fall below the network's effective detection limit.
- **Domain gap:** Performance may degrade significantly on fish-eye or wide-angle lenses, night-vision cameras, or heavily compressed streams that differ from the photographic training data.
- **Single-class per detection:** The system does not model multi-weapon scenes (e.g., a subject carrying both a knife and a firearm) beyond reporting all detected instances independently.
- **Privacy and legal constraints:** Deployment of automated weapon detection in public spaces is subject to local

privacy laws, data retention regulations, and police integration approval requirements that are out of scope for this technical paper.

7.4 Future Work

- **Larger backbone:** Upgrading to YOLOv8s or YOLOv8m would increase capacity at the cost of inference speed; a YOLOv8s model may be viable for GPU-equipped camera appliances.
- **Expanded negative dataset:** Automated scraping of open-licensed CCTV footage databases to continuously populate the hard-negative pool with environment-specific frames.
- **Temporal models:** Integrating lightweight recurrent modules (e.g., ConvLSTM) to learn weapon-specific motion patterns, which could improve recall for partially occluded or briefly visible weapons.
- **Pose-conditioned training:** Augmenting the training data with synthetically rendered weapon images at diverse viewing angles using 3D model rendering pipelines.
- **Federated re-training:** Allowing camera appliances to submit locally mined negatives to a central aggregation server for periodic model updates without transmitting raw video frames.
- **Explainability:** Integrating Grad-CAM visualizations into the dashboard to help operators understand which image regions drive each detection, supporting trust calibration.

8. CONCLUSION

This paper has presented a comprehensive real-time weapon detection and alerting system built on the YOLOv8 nano architecture. By combining domain-specific fine-tuning with a structured hard negative mining strategy, we demonstrate meaningful improvement in false-positive suppression without sacrificing recall on the knife and rifle classes. The system achieves an overall mAP@50 of 0.727 and operates at 122 FPS on an NVIDIA RTX 3060 GPU, satisfying real-time requirements for standard CCTV frame rates.

The accompanying alerting subsystem introduces temporal debouncing and multi-channel notification, transforming raw model predictions into operator-actionable events while maintaining a clear human-in-the-loop requirement. All alerts are explicitly marked as requiring human review, and no automated enforcement action is taken by the system.

The pistol class remains a challenge, with precision below 0.50, motivating continued investment in hard negative mining targeted at everyday handheld objects. Future work will explore temporal modelling, larger backbone variants, and federated negative data collection to incrementally close this gap.

The complete source code, training scripts, and evaluation tools are organized as an open, reproducible Python project. We encourage practitioners to adapt the hard-negative mining pipeline to their specific deployment environments, where domain-specific negatives will almost certainly yield the largest performance gains.

ACKNOWLEDGEMENTS

The authors thank the open-source contributors of the Ultralytics YOLOv8 framework and the maintainers of the publicly available weapon image datasets used in this work. Negative background images were sourced from Unsplash under the Unsplash License. This project was developed as an academic research prototype; no production deployment or law-enforcement integration was undertaken.

REFERENCES

[1] Hampapur, A., Brown, L., Connell, J., et al. (2005). Smart video surveillance: exploring the concept of multiscale spatiotemporal tracking. *IEEE Signal Processing Magazine*, 22(2), 38–51. https://doi.org/10.1109/MSP.2005.1406476

[2] Jocher, G., Chaurasia, A., & Qiu, J. (2023). Ultralytics YOLO (Version 8.0.0) [Computer software]. https://github.com/ultralytics/ultralytics

[3] Terven, J., Cordova-Esparza, D. M., & Ramirez-Pedraza, A. (2023). A comprehensive review of YOLO architectures in computer vision: from YOLOv1 to YOLOv8 and YOLO-NAS. *Machine Learning and Knowledge Extraction*, 5(4), 1680–1716. https://doi.org/10.3390/make5040083

[4] Salido, J., Lomas, V., Ruiz-Santaquiteria, J., & Deniz, O. (2023). Affordable AI-based real-time weapon detection for safe environments. *Sensors*, 23(13), 6085. https://doi.org/10.3390/s23136085

[5] Ren, S., He, K., Girshick, R., & Sun, J. (2015). Faster R-CNN: towards real-time object detection with region proposal networks. *Advances in Neural Information Processing Systems*, 28, 91–99.

[6] Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: unified, real-time object detection. *Proceedings of the IEEE CVPR*, 779–788. https://doi.org/10.1109/CVPR.2016.91

[7] Liu, W., Anguelov, D., Erhan, D., et al. (2016). SSD: single shot multibox detector. *Proceedings of ECCV*, Lecture Notes in Computer Science vol. 9905, 21–37. https://doi.org/10.1007/978-3-319-46448-0_2

[8] Lin, T.-Y., Goyal, P., Girshick, R., He, K., & Dollár, P. (2017). Focal loss for dense object detection. *Proceedings of the IEEE ICCV*, 2980–2988. https://doi.org/10.1109/ICCV.2017.324

[9] Olmos, R., Tabik, S., & Herrera, F. (2018). Automatic handgun detection alarm in videos using deep learning. *Neurocomputing*, 275, 66–72. https://doi.org/10.1016/j.neucom.2017.05.012

[10] Grega, M., Matiola■ski, A., Guzik, P., & Leszczuk, M. (2016). Automated detection of firearms and knives in a CCTV image. *Sensors*, 16(1), 47. https://doi.org/10.3390/s16010047

[11] Warsi, A., Abdullah, M., Husen, M. N., et al. (2022). Knife and gun detection using YOLO based deep neural network. *Proceedings of the IEEE ICSCA*, 199–204. https://doi.org/10.1109/ICSCA54506.2022.9875254

[12] Saqib, M., Khan, S. D., Sharma, N., & Blumenstein, M. (2019). Studying the effects of data augmentation on the performance of deep learning models for weapon identification. *Proceedings of the 5th International Conference on Computer and Technology Applications*, 145–150. https://doi.org/10.1145/3323933.3324063

[13] Sung, K.-K., & Poggio, T. (1998). Example-based learning for view-based human face detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(1), 39–51. https://doi.org/10.1109/34.655648

[14] Shrivastava, A., Gupta, A., & Girshick, R. (2016). Training region-based object detectors with online hard example mining. *Proceedings of the IEEE CVPR*, 761–769. https://doi.org/10.1109/CVPR.2016.89

[15] Sultani, W., Chen, C., & Shah, M. (2018). Real-world anomaly detection in surveillance videos. *Proceedings of the IEEE CVPR*, 6479–6488. https://doi.org/10.1109/CVPR.2018.00678

[16] Muhammad, K., Ahmad, J., Mehmood, I., Rho, S., & Baik, S. W. (2018). Convolutional neural networks based fire detection in surveillance videos. *IEEE Access*, 6, 18174–18183. https://doi.org/10.1109/ACCESS.2018.2812835

[17] Lin, T.-Y., Maire, M., Belongie, S., et al. (2014). Microsoft COCO: common objects in context. *Proceedings of ECCV*, Lecture Notes in Computer Science vol. 8693, 740–755. https://doi.org/10.1007/978-3-319-10602-1_48

[18] Zheng, Z., Wang, P., Liu, W., Li, J., Ye, R., & Ren, D. (2020). Distance-IoU loss: faster and better learning for bounding box regression. *Proceedings of the AAAI*, 34(7), 12993–13000. https://doi.org/10.1609/aaai.v34i07.6999