

# ***Real-Time Fire Hazard Detection and Alerting in a Collaborative Edge-Cloud System***

**Jiayu Sun**

*College of Computer and Electronic Information, Nanjing Normal University, Nanjing, China  
19220215@njnu.edu.cn*

**Abstract.** Early visual detection of fire hazards is critical for warehouses, factories, campuses and public venues. However, conventional smoke and heat alarms respond only after conditions cross a threshold and leave no visual record, while cloud-based analytics incur high bandwidth and scalability costs. To address these limitations, this paper develops a real-time fire hazard detection and alerting system that partitions the computational workload between edge and cloud. Each edge node runs a YOLOv11n detector for flame and smoke detection. Frame sampling reduces the inference workload, while confidence gating prevents low-confidence detections from being reported. Only confirmed events are transmitted to the cloud as annotated snapshots and metadata and visualized on operator dashboards. Experiments on a public fire-and-smoke dataset and four representative video streams show that the prototype achieves sub-second first-alert latency, sustains above-real-time processing under concurrent multi-camera load, and produces no false positives on normal or red-object interference footage.

**Keywords:** Fire hazard detection, Edge computing, Real-time alerting, Event-driven system

## **1. Introduction**

Fire hazards in warehouses, factories, campuses, and densely occupied public spaces require early warning because even short alarm delays can reduce the time available for intervention and evacuation [1, 2]. Since flame and smoke are often the first visible signs of an incipient fire, vision-based detection is a natural basis for rapid alerting [3].

Existing monitoring solutions fail to provide early visual evidence of fire hazards. Conventional smoke and heat sensors activate only after smoke concentration or temperature exceeds a fixed threshold and yield no visual record for subsequent review [3]. Cloud-based video analytics require continuous upload to remote servers, incurring substantial bandwidth consumption and round-trip latency under contended uplinks [4, 5]. In multi-node deployments, each additional stream arriving at the cloud translates into proportionally higher storage and compute overhead, compromising scalability [6].

Edge computing offers a practical alternative by moving inference close to the camera [6, 7]. Because most surveillance frames contain no hazard, local detection can discard routine frames and transmit only confirmed events. This event-driven strategy reduces upstream traffic while preserving the response speed required for fire monitoring [4]. However, this edge-based setting also faces

challenges: the pipeline must remain lightweight; the detector must distinguish flame and smoke from visually similar interference; repeated frames from a single physical event must be deduplicated; and distributed camera nodes still need a unified backend for storage, monitoring, and retrospective review.

To address these challenges, this paper designs a three-tier edge-cloud pipeline. Each camera node performs local dual-class YOLOv11n inference and uses frame sampling plus confidence gating before uploading an annotated snapshot [8]. A FastAPI cloud service aggregates alerts, stores event history, applies cooldown-based deduplication, and serves a web dashboard for live status and historical review [9].

This work introduces a three-tier edge-cloud architecture to enhance multi-camera fire hazard monitoring by separating edge-side inference from cloud-side tasks. It features a lightweight dual-class YOLOv11n detector for efficient flame and smoke recognition, suitable for edge deployment. Additionally, an event-triggered alerting workflow optimizes video transmission and minimizes false alarms through innovative edge and cloud interactions.

## 2. Methodology

### 2.1. System overview

The system is a three-tier edge-cloud pipeline [6]: an edge tier for local fire and smoke detection, a cloud tier for alert aggregation and persistence, and a web tier for unified operator monitoring. Each camera runs an independent edge instance connected to a single central cloud service that exposes a browser-accessible dashboard, as shown in Fig.1.

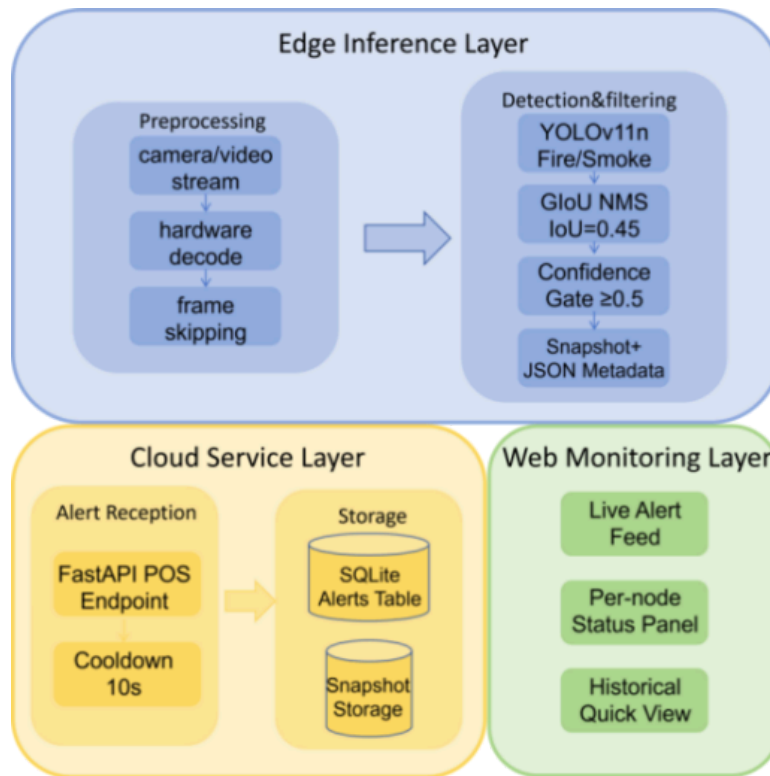


Figure 1. Architecture of the proposed fire hazard detection and alerting system

The three tiers communicate through narrow, well-defined interfaces. Edge-to-cloud alerts are sent as HTTP POST requests carrying one annotated snapshot and a small JSON metadata object; cloud-to-web state is served as JSON over the same web framework. This separation keeps the edge logic lightweight and allows the backend or dashboard to evolve without changing local detection.

## 2.2. Edge-side fire/smoke detection

On each edge node, YOLOv11n performs dual-class flame and smoke detection on locally decoded video and passes only confidence-filtered detections to the reporting module [8].

Frames are decoded with OpenCV [10], using hardware acceleration when available through NVIDIA video decoding [11]. To reduce computation, the node samples the stream and sends only selected frames to the detector.

The detector is initialized from the Ultralytics YOLO family, pre-trained on COCO, and fine-tuned for flame and smoke using PyTorch [8, 12, 13]. After inference, generalized IoU-based non-maximum suppression removes redundant boxes, leaving class labels, confidence scores, and frame coordinates for each detection [14].

A confidence gate at threshold 0.5 determines which detections are propagated from the edge node. Detections below this threshold are discarded before any further processing. The threshold trades recall against false-positive rate; in our experimental prototype it eliminates all false alarms on normal and red-object interference test sequences while still retaining fire and smoke detections on the corresponding positive sequences. Algorithm 1 summarizes the per-stream processing loop on the edge node.

## 2.3. Event-driven alert reporting

Edge nodes forward only annotated snapshots and JSON metadata when at least one detection passes the confidence gate. Each payload contains a JPEG with rendered bounding boxes plus the alert timestamp, node identifier, and per-detection class labels, bounding boxes, and confidence scores. Under sparse-event conditions, this event-driven design reduces upstream bandwidth by approximately three orders of magnitude—a raw 720p stream at 25 fps consumes several Mbps (rising to tens of Mbps at 4K) while a confirmed-event payload totals only tens of kilobytes. Deduplication is enforced at the cloud tier: a single physical fire yields one alert per sampled frame at the edge and one operator-facing alert after cooldown.

## 2.4. Cloud-side alert management

A FastAPI service at the cloud tier ingests alerts from all edge nodes, applies an event-level cooldown window to suppress duplicate notifications, persists alert history in a relational database, and exposes a web dashboard for unified monitoring.

The cloud service exposes a dedicated POST endpoint that accepts the multipart alert payload from the edge. Upon receipt, the service first checks whether the originating node is inside its 10-second cooldown window relative to its last accepted alert; if so, the new alert is logged as a duplicate and dropped without further action. Accepted alerts are written to a SQLite alerts table, indexed by timestamp and node identifier, and the snapshot file is stored on local disk under a path derived from the alert id.

The web monitoring layer is a stateless consumer of the cloud's persisted state. The dashboard renders three views: a live alert feed that periodically polls the cloud for new entries since the last

refresh, a per-node status panel showing each edge node's online state and recent alert indicators, and a historical query view that filters past alerts by time range, node, and class. The dashboard can be opened from any browser without direct access to the edge network because all state is stored in the cloud. This centralized backend keeps distributed camera nodes manageable while preserving a searchable event history for later review.

### 3. Experiments & results

#### 3.1. Experimental setup

This study fine-tunes the detector on the Smoke Fire Detection YOLO dataset, which contains 21,527 annotated images of flame and smoke, using the dataset's original train/val/test partition [15]. End-to-end evaluation additionally uses four in-house test videos independent of the training data: two positive clips (720p flame-and-smoke, 1080p pure smoke) expected to trigger alerts, and two negative clips (720p normal indoor, 4K with red-object interference) serving as a robustness benchmark.

YOLOv11n is trained in PyTorch for 100 epochs (batch 64, SGD with cosine decay) with mosaic and HSV augmentation on  $320 \times 320$  inputs [16]. At inference, frames are resized to  $640 \times 640$  with letterbox padding, decoded via OpenCV with H.264 hardware acceleration, and sampled every K frames on an RTX 3060 Laptop GPU [10]. The cloud tier is a FastAPI service backed by SQLite. The default configuration fixes  $K = 10$ ,  $\text{conf} = 0.5$ ,  $\text{cooldown} = 10$  s, and  $N = 4$  concurrent streams; ablations vary a single parameter, and YOLOv11n is additionally compared with YOLOv11s.

#### 3.2. Detection performance

YOLOv11n and YOLOv11s were trained under the same budget on the same partition. YOLOv11n reaches 0.702 mAP@0.5 and 0.477 mAP@0.5:0.95, while YOLOv11s (3.6× more parameters) achieves higher mAP@0.5 (+5.4%) and recall (+29.2%) but lower mAP@0.5:0.95 (−10.3%) and precision (−12.1%). YOLOv11s produces more candidates with less precise localization and more false positives at matched recall—unfavorable for cooldown-based deduplication that depends on per-detection precision. YOLOv11n is therefore preferred for edge deployment due to its 3.6× smaller memory footprint. Figure 2 clearly shows the comparison experiment results of the two models.

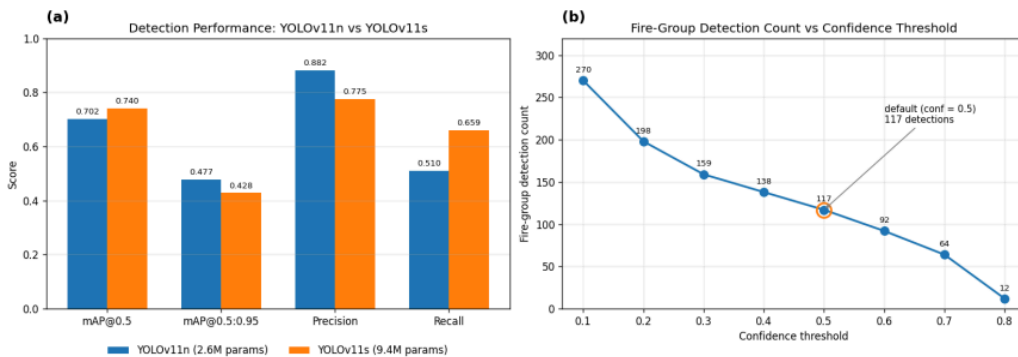


Figure 2. YOLOv11n evaluation. (a) versus YOLOv11s; (b) versus confidence threshold

### 3.3. Inference robustness

The confidence threshold trades recall against false-positive rate. This study sweeps confidence  $\in \{0.1, 0.2, \dots, 0.8\}$  at fixed  $K = 10$  and fixed cooldown = 10 s. At conf = 0.1, the system achieves the highest recall but also produces the only false positives observed across the threshold range. From conf = 0.2 upward, the two negative clips produce zero false positives across all threshold values, including the 4K red-object interference clip, which the model never misclassifies as fire even at the lowest tested threshold. Above conf = 0.5, recall continues to drop (117 to 12 detections across the positive clips) and first-alert latency increases substantially, because early low-confidence fire frames are discarded. The model is more robust on smoke than on fire across the threshold range, consistent with the per-class precision and recall values reported in the previous subsection (see Fig. 2).

At conf = 0.5, the system achieves zero false positives on both negative clips, retains 117 of 270 fire-group detections, and matches the  $K/\text{fps}$  theoretical first-alert latency. Across the full sweep, the only false positives are 2 smoke detections on `no\_smoke\_no\_fire` at conf = 0.1; the 4K red-object clip remains false-positive-free at all thresholds.

### 3.4. Multi-stream efficiency

The final ablation evaluates whether the edge-cloud pipeline can support multiple concurrent cameras. This study runs  $N \in \{1, 2, 3, 4\}$  edge nodes connected to one cloud instance and uses slowest-stream throughput as the main metric. The slowest-stream throughput is the relevant metric: an operator's worst-case video feed must remain above  $1.0 \times \text{real-time}$ .

Fig. 3 reports per-stream throughput and scaling efficiency as the number of concurrent edge nodes  $N$  grows from 1 to 4. Even at  $N = 4$ , the slowest stream still runs at  $2.22 \times \text{real-time}$ , comfortably above the  $1 \times \text{real-time}$  threshold required for uninterrupted alerting. Scaling efficiency is defined as the ratio of per-stream throughput at  $N$  concurrent nodes to single-stream throughput. This metric drops from 1.00 to 0.26, reflecting increasing GPU contention as more edge nodes share the same CUDA device. Notably, detection counts and false-positive rates at every  $N$  are identical to the single-stream baseline: added concurrency degrades throughput but does not compromise correctness.

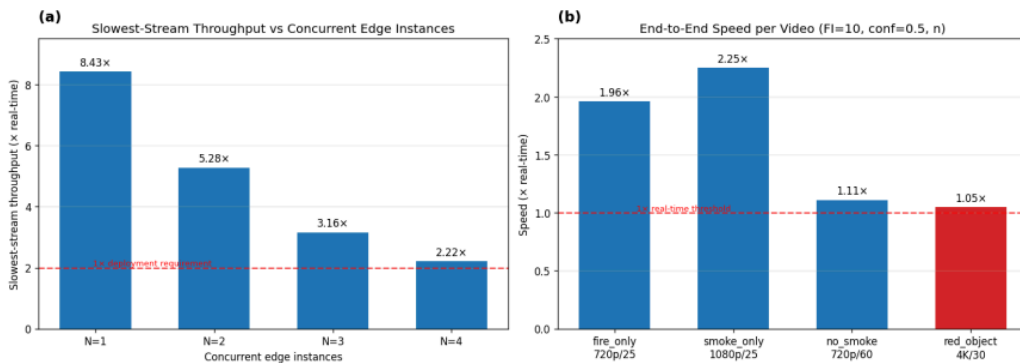


Figure 3. Sustained real-time performance. (a) concurrent; (b) per-video

### 3.5. Real-time alerting

Real-time alerting performance is governed by two deployment parameters: the frame interval  $K$  and the cloud-side cooldown window. Both are evaluated on the four test videos with the default model

(YOLOv11n), default confidence (0.5), and default cooldown (10 s for the K ablation; default K = 10 for the cooldown ablation).

With  $K \in \{1, 3, 5, 10, 20\}$ , the system processes the four test videos. At  $K = 10$ , all four videos achieve end-to-end speed above  $1.0\times$  real-time:  $1.96\times$  for 'fire\_only' (720p/25 fps),  $2.25\times$  for 'smoke\_only' (1080p/25 fps),  $1.11\times$  for 'no\_smoke\_no\_fire' (720p/60 fps), and  $1.05\times$  for 'red\_object\_interfere' (4K/30 fps). The first-alert latency is 0.40 s for every positive video, matching the theoretical worst-case latency  $K/\text{fps} = 10/25$  s and showing that the fire or smoke signature is detected on the first sampled frame. Fig. 3 illustrates that all four test videos sustain above  $1\times$  real-time end-to-end throughput at the default configuration, with the 4K red-object clip operating on the thinnest margin ( $1.05\times$ ).

The cooldown window determines how aggressively the cloud deduplicates repeated triggers of the same physical event. With  $\text{cooldown} \in \{1, 3, 5, 10, 30\}$  s, the 10.8-s 'fire\_only' clip produces 4 alerts at 1 s, 2 at 3 s, 3 at 5 s, 1 at 10 s, and 1 at 30 s. The 22.3-s 'smoke\_only' clip produces 11 alerts at 1 s, 6 at 3 s, 4 at 5 s, 3 at 10 s, and 1 at 30 s. At 10 s, each positive clip yields exactly one operator-visible alert; at 30 s the same, but the first-alert latency for a second event arriving within 30 s would be degraded. The choice of 10 s is therefore the smallest window that collapses a single physical event to one alert without sacrificing responsiveness to a subsequent event.

At the default configuration ( $K = 10$ ,  $\text{cooldown} = 10$  s), all four representative video streams run above  $1\times$  real-time, the first-alert latency is 0.40 s (matching the  $K/\text{fps}$  theoretical worst case), and every physical fire or smoke event yields exactly one operator-visible alert.

## 4. Conclusion

This paper presents a three-tier edge–cloud system for real-time fire hazard detection and alerting. The dual-class YOLOv11n detector achieves a favorable accuracy–efficiency trade-off over its larger YOLOv11s counterpart, making it suitable for deployment on consumer-grade edge GPUs shared across multiple camera streams. End-to-end evaluation confirms sub-second first-alert latency, above-real-time processing across a range of input resolutions, and zero false positives on both normal and red-object interference footage. The event-driven reporting strategy reduces upstream bandwidth by orders of magnitude relative to continuous video transmission, and throughput remains above real-time under concurrent multi-camera load. The current evaluation is limited to a single workstation and a small set of in-house test videos; geographically distributed deployment and robustness under low-light or backlit conditions remain as future work.

## References

- [1] Çetin, E., Dimitropoulos, K., Gouverneur, B., Grammalidis, N., Günay, O., Habiboğlu, Y. H., Töreyn, B. U., & Verstockt, S. (2013). Video fire detection – Review. *Digital Signal Processing*, 23(6), 1827–1843. <https://doi.org/10.1016/j.dsp.2013.07.003>
- [2] Hall, J. R., Jr. (2014). The total cost of fire in the United States. *Fire Technology*, 50(2), 261–286.
- [3] Töreyn, B. U., Dedeoğlu, Y., Güdükbay, U., & Çetin, A. E. (2006). Computer vision based method for real-time fire and flame detection. *Pattern Recognition Letters*, 27(1), 49–58. <https://doi.org/10.1016/j.patrec.2005.06.021>
- [4] Ananthanarayanan, G., Bahl, P., Bodík, P., Chintalapudi, K., Philipose, M., Ravindranath, L., & Sinha, S. (2017). Real-time video analytics: The killer app for edge computing. *Computer*, 50(10), 58–67. <https://doi.org/10.1109/MC.2017.3641638>
- [5] Hsieh, K., Hao, Y., Chen, B., & Kozuch, M. (2018). Focus: Querying large video datasets with low latency and low cost. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)* (pp. 1–16).

- [6] Satyanarayanan, M. (2017). The emergence of edge computing. *Computer*, 50(1), 30–39. <https://doi.org/10.1109/MC.2017.9>
- [7] Shi, W., Cao, J., Zhang, Q., Li, Y., & Xu, L. (2016). Edge computing: Vision and challenges. *IEEE Internet of Things Journal*, 3(5), 637–646. <https://doi.org/10.1109/JIOT.2016.2579198>
- [8] Ultralytics. (2024). Ultralytics YOLO11 documentation. *Ultralytics*. <https://docs.ultralytics.com/models/yolo11/>
- [9] Ramírez, S. (n.d.). FastAPI: A modern, fast (high-performance) web framework for building APIs with Python. *FastAPI*. <https://fastapi.tiangolo.com/>
- [10] Bradski, G. (2000). The OpenCV library. *Dr. Dobb's Journal of Software Tools*, 25(11), 120–125.
- [11] NVIDIA Corporation. (2024). Video Codec SDK: Video decode and presentation API for NVIDIA GPUs. *NVIDIA Developer*. <https://developer.nvidia.com/video-codec-sdk>
- [12] Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., & Zitnick, C. L. (2014). Microsoft COCO: Common objects in context. In *Proceedings of the European Conference on Computer Vision (ECCV)* (pp. 740–755).
- [13] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., & Chintala, S. (2019). PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems (NeurIPS)* (pp. 8024–8035).
- [14] Rezatofighi, H., Tsoi, N., Gwak, J., Sadeghian, A., Reid, I., & Savarese, S. (2019). Generalized intersection over union: A metric and a loss for bounding box regression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 658–666).
- [15] Kaggle. (2022). Smoke fire detection YOLO [Data set]. <https://www.kaggle.com/datasets/sayedgamel99/smoke-fire-detection-yolo/data>
- [16] Padilla, R., Netto, S. L., & da Silva, E. A. B. (2020). A survey on performance metrics for object-detection algorithms. In *Proceedings of the International Conference on Systems, Signals and Image Processing (IWSSIP)* (pp. 237–242). <https://doi.org/10.1109/iwSSIP48289.2020.9145130>