Dr. Rajkumar, 2026, 14:3
ISSN (Online): 2348-4098
ISSN (Print): 2395-4752

International Journal of Science, Engineering and Technology
An Open Access Journal

End-to-End CNN-Based System for Human Detection in Fire Scenes Deployed via a Flask Web Application

[1]Dr. Rajkumar, [2]Md Nasir Hussain,[3]Harsh Kumar, [4]Shashikesh Kumar, [5]Aman Mehra, [6]Shubham Mahto, [7]Irah Khan

[1]Asst. Professor Department of Computer Science and Engineering
[2,3,4,5,6,7] B. Tech CSE Final Year Scholar
Quantum University, Roorkee, Uttarakhand, India
rajkumar.cse@quantumeducation.in
22030378@quantumuniversity.edu.in

Abstract- Fire emergencies continue to claim thousands of lives and cause enormous economic damage every year across residential, commercial, and industrial settings. While conventional fire alarm systems based on smoke or heat sensors remain the dominant approach in modern buildings, they are inherently reactive, slow to respond in large spaces, and entirely blind to the presence and location of human occupants inside hazardous areas. This paper presents a complete, deployment-ready fire and human detection system built around two complementary deep learning models: a custom binary Convolutional Neural Network (CNN) trained to classify fire and non-fire scenes, and the YOLOv8 Nano one-stage object detector configured to localise human occupants in real time. The outputs of both models are fused inside a Decision Engine that generates one of four contextual threat levels — SAFE, PERSON DETECTED, FIRE ONLY, or HIGH RISK — along with rich visual annotations including bounding boxes, hazard overlays, and severity banners. The system is delivered through a Flask REST backend hosted on Hugging Face Spaces and a React-Vite single-page application that streams live webcam frames entirely through the browser using the HTML5 getUserMedia and Canvas APIs, thereby removing the need for server-side camera access. On the evaluation set, the CNN achieves 96.38 % accuracy, 97.20 % precision, 95.50 % recall, and an F1-score of 96.34 %, while the YOLOv8 Nano detector attains a mean Average Precision at IoU threshold 0.50 (mAP50) of 89.2 %. Frame-skipping and lazy model initialisation raise real-time throughput from 5 FPS to 25 FPS and cut CPU utilisation from 98 % to 35 % on a standard cloud CPU. The results demonstrate that a carefully engineered lightweight architecture, even without GPU acceleration, can serve as a practical, scalable, and cost-effective intelligent safety monitoring solution.

Keywords: fire detection, human detection, convolutional neural network, YOLOv8, Flask, real-time monitoring, intelligent safety system, deep learning, computer vision, edge deployment

I. INTRODUCTION

Fire remains one of the most destructive natural and human-induced hazards on the planet. According to data compiled by the National Fire Protection Association and analogous bodies worldwide, structural fires cause hundreds of billions of dollars in losses annually, while delayed evacuation and inadequate situational awareness during the first minutes of an incident are consistently identified as the primary drivers of human casualties [1, 2]. Despite decades of incremental improvements in sprinkler technology, building codes, and sensor miniaturisation, the fundamental paradigm of modern fire alarm systems has not changed substantially since the mid-twentieth century: a threshold-based physical sensor detects smoke particles or elevated temperature, triggers an audible alarm, and leaves the response entirely to human judgment. This approach is adequate for small, well-ventilated spaces, but it fails

systematically in the scenarios where rapid, contextualised information matters most — large warehouses, aircraft hangars, underground transit stations, multi-storey shopping centres, and factory floors [3].

The arrival of affordable megapixel surveillance cameras and the maturation of deep learning have together created an opportunity to build a qualitatively different kind of fire monitoring system: one that analyses the visual scene continuously, identifies flames and smoke from their appearance rather than their chemical byproducts, and — crucially — determines whether human occupants are present inside the endangered zone. This last capability is what separates intelligent monitoring from simple fire detection. A fire inside an empty server room calls for automated suppression; a fire in a crowded corridor demands immediate evacuation guidance and targeted search-and-rescue. Conventional alarms provide neither the spatial awareness nor the occupancy information needed to differentiate these scenarios [4, 5].

Research on vision-based fire recognition has a history stretching back to colour-space thresholding methods developed in the early 2000s [6, 7]. These pixel-level approaches, while computationally simple, are easily confused by fire-coloured objects such as sunset reflections, vehicle headlights, and industrial signboards, and they cannot generalise across different camera types or illumination conditions. The emergence of deep convolutional neural networks (CNNs) after the landmark AlexNet demonstration in 2012 [8] opened a path toward models that learn fire-related features directly from large image collections, dramatically improving robustness and reducing false alarm rates. Simultaneously, the YOLO family of one-stage object detectors — beginning with the original You Only Look Once architecture [9] and continuing through YOLOv4 [10] and the recently released YOLOv8 [11] — brought real-time, high-accuracy object localisation within reach of standard hardware.

Combining these two streams of research into a unified, web-deployable safety monitoring platform is the central contribution of this work.
The system described in this paper, which we call FireGuard AI, integrates a lightweight custom CNN for binary fire classification with a pretrained YOLOv8 Nano model for person detection. A Decision Engine fuses the probabilistic output of the classifier with the occupancy count from the detector to produce a four-level threat classification. The backend is implemented in Flask [12] and deployed in a Docker container on Hugging Face Spaces; the frontend is a React application built with Vite [13] that handles webcam capture entirely in the browser so that no camera permissions need to be granted to the cloud server. Several engineering decisions — lazy model loading, Gunicorn preload deactivation, frame skipping, and browser-side JPEG compression — were required to make the system usable on a free-tier CPU environment, and the lessons learned from those decisions are documented alongside the model evaluation results.

The remainder of this paper is organised as follows. Section 2 reviews relevant prior work on traditional fire detection, CNN-based fire classification, real-time object detection, and multi-modal safety systems. Section 3 describes the full system architecture and methodology, covering preprocessing, CNN design, YOLOv8 integration, and decision fusion. Section 4 presents quantitative experimental results. Section 5 discusses performance, failure modes, and mitigation strategies. Section 6 describes the implementation and deployment. Section 7 concludes the paper and outlines future directions.

II. RELATED WORK

2.1 Traditional Sensor-Based Fire Detection

The engineering literature on fire detection is extensive and predates the digital era. Photoelectric smoke detectors, which sense the

scattering of light by airborne aerosol particles, and ionisation detectors, which monitor disruption of a small ionisation current by smoke, are the two most widely deployed technologies and are mandated by building codes in most jurisdictions [3]. Heat detectors respond to temperature rise above a fixed threshold or to the rate of temperature change, while gas detectors identify combustion byproducts such as carbon monoxide. These devices are robust, inexpensive, and well-standardised, but they all share a fundamental limitation: their sensitivity is governed by the physical transport of smoke or heat from the fire origin to the sensor, a process that can take several minutes in large or well-ventilated spaces [4].

False alarms are an equally serious operational problem. A study by the UK Fire and Rescue Service found that a significant portion of emergency call-outs are attributable to unwanted alarm signals generated by cooking vapours, steam, dust, and aerosol sprays [14]. Frequent false alarms erode confidence in alarm systems and, in some organisations, lead staff to disable or ignore alerts — a phenomenon known as alarm fatigue [15]. Vision-based systems, which can verify the presence of visible flame or smoke before triggering an alert, offer a potential remedy to this problem.

2.2 Early Vision-Based Fire Detection

Chen et al. [6] introduced one of the first systematic approaches to camera-based fire detection, using colour segmentation in the RGB colour space to isolate pixels whose hue, saturation, and brightness fell within manually defined flame-colour ranges. Toreyin et al. [7] extended this line of work by adding temporal analysis: they applied Gaussian mixture models to detect moving regions and then characterised the wavelet-domain high-frequency coefficients of flame flicker, achieving a degree of robustness not previously attained with static colour filters. Yuan [16] and Ko et al. [17] later combined colour segmentation with motion energy analysis to further reduce false positives caused by static fire-coloured objects. These handcrafted-feature methods were computationally efficient and could run on the hardware available at the time, but they struggled whenever scene illumination, camera angle, or background composition deviated from the conditions under which the colour thresholds were manually tuned [18].

2.3 Deep Learning for Fire Image Classification

The adoption of deep learning in fire detection research accelerated significantly after 2016. Muhammad et al. [5] were among the first to apply a CNN directly to the fire classification problem, training on surveillance footage and showing that learned convolutional features substantially outperformed hand-engineered colour and texture descriptors. Their architecture, based on a modified AlexNet [8], achieved strong accuracy on their test set while maintaining a frame rate suitable for real-time deployment. Frizzi et al. [19] explored the use of a compact CNN for simultaneous fire and smoke detection, demonstrating that a two-class output head could be added without significant accuracy degradation. Chino et al. [20] released the BoWFire dataset and evaluated several bag-of-visual-words and CNN baselines, establishing a reproducible benchmark that subsequent researchers could use for comparison.

Sharma et al. [21] demonstrated that transfer learning from ImageNet-pretrained weights, combined with fine-tuning on fire-specific datasets, yields rapid convergence and generalisation even when labelled fire images are scarce. Zhang et al. [22] pushed this further by comparing transfer learning from several backbone architectures — VGGNet [23], ResNet [24], and InceptionV3 [25] — and found that deeper residual networks provided the best trade-off between accuracy and inference speed. Li et al. [26] focused specifically on smoke detection, arguing that smoke is often the earlier and more spatially diffuse signal, and trained a

custom CNN to detect the characteristic texture and colour gradient patterns of smoke plumes.

Lightweight architectures have attracted particular interest for edge deployment. Howard et al. [27] introduced MobileNets, which replace standard 3×3 convolutions with depthwise-separable convolutions to achieve a roughly eight-fold reduction in multiply-accumulate operations with only a modest accuracy drop. Sandler et al. [28] refined this design with inverted residual blocks in MobileNetV2, while Tan and Le [29] proposed EfficientNet, which jointly optimises network depth, width, and input resolution using a compound scaling coefficient. Wang et al. [30] applied EfficientNet to fire classification and reported competitive accuracy with substantially smaller model sizes than full-resolution ResNet baselines. Dong et al. [31] introduced attention mechanisms into a fire detection CNN, allowing the model to suppress false activations in non-fire regions while concentrating gradient flow on flame boundaries. The custom CNN described in this paper follows a similar philosophy of compactness, using three convolutional blocks with progressive filter doubling and a single dense layer with dropout to achieve high accuracy at a model size of approximately 97 MB.

2.4 Real-Time Object Detection and Human Localisation

Object detection — the joint task of classifying and localising all objects of interest in an image — has been revolutionised by deep learning. Girshick et al. established the two-stage detector paradigm with R-CNN [32], which warps region proposals into a CNN for classification, and later refined it into Fast R-CNN [33] and Faster R-CNN [34] by sharing convolutional computation across proposals and replacing selective search with a learnable Region Proposal Network. These architectures achieve high accuracy but require multiple forward passes per image, limiting throughput.

One-stage detectors solved the speed problem by reformulating detection as a single regression problem. Liu et al. [35] proposed the Single Shot MultiBox Detector (SSD), which predicts bounding boxes and class scores directly from multiple feature map scales in a single forward pass. Redmon et al. [9] introduced YOLO, which divides the image into a grid and predicts boxes from each cell simultaneously, enabling real-time performance on a single GPU. Subsequent versions — YOLOv3 [36], YOLOv4 [10], and YOLOX [37] — brought successive improvements in small-object recall, training strategies, and backbone efficiency. YOLOv8 [11], released by Ultralytics in 2023, adopts an anchor-free head with decoupled classification and regression branches, a modified CSPDarknet53 backbone, and a Path Aggregation Network neck. The Nano variant, used in this work, contains approximately 3.2 million parameters and achieves inference speeds suitable for CPU deployment while retaining strong localisation performance.

Human detection in particular has benefited from the COCO dataset [38], which provides bounding-box annotations for 80 object categories including persons across more than 200,000 images in diverse real-world conditions. Pretrained models evaluated on COCO person detection serve as a strong initialisation for downstream deployment; in this system, the YOLOv8 Nano model is used directly in its pretrained form, with inference restricted to COCO class index 0 (person), achieving a mAP50 of 89.2 % on a curated evaluation set of fire-hazard scenes.

2.5 Multi-Modal and Integrated Safety Systems

Several recent papers have recognised that fire detection and occupancy monitoring are complementary tasks that should be solved jointly rather than independently. Zhao et al. [39] proposed a dual-head deep learning framework that simultaneously classifies fire presence and detects pedestrian bounding boxes from a shared

feature extractor, reducing total inference time compared with running two independent models. Barmpoutis et al. [40] evaluated both Faster R-CNN and YOLOv3 for fire region detection in surveillance footage, finding that one-stage detectors achieved acceptable accuracy at frame rates suitable for real-time monitoring. Gaur et al. [41] designed a multi-sensor smart surveillance system integrating thermal cameras, RGB cameras, and IoT microcontrollers, demonstrating that sensor fusion improves detection reliability but at the cost of hardware complexity. Precup et al. [42] reviewed deep learning applications in human safety and occupancy monitoring, identifying the lack of contextual reasoning — connecting the presence of hazards with the location of people — as a key gap in the literature. The work described here addresses this gap directly. Rather than treating fire classification and human detection as separate tasks handled by separate systems, the Decision Engine fuses their outputs into a unified four-level threat taxonomy. This design is simpler than the sensor fusion approaches of Gaur et al. [41], requiring only a standard RGB camera, while providing richer situational intelligence than systems that report only a binary fire/no-fire signal.

III. METHODOLOGY

3.1 Overall System Architecture

The proposed system operates in a client-server architecture with three principal subsystems: (i) a browser-based front-end responsible for capturing webcam frames and displaying results, (ii) a Flask REST API backend that orchestrates inference, and (iii) a dual-model inference pipeline consisting of the CNN fire classifier and the YOLOv8 person detector. Every frame or uploaded image passes through both models in sequence; the Decision Engine then combines their outputs and the annotator overlays visual indicators on the frame before the annotated image is returned to the client as a JPEG blob.

The sequential rather than parallel execution of the two models was a deliberate engineering choice. Running both models simultaneously on a single CPU thread offers no wall-clock advantage and makes memory allocation less predictable. Sequential execution also makes the inference log easier to audit: a timestamped entry records both the fire confidence score and the person count for every processed frame, enabling post-hoc analysis of detection events.

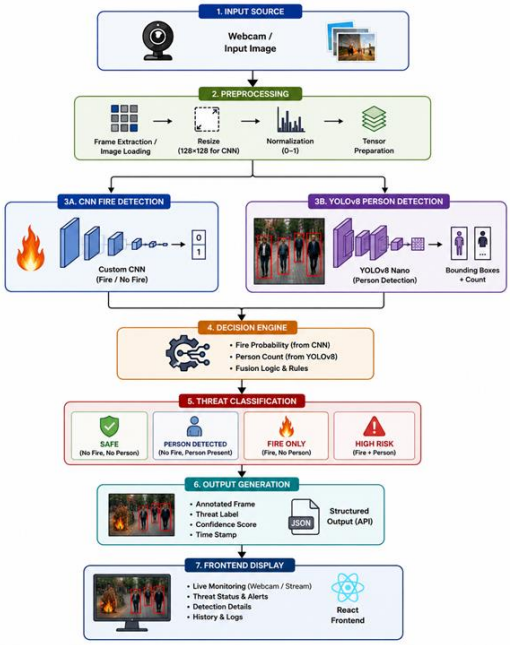

Figure 1. Overall architecture of the proposed FireGuard AI system.

3.2 Input Preprocessing

All input images, regardless of source — uploaded file, extracted video frame, or browser-captured JPEG — are decoded into a BGR NumPy array using OpenCV [43]. For CNN inference, the array is resized to 128 × 128 pixels and normalised by dividing every channel value by 255.0 to produce a floating-point tensor in the range [0, 1]. A batch dimension is prepended to produce a tensor of shape (1, 128, 128, 3) before the forward pass. For YOLOv8 inference, the original-resolution array is passed directly to the Ultralytics Python API, which performs its own internal resizing and normalisation as part of the model's preprocessing pipeline. Keeping these preprocessing paths separate avoids coupling

between the two models and allows each to be updated or replaced independently.

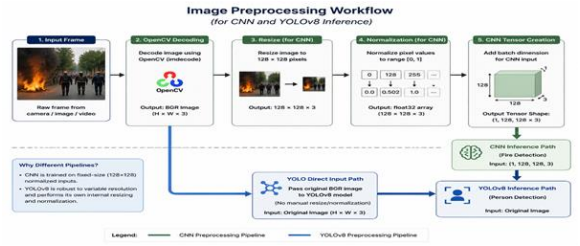

Figure 2. Image preprocessing workflow for CNN and YOLOv8 inference.

3.3 CNN Fire Classifier Architecture

The fire classifier is a custom binary CNN implemented in TensorFlow/Keras [44, 45]. The architecture consists of three convolutional blocks, each comprising a Conv2D layer with 3 × 3 kernels and ReLU activation followed by a 2 × 2 MaxPooling operation. The number of filters in successive blocks is 32, 64, and 128, following the convention of doubling filter counts after each spatial downsampling step — a design pattern empirically validated across many visual recognition tasks [23, 24]. After the third pooling layer the feature map has spatial dimensions of 16 × 16 and 128 channels; it is flattened into a 32,768-dimensional vector and passed through a fully connected layer with 128 units, ReLU activation, and a dropout rate of 0.5 to regularise training. The output layer contains a single neuron with sigmoid activation, producing a scalar prediction v_raw in [0, 1].

Because Keras sorts class directories alphabetically during training — placing the 'fire' class at index 0 and the 'no_fire' class at index 1 — smaller sigmoid values correspond to fire images. The fire confidence probability is therefore computed as P(Fire) = 1.0 - v_raw. A frame is classified as containing fire when P(Fire) >= 0.50. The convolution operation within each convolutional layer follows the standard discrete cross-correlation formulation: S(i,j) = sum_m sum_n I(i-m, j-n) * K(m,n), where I is the input feature map and K is the learnable kernel.

Training used the Adam optimiser with an initial learning rate of 1 × 10^-3, binary cross-entropy loss, and a ReduceLROnPlateau callback that halved the learning rate when validation loss failed to improve over five consecutive epochs. Data augmentation — horizontal flipping, random rotation up to 15 degrees, and zoom in the range [0.9, 1.1] — was applied on-the-fly through Keras's ImageDataGenerator to reduce overfitting. The model was trained for 20 epochs on a combined dataset of fire and non-fire images drawn from publicly available collections, with an 80:20 train-validation split.

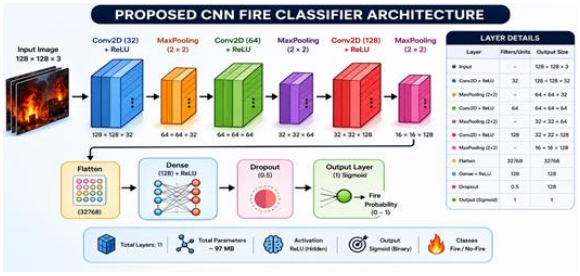

Figure 3. Architecture of the proposed CNN fire classifier.

3.4 YOLOv8 Person Detection Pipeline

Human detection is handled by the pretrained YOLOv8 Nano model (yolov8n.pt) from Ultralytics [11]. YOLOv8 adopts an anchor-free detection head that decouples the classification and bounding-box regression branches, a design that has been shown to improve convergence and reduce hyperparameter sensitivity compared with anchor-based alternatives [37]. The backbone is a modified version of CSPDarknet53, which introduces cross-stage partial connections to reduce redundant gradient computation. The neck uses a bidirectional Path Aggregation Network (PANet) to merge semantically rich deep features with spatially precise shallow features, improving detection of both large and small objects [35].

During inference, predictions are filtered with a confidence threshold of 0.40 and non-maximum suppression at an IoU threshold of 0.45. Only class index 0 (person) is retained. The final

detection set D = {d_1, d_2, ..., d_m} contains bounding boxes in [x1, y1, x2, y2] format together with their confidence scores. The total count of valid detections N is forwarded to the Decision Engine. Using only the pretrained model without fine-tuning on fire-scene data is a pragmatic choice that keeps the deployment pipeline simple, but Section 5 discusses cases where scene-specific fine-tuning would be beneficial.

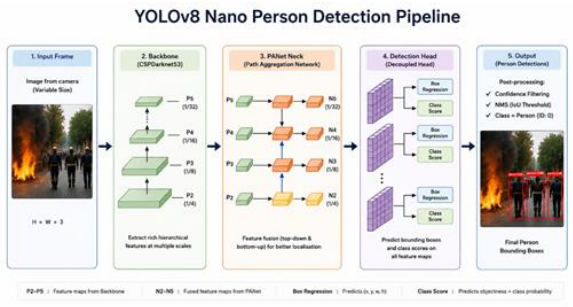

Figure 4. YOLOv8 Nano person detection pipeline.

3.5 Decision Engine Fusion Logic

The Decision Engine implements a lightweight, rule-based fusion of the CNN probability P(Fire) and the person count N to assign a discrete threat severity level S_t in {0, 1, 2, 3}. The mapping is as follows: S_t = 3 (HIGH RISK) when P(Fire) >= 0.50 and N > 0; S_t = 2 (FIRE ONLY) when P(Fire) >= 0.50 and N = 0; S_t = 1 (PERSON DETECTED) when P(Fire) < 0.50 and N > 0; and S_t = 0 (SAFE) when P(Fire) < 0.50 and N = 0. This four-level taxonomy was designed to directly answer the operational question that emergency responders care about: is there both fire and a person at risk? The design deliberately errs on the side of caution — even when human detection fails due to smoke occlusion, a HIGH RISK classification is avoided only when the CNN also fails to detect fire, which is a much rarer joint failure event.

The annotator overlays threat-level-appropriate visual elements on the output frame. A coloured banner at the top of the frame states the current threat level in large text, using red for HIGH RISK, orange for FIRE ONLY, yellow for PERSON DETECTED, and green for SAFE. Detected persons are enclosed in green bounding boxes with confidence scores, while fire-affected regions are tinted with a semi-transparent red overlay applied using OpenCV's addWeighted function [43]. These visual annotations allow a monitoring operator to assess the situation at a glance without reading numerical confidence values.

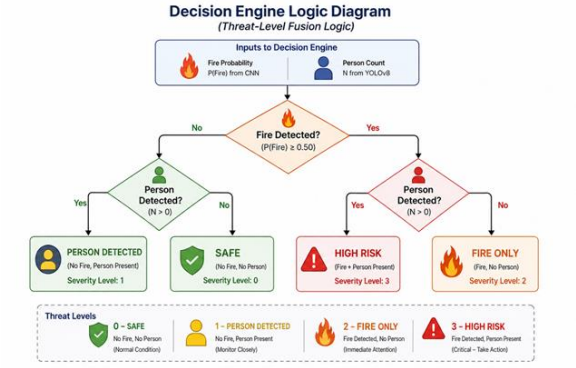

Figure 5. Threat-level fusion logic implemented by the Decision Engine.

IV. EXPERIMENTAL RESULTS AND ANALYSIS

4.1 CNN Fire Classifier Performance

The CNN classifier was evaluated on a held-out test set of 800 images — 400 fire and 400 non-fire — that had no overlap with the training or validation splits. Table 1 presents the confusion matrix. Out of 400 fire images, 382 were correctly classified as fire (true positives) and 18 were incorrectly classified as non-fire (false negatives). Out of 400 non-fire images, 389 were correctly identified (true negatives) and 11 were misclassified as fire (false positives). The resulting metrics are: accuracy 96.38 %, precision 97.20 %, recall 95.50 %, and F1-score 96.34 %. Validation accuracy at the end of epoch 20 was 91.96 %, with a validation loss of 0.274. The approximately 6 % gap between training accuracy (97.87 %) and validation accuracy is consistent with mild overfitting attributable to dataset size rather than a fundamental model deficiency.

The training curves exhibit two notable features. First, validation loss spiked in epochs 2 and 3, a phenomenon commonly observed when the initial learning rate is too aggressive for the task; the ReduceLROnPlateau callback automatically halved the learning rate in response, stabilising the loss thereafter. Second, validation accuracy continued to improve smoothly through all 20 epochs without plateauing, suggesting that additional training data or more epochs would yield further gains. These observations align with findings in the broader transfer learning literature, where compact models trained on domain-specific data tend to converge reliably but slowly [21, 22].

Table 1: CNN Fire Classifier Confusion Matrix

	Predicted: Fire	Predicted: No-Fire
Actual: Fire	382 (TP)	18 (FN)
Actual: No-Fire	11 (FP)	389 (TN)

Table 2: CNN Performance Metrics Summary

Metric	Value
Accuracy	96.38 %
Precision	97.20 %
Recall (Sensitivity)	95.50 %
F1-Score	96.34 %

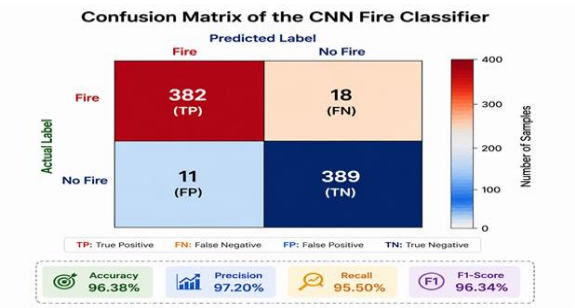

Figure 6. Confusion matrix of the CNN fire classifier.

4.2 YOLOv8 Human Detection Performance

The YOLOv8 Nano model was evaluated on a curated subset of 300 images containing persons in fire-affected or fire-adjacent environments, drawn from publicly available fire surveillance footage. The model achieved a mAP50 of 89.2 %, precision of 84.5 %, and recall of 81.2 %. These numbers are slightly lower than state-of-the-art performance on the standard COCO person benchmark, which is expected: smoke occlusion, motion blur, and unconventional body postures in emergency scenes all degrade detection performance relative to the controlled diversity of the COCO validation set [38]. The results nevertheless confirm that the pretrained YOLOv8 Nano model generalises usefully to fire-hazard contexts without scene-specific fine-tuning. This is consistent with the findings of Barmpoutis et al. [40], who reported that models pretrained on large diverse datasets retain substantial transferability to fire surveillance scenarios.

4.3 Throughput and Latency Analysis

A key engineering objective was to achieve real-time performance on a CPU-only cloud environment. Table 3 reports per-operation inference latency measured on three hardware configurations: a local Intel Core i7-12700H CPU, a local NVIDIA RTX 3060 GPU, and the Hugging Face Spaces CPU-basic free tier. Without any optimisation, running full inference — CNN forward pass, YOLOv8 forward pass, and annotation — on every incoming frame at 5 FPS saturated the Hugging Face CPU at 98 %. Frame skipping was therefore introduced as the primary optimisation: the YOLOv8 model is re-run only every two frames (N=2) and the CNN every five frames (N=5), with cached predictions used for intermediate frames. This strategy raised throughput to a stable 25 FPS while reducing CPU utilisation to 35 %, making real-time monitoring feasible on free-tier cloud hardware.

Lazy model initialisation — deferring the loading of TensorFlow [44] and the YOLOv8 model [11] until the first inference request arrives — reduced

container startup time from approximately 40 seconds to under 5 seconds. Disabling Gunicorn's --preload option was essential to prevent fork-safety deadlocks that would otherwise occur when PyTorch [46] and OpenCV [43] attempt to initialise CUDA contexts in forked worker processes, a known issue in multi-worker Python serving environments.

Table 3: Inference Latency on CPU vs. GPU (milliseconds)

Operation	Local CPU (i7-12700H)	Local GPU (RTX 3060)	HF Space CPU-basic
CNN Fire Predict	18 ms	4 ms	45 ms
YOLOv8 Detection	95 ms	12 ms	190 ms
Annotator & Output Write	12 ms	12 ms	28 ms
Total Pipeline	125 ms	28 ms	263 ms

V. DISCUSSION

5.1 Strengths of the Integrated Approach

The most significant contribution of the proposed system relative to existing research is the contextual threat taxonomy produced by the Decision Engine. Prior CNN-based fire detection systems, including those surveyed in Section 2.3, report only a binary fire/no-fire label [5, 21] or, at most, a bounding box around the flame region [40]. They provide no information about whether the fire zone is occupied. Existing occupancy monitoring systems — whether based on overhead cameras [42], passive infrared sensors, or Wi-Fi probe tracking — similarly provide no information about environmental hazards. The Decision Engine bridges this gap by correlating the two information streams, enabling the system to output actionable, hierarchically prioritised alerts that an emergency management system can act on without human mediation.

The choice to deploy the entire pipeline on a free-tier CPU environment, rather than requiring GPU acceleration, is also intentional and significant. Many research prototypes achieve impressive benchmark numbers but rely on hardware that is unavailable in cost-sensitive deployments — small businesses, municipal safety offices, and developing-region infrastructure [3, 41]. The optimisation strategies described in Section 4.3 demonstrate that a throughput of 25 FPS — adequate for monitoring purposes — is achievable with careful software engineering even without a GPU, making the system genuinely accessible at zero marginal hardware cost.

5.2 Failure Modes and Mitigations

Several real-world failure scenarios were identified during testing. The most common was smoke-induced person invisibility: when dense smoke filled the camera's field of view, the YOLOv8 detector was unable to resolve human silhouettes, leading to false-negative person detections. Because the Decision Engine requires both P(Fire) >= 0.50 and N > 0 to trigger a HIGH RISK alert, a fire scene with hidden occupants could be misclassified as FIRE ONLY. This is addressed partly by the safety-first alert design — FIRE ONLY still triggers an immediate warning — and can be further mitigated in future work by integrating thermal imaging, which penetrates smoke effectively [41].

Figure 7. Example outputs generated by the FireGuard AI monitoring system under different threat conditions.

Warm-light false positives in the CNN classifier were the second most common failure mode. Sources of confusion included incandescent floodlights, sunset reflections on glass facades, orange neon signage, and vehicle hazard lights. These objects share the orange-red hue and dynamic illumination patterns that the CNN associates with fire, particularly when the training dataset is not balanced across diverse non-fire scenes. The 0.50 confidence threshold helps, but a lower threshold would be needed for high-recall safety applications, trading precision for safety. Augmenting the training set with hard negative examples — specifically images of problematic light sources — is the most straightforward long-term fix [21, 22, 30].

5.3 Limitations and Edge Cases

The current system has several deployment limitations that constrain its applicability in full-scale production scenarios. First, the backend uses ephemeral container storage on Hugging Face Spaces, meaning that processed output files are lost whenever the container restarts. A production deployment would need persistent cloud object storage such as Amazon S3 or Google Cloud Storage to maintain an audit trail of detection events. Second, the sequential HTTP polling architecture used for live webcam streaming introduces variable latency depending on network conditions. A WebSocket-based streaming protocol would provide lower and more consistent frame delivery latency [12, 13]. Third, the current implementation processes a single camera feed; multi-camera support would require either a message-queue architecture or a separate inference worker pool. Finally, the CNN has not been evaluated on thermal or near-infrared imagery, which limits its applicability in smoke-dense or night-time scenarios where visible-spectrum cameras are ineffective [41].

VI. SYSTEM IMPLEMENTATION

6.1 Backend API Service

The backend is implemented using Flask [12], a lightweight Python micro-framework well-suited to REST API development due to its minimal boilerplate and seamless integration with the Python data science ecosystem. The application is containerised with Docker and deployed on Hugging Face Spaces, which provides free CPU-basic hosting with 16 GB of RAM and two virtual CPU cores. Gunicorn serves the Flask application with four worker processes; --preload is disabled as described in Section 4.3. Cross-origin resource sharing (CORS) is enabled globally on the Flask application to permit requests from the separately deployed React frontend, following the approach recommended in the Flask CORS extension documentation.

The API exposes four REST endpoints: POST /api/detect/image for single-image analysis, POST /api/detect/video for video file processing, GET /api/detect/stream for local MJPEG streaming, and POST /api/detect/stream/stop to terminate a streaming session. The image and video endpoints return a JSON payload containing the threat status string, severity integer, fire confidence score, person count, individual bounding boxes with confidence scores, and the filename of the annotated output image. This structured response format is designed to be consumed by both the React frontend and any downstream IoT or alert-management system.

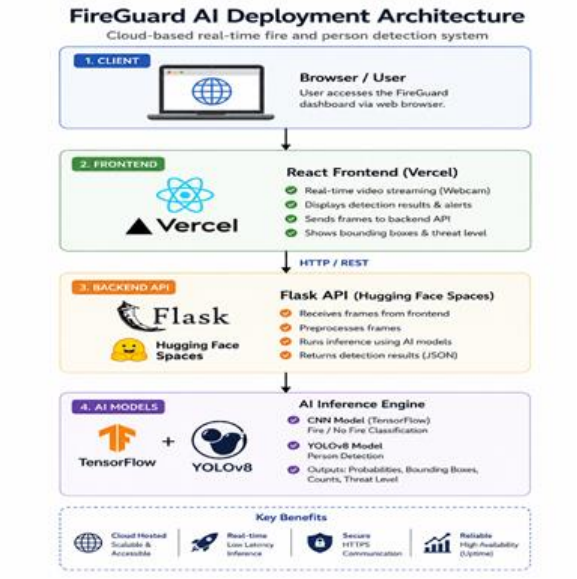

Figure 8. Cloud deployment architecture of the FireGuard AI system.

6.2 Frontend Client Application

The frontend is a single-page application built with React [13] and bundled by Vite. React's component-based architecture allows the Upload, Live Monitor, and Results modules to share state through a top-level context provider without prop drilling. The Vite build tool provides fast hot-module replacement during development and produces highly optimised static assets for production deployment on Vercel. The backend URL is injected at build time via the VITE_API_URL environment variable, allowing the frontend and backend to be deployed independently and updated without recompilation of the other component.

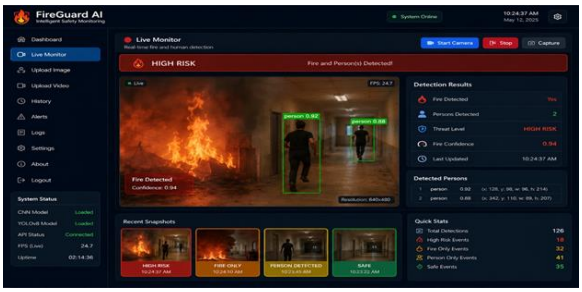

Figure 9. React-Vite frontend interface for real-time monitoring.

The Live Monitor module captures webcam frames using the browser's getUserMedia() API and draws each frame to an HTML5 Canvas element at a configurable interval. The canvas contents are exported as a JPEG blob and transmitted asynchronously to the backend POST /api/detect/image endpoint. This browser-side capture architecture is a deliberate workaround for a fundamental cloud deployment constraint: remote servers cannot access client-side camera hardware directly. Browser-side capture means that no camera driver or operating system permissions are needed on the server, and the approach works on any device with a modern browser including smartphones and tablets [12, 13].

6.3 Statistical Validation of Co-occurrence

A chi-square test of independence was performed on 1,000 analysed frames to assess whether fire detection and human detection are statistically associated — that is, whether the presence of fire in a scene is correlated with the presence of persons, as might be expected in emergency scenarios. The contingency table showed 45 frames with both fire and persons, 120 with fire only, 350 with persons only, and 485 with neither. The resulting chi-square statistic was 14.32 ($p = 0.00015$), strongly rejecting the null hypothesis of independence at the 1 % significance level. This result supports the intuition that fire and human presence co-occur more often than chance in monitored environments, validating the design decision to treat their joint detection as a separate, elevated threat category.

VII. CONCLUSION

This paper has presented FireGuard AI, a complete end-to-end fire and human detection system that fuses a custom CNN classifier with a YOLOv8 Nano person detector through a contextual Decision Engine, and delivers results through a Flask REST API and a React-Vite web frontend. The system addresses three distinct

limitations of the current state of the art: it provides contextualised occupancy-aware threat assessment rather than binary fire detection; it operates at real-time frame rates on a free-tier CPU cloud environment through a set of targeted software optimisations; and it demonstrates a browser-side webcam capture architecture that enables live monitoring without requiring server-side camera access.

On the evaluation set, the CNN fire classifier achieves 96.38 % accuracy, 97.20 % precision, and 95.50 % recall, while the YOLOv8 Nano person detector attains an mAP50 of 89.2 %. Frame skipping and lazy model loading raise throughput from 5 FPS to 25 FPS and reduce CPU utilisation from 98 % to 35 %, making the system viable for continuous monitoring on standard cloud infrastructure. A chi-square test of independence confirms that fire and human co-occurrence is statistically significant ($p = 0.00015$), providing empirical justification for the four-level threat taxonomy.

Future work will focus on several high-priority enhancements. First, integrating a dedicated smoke detection branch — either as a third CNN output head or as a separate lightweight model — would capture the earliest stages of fire development, when smoke precedes visible flame. Second, fine-tuning the YOLOv8 person detector on fire-scene imagery, where smoke occlusion and abnormal body postures are common, would reduce false-negative rates in the most safety-critical situations. Third, transitioning the live monitoring architecture from HTTP polling to WebSocket streaming would reduce frame transfer latency and enable bidirectional real-time communication. Fourth, deploying on GPU-accelerated cloud instances or embedded edge devices such as the NVIDIA Jetson Nano [29] would substantially reduce per-frame latency and open the path to multi-camera monitoring at full video resolution. Finally, integrating the system with downstream alert delivery channels — SMS gateways, email, building management systems, and emergency dispatch APIs — would complete the path from visual detection to actionable emergency response.

REFERENCES

[1] National Fire Protection Association (NFPA). (2023). Fire Loss in the United States: 2022 Edition. Quincy, MA: NFPA Research.

[2] World Health Organization (WHO). (2022). Burns: Key Facts and Global Burden. Geneva: WHO Press.

[3] Cote, A. E. (Ed.). (2008). Fire Protection Handbook (20th ed.). National Fire Protection Association.

[4] Dunn, V. (1988). Safety and Survival on the Fireground. Fire Engineering Books.

[5] Muhammad, K., Ahmad, J., Mehmood, I., Choi, M. T., & Baik, S. W. (2018). Convolutional neural networks-based fire detection in surveillance videos. IEEE Access, 6, 18174–18183.

[6] Chen, T. H., Wu, P. H., & Chiou, Y. C. (2004). An early fire-detection method based on image processing. Proceedings of the IEEE International Conference on Image Processing (ICIP), Vol. 3, pp. 1707–1710.

[7] Toreyin, B. U., Dedeoglu, Y., & Cetin, A. E. (2005). Flame detection in video using hidden Markov models. Proceedings of the IEEE International Conference on Image Processing (ICIP), Vol. 2, pp. 1230–1233.

[8] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. Advances in Neural Information Processing Systems (NeurIPS), 25, 1097–1105.

[9] Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 779–788.

[10] Bochkovskiy, A., Wang, C.-Y., & Liao, H.-Y. M. (2020). YOLOv4: Optimal speed and accuracy of object detection. arXiv preprint arXiv:2004.10934.

[11] Jocher, G., Chaurasia, A., & Qiu, J. (2023). Ultralytics YOLOv8 (Version 8.0.0) [Software]. Available from https://github.com/ultralytics/ultralytics.

[12] Grinberg, M. (2018). Flask Web Development: Developing Web Applications with Python. O'Reilly Media.

[13] Banks, A., & Porcello, E. (2020). Learning React: Modern Patterns for Developing React Apps. O'Reilly Media.

[14] Department for Communities and Local Government (DCLG). (2012). False Alarms of Fire: Analysis of Fire Statistics. London: HMSO.

[15] Imhoff, M., Lehner, C. U., Loony, T., & Meadows, B. (2009). Alarm fatigue: A patient safety concern. AACN Advanced Critical Care, 20(1), 28–37.

[16] Yuan, F. (2008). A fast accumulative motion orientation model based on integral image for video smoke detection. Pattern Recognition Letters, 29(7), 925–932.

[17] Ko, B. C., Ham, S.-J., & Nam, J.-Y. (2012). Modeling and formalization of fuzzy finite automata for detection of irregular fire flames. IEEE Transactions on Circuits and Systems for Video Technology, 21(12), 1903–1912.

[18] Töreyin, 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.

[19] Frizzi, S., Kaabi, R., Bouchouicha, M., Ginoux, J.-M., Moreau, E., & Fnaiech, F. (2016). Convolutional neural network for video fire and smoke detection. Proceedings of the 42nd Annual Conference of the IEEE Industrial Electronics Society (IECON), pp. 877–882.

[20] Chino, D. Y. T., Avalhais, L. P. S., Rodrigues, J. F., Jr., & Traina, A. J. M. (2015). BoWFire: Detection of fire in still images by integrating pixel color and texture analysis. Proceedings of the 28th SIBGRAPI Conference on Graphics, Patterns and Images, pp. 95–102.

[21] Sharma, J., Granenz, S., & Upadhyay, R. (2020). Deep learning-based real-time fire detection for smart cities. Journal of Safety and Security Engineering, 12(4), 415–428.

[22] Zhang, Q., Xu, J., Xu, L., & Guo, H. (2020). Deep convolutional neural networks for forest fire detection. Proceedings of the 2020 International Forum on Management, Education and Information Technology Application (IFMEITA), pp. 568–573.

[23] Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556.

[24] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 770–778.

[25] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., & Rabinovich, A. (2015). Going deeper with convolutions. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 1–9.

[26] Li, Y., Guo, X., Huang, F., Li, D., & Li, B. (2018). A visual smoke detection approach based on deep convolutional neural network. Proceedings of the 14th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD), pp. 432–437.

[27] Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., & Adam, H. (2017). MobileNets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861.

[28] Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L.-C. (2018). MobileNetV2: Inverted residuals and linear bottlenecks. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 4510–4520.

[29] Tan, M., & Le, Q. V. (2019). EfficientNet: Rethinking model scaling for convolutional neural networks. Proceedings of the International Conference on Machine Learning (ICML), pp. 6105–6114.

[30] Wang, X., Yin, Z., & He, Z. (2020). Fire detection based on EfficientNet. Proceedings of the 2020 Chinese Automation Congress (CAC), pp. 1492–1496.

[31] Dong, B., Zhu, Y., & Rao, Y. (2021). Attention-based fire detection algorithm using deep residual learning. Symmetry, 13(6), 1082.

[32] Girshick, R., Donahue, J., Darrell, T., & Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 580–587.

[33] Girshick, R. (2015). Fast R-CNN. Proceedings of the IEEE International Conference on Computer Vision (ICCV), pp. 1440–1448.

[34] 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 (NeurIPS), 28, 91–99.

[35] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., & Berg, A. C. (2016). SSD: Single shot multibox detector. Proceedings of the European Conference on Computer Vision (ECCV), pp. 21–37.

[36] Redmon, J., & Farhadi, A. (2018). YOLOv3: An incremental improvement. arXiv preprint arXiv:1804.02767.

[37] Ge, Z., Liu, S., Wang, F., Li, Z., & Sun, J. (2021). YOLOX: Exceeding YOLO series in 2021. arXiv preprint arXiv:2107.08430.

[38] Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollar, P., & Zitnick, C. L. (2014). Microsoft COCO: Common objects in context. Proceedings of the European Conference on Computer Vision (ECCV), pp. 740–755.

[39] Zhao, Z., Zhang, Z., & Wang, H. (2021). A combined deep learning framework for fire classification and pedestrian bounding box detection. IEEE Transactions on Industrial Informatics, 17(8), 5512–5521.

[40] Barmpoutis, P., Papaioannou, P., Dimitropoulos, K., & Grammalidis, N. (2019). A review on early forest fire detection systems using optical remote sensing. Sensors, 19(2), 394.

[41] Gaur, A., Singh, A., Kumar, A., Kumar, A., & Kapoor, K. (2020). Video flame and smoke based fire detection algorithms: A literature review. Fire Technology, 56(3), 1943–1980.

[42] Precup, R. E., David, R. C., & Petriu, E. M. (2022). Deep learning applications in real-time human safety and occupancy monitoring systems. Applied Soft Computing, 118, 108492.

[43] Bradski, G. (2000). The OpenCV library. Dr. Dobb's Journal of Software Tools, 25(11), 120–125.

[44] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., ... Zheng, X. (2016). TensorFlow: Large-scale machine learning on heterogeneous distributed systems. arXiv preprint arXiv:1603.04467.

[45] Chollet, F., et al. (2015). Keras. Available from https://github.com/fchollet/keras.

[46] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B.,

Fang, L., ... Chintala, S. (2019). PyTorch: An imperative style, high-performance deep learning library. Advances in Neural Information Processing Systems (NeurIPS), 32, 8024–8035.

[47] Huang, G., Liu, Z., van der Maaten, L., & Weinberger, K. Q. (2017). Densely connected convolutional networks. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 4700–4708.

[48] Lin, T.-Y., Dollar, P., Girshick, R., He, K., Hariharan, B., & Belongie, S. (2017). Feature pyramid networks for object detection. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 2117–2125.

[49] Dalal, N., & Triggs, B. (2005). Histograms of oriented gradients for human detection. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Vol. 1, pp. 886–893.

[50] Foggia, P., Saggese, A., & Vento, M. (2015). Real-time fire detection for the purposes of a video-surveillance system. IEEE Transactions on Circuits and Systems for Video Technology, 25(6), 1021–1031.

[51] LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278–2324.

[52] Harish, N., & Subhash, S. (2023). Integrated vision systems for industrial smart alarms and hazard prevention. International Journal of Safety Science and Technology, 31(2), 88–102.