International Journal of Scientific Engineering and Science
Volume 9, Issue 11, pp. 125-132, 2025. *ISSN (Online): 2456-7361*

INFERNO: Smart Real-Time Security Camera System with Fire, Smoke, and Intruder Detection with Integrated Notification Capabilities

Kuo-Pao Yang[1], Olisemeka Nmarkwe[2], Brian Britton[3], Collin Cowart[4], Angel Njoku[5]
[1, 2, 3, 4, 5]Computer Science Department, Southeastern Louisiana University, Hammond, Louisiana, USA

Abstract—*Fire and smoke detection systems play a crucial role in preventing the catastrophic damage to property and life. This paper introduces a smart detection system designed to identify fire, smoke, people, and faces in real-time with integrated notification capability. The recurring and destructive wildfires inspire this project in and around the Los Angeles area, where early detection can mean the difference between safety and tragedy. By combining deep learning with computer vision, the system can analyze video feeds to detect signs of danger and locate individuals who may be trapped or in harm's way. INFERNO features a responsive dashboard for real-time monitoring, configuration management and alert visualization. Experimental results demonstrate this system effectiveness in accurately detecting fire and smoke incidents with minimal false alarms; while running on a resource constrained device The INFERNO system represents a significant advancement in accessible, customizable, and reliable fire safety monitoring solution. This paper covers the design, implementation, testing, and future potential of the system, which is aimed at enhancing safety in wildfire-prone regions and potentially scaling to other disaster response scenarios.*

Keywords— *Fire Detection, Computer Vision, YOLO, Raspberry Pi, Real-Time Monitoring.*

I. Introduction

Fire incidents are one of the most devastating disasters affecting both residential and commercial buildings; they even affect forests too. According to the National Fire Protection Association (NFPA) research, the fire departments around the US responded to an estimated 1.4 million fires in 2020. These fires caused the death of 3,500 and injuries to 15,200 civilians and also an estimated 21.9 billion USD in property damage [1]. Early detection of fire is important for saving lives and minimizing damage done to property; even a few second warnings can significantly improve emergency response time and evacuation.

Traditional methods of fire detection rely on physical sensors to detect smoke, infrared, or temperature. While these systems have been effective in some cases, they have inherent drawbacks. Firstly, these sensors require the smoke or heat to reach the surface of the device before an alarm is triggered, which causes delayed detection. Additionally, these systems struggle with false alarms as the systems can be deceived, for example, an infrared sensor can be triggered by sunlight or the presence of certain vapors, and smoke alarms can be triggered by cooking fumes. They also provide limited information on the location of the fire.

Computer-vision-based fire and smoke detection systems have emerged as a way to complement traditional sensor-based detection methods. These systems use cameras to monitor spaces and apply image processing and machine learning techniques to detect visual patterns associated with fire and smoke in these spaces. Early computer vision methods employed rule-based algorithms that focused on color, motion, and texture features of the fire and smoke. While these methods are successful, they struggle with artificial lighting, complex backgrounds, and different types of fires.

A more recent approach involves using deep learning architectures like convolutional neural networks (CNN) to improve the accuracy of detections. This approach has led to a significant improvement over traditional approaches but faces challenges as the video frames are processed in real time, and this is challenging for edge devices. Recent studies like FireNet have shown that lightweight neural networks can achieve real-time fire and smoke detection even on resource-constrained IoT devices [2]. Also, many of the present solutions focus on detection but lack a sufficient notification pipeline, live monitoring, and a practical management system.

The Intelligent Network for Flame Evaluation and Rapid Notification Operations (INFERNO) system addresses these limitations by using hybrid detection, which combines a state-of-the-art object detection model and a traditional smoke MQ-2 sensor, with a user-friendly web interface and good notification channels.

The primary objective of this system is to provide real-time detection of fire, smoke, and intruders by using efficient algorithms that ensure high accuracy and minimal false alarms. It aims to offer multi-hazard monitoring by including face and motion detection for comprehensive safety and security coverage. This system also focuses on delivering a user-friendly interface through an intuitive web dashboard for monitoring, configuration, and notification management. Additionally, it is designed for compatibility with edge devices, optimized to run on resource-constrained hardware like the Raspberry Pi through techniques such as multithreading and native media playback. Finally, it supports customizability by allowing users to adjust detection parameters, notification settings, and the IP camera used.

The remainder of this paper is organized as follows. Section II reviews related work in fire detection systems, computer vision–based approaches, and intruder detection technologies. Section III presents the implementation details and overall system architecture of INFERNO. Section IV evaluates the system's performance in terms of detection accuracy, processing speed, and resource utilization. Finally, Section V discusses the system's strengths and limitations, outlines potential improvements, and offers concluding remarks.

II. Related Work

YOLOv8 [3] was employed in the object detection model to identify fire and smoke, as it has proven to be highly effective for fire detection due to its accuracy and low latency. Similar to our project, two classes are used to differentiate smoke and fire. They also include a virtual language model called Blip-2 that "reaches state-of-the-art performance in both image captioning and VQA." The IC and VQA functions of Blip-2 check the highlighted image frame that is displayed when YOLO perceives a fire and determine whether this frame is a fire, smoke, both fire and smoke, or neither fire nor smoke. They train Blip-2 in a variety of situations while training YOLO on specific fire situations to get the best results of both models. Hence, future versions of this project could achieve greater results with multiple integrated models, yet more powerful hardware would likely be required.

A real-time fire detection system was developed using convolutional neural networks (CNNs) applied to surveillance video streams [4]. Their goal was to fix the problems with traditional fire detection methods, such as high false alarm rates and slow response times. After seeing the success of their work, we decided to adapt their method for our system, and it worked well for us. They trained their system with a dataset of different fire and non-fire images, allowing it to learn important visual features like color, texture, and flame movement. By using a CNN model, the system was able to automatically pick up on these features without needing anyone to manually design them. This was another idea we applied from their research, as we trained our model with thousands of fires and non-fire images.

The smart home security system [5][6] integrates IoT devices with machine learning algorithms to enable real-time intrusion detection and rapid alerts. As with the advanced security system [7], the primary contribution of our system is the integration of AI models for smoke and fire detection. Although our project includes several similar components, such as an MQ-2 sensor and AI-based facial recognition, their use of an ultrasonic motion sensor and a more advanced GUI provide useful inspiration for creating a hybrid design that could further improve the final system.

Using this approach, INFERNO was able to accurately differentiate between fire and non-fire images and even estimate the distance of the fire within these images. The CNN system demonstrated that deep learning methods outperform traditional vision techniques in complex environments by automatically learning high-level feature representations. Their research showed that deep learning techniques provide a reliable and scalable solution for real-world fire surveillance, particularly in situations where conventional sensors may be insufficient.

Inspired by an intelligent classroom system where facial recognition tracked human presence and activity [8][9], we applied a comparable approach for detecting faces and motion in our project. In our case, we used a camera to monitor spaces, detect faces, and send images of any detected intruder directly to the user via Telegram. This allowed us to extend our fire detection system into a complete fire and security monitoring platform.

III. Implementation

A. Overall System Design

The system is designed to be a comprehensive solution for fire, smoke, and intruder detection with integrated notification capabilities. The system architecture consists of four main components: the detection system (core), web server, frontend interface, and notification service. Fig. 1 is a simple diagram that illustrates the overall system architecture and the interaction between these components.

The system follows a client-server architecture where the detection system processes the incoming video feed, which is sent by an external IP camera, the web server handles the api requests and serves it to the frontend, the frontend interface then provides a user-friendly interface for monitoring and configuration. The notification service delivers the alert through the Telegram API and the alarm system.

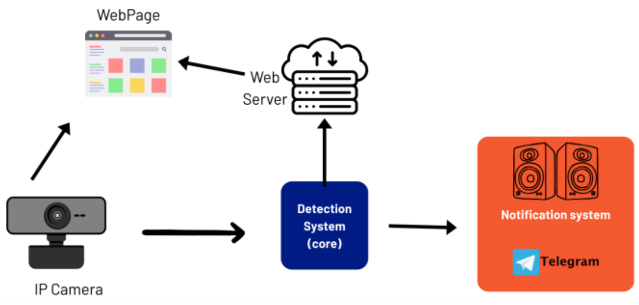

Fig. 1. Overall System Architecture

B. Training the Deep Learning Model

AI and its implementation: Our journey with AI began after noticing how much artificial intelligence has become relevant in the everyday life of people, from smart assistants like Siri to security systems that can recognize faces and objects in real time. We were particularly interested in how AI could spot and recognize patterns that traditional sensors often missed. This advantage it had over traditional sensors motivated us to include AI directly into our fire and security system.

We leveraged computer vision models to achieve this, training a CNN based on YOLOv8 that enables our camera to analyze live video and images while detecting early signs of fire and smoke. Our approach builds on research demonstrating that CNNs excel at real-time fire detection in surveillance footage. As noted in the study, "deep learning methods such as CNNs outperform traditional vision techniques in complex environments by automatically learning high-level feature representations."

By leveraging these AI strategies, our system became smarter, faster, and more accurate in detecting fire hazards and potential security threats. To develop a reliable fire detection solution, we trained a new model using a custom dataset.

An open-source dataset, FASDD_CV, containing 95,314 images, was obtained from the Science Data Bank website

[10]. To improve the diversity and robustness of the training data, data augmentation techniques such as flips, brightness adjustments, and cropping were applied, increasing the dataset size to approximately 160,000 images. The augmentation pipeline was applied before training.

The YOLOv8 architecture was selected for the object detection tasks. The model was trained on the Ultralytics hub platform, leveraging cloud GPUs and preconfigured parameters. The model was trained on 100 epochs, using the Adam optimizer. Similarly, compact CNN architectures such as NasNet-A-OnFire and ShuffleNetV2-OnFire have been shown to balance detection accuracy with computational efficiency for real-time fire monitoring [11].

After training, the model was exported to the NCNN framework for deployment in resource-constrained environments. NCNN's lightweight architecture allows for fast and efficient inference on edge devices, making it well-suited for real-time detection systems. The team transitioned from TensorFlow Lite to NCNN to take advantage of these performance benefits.

C. Hardware Architecture

The INFERNO system is designed to run on a Raspberry Pi, making it accessible for widespread deployment. The hardware setup includes a Raspberry Pi with 8 GB of RAM, a Logitech C270 HD webcam, and an MQ-2 sensor for smoke and gas detection. Audio is handled by a WM8960 audio hat, while power is supplied by a SunFounder battery pack. The system also incorporates an I2C logic level converter, a fan and heatsink for cooling, and a breadboard with jumper wires to facilitate connections.

Fig. 2 shows that each component works with each other to receive the desired functionality of the project. Each component is labeled with a number for the sake of the reader to identify what each component is. Component 1 is a MQ-2 smoke and gas detector, component 2 is a I2C logic level converter, component 3 is the Local Area Network (LAN), component 4 is the Telegram service, component 5 is a Raspberry Pi (the heart of the project) with a WM8960 audio HAT module on top, component 6 is a personal computer, component 7 is a uninterruptible power supply, component 8 is the speakers, and component 9 is a 720p Logitech USB web camera.

It should be noted that while the MQ-2 sensor can detect a variety of gases or smoke, it cannot identify which is which. The purpose of our project is still achieved, though, as our goal is to notify the user of danger so that they can take immediate action for safety.

To begin, the Raspberry Pi supplies ground and a 3.3-volt power connection to the I2C logic level converter to enable its operation. The Raspberry Pi then supplies the MQ-2 sensor with 5.0 volts of power and ground. The I2C sensor then converts the digital output from MQ2 from 5.0 volts to 3.3 volts. The converter is essential as the smoke sensor outputs a 5.0-volt signal when smoke or combustible gas is detected. The Raspberry Pi input pins (GPIO pins) are designed to only take up to 3.3 volts, so the converter takes the 5.0-volt signal and steps it down to 3.3 volts.

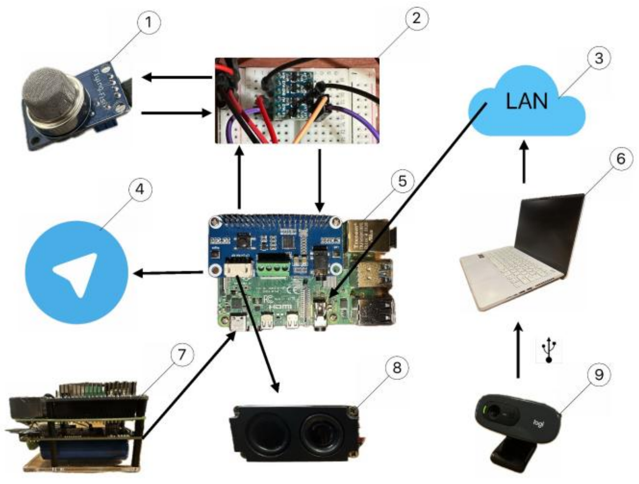

Fig. 2. Hardware Components

The MQ-2 Sensor features two types of output signals: (1) a digital out where the output is either on or off, and (2) an analog output where the amount the sensor detects can be transmitted to Raspberry Pi. The former is chosen as the digital output is the simplest approach and is adequate for the needs of this project. The MQ-2 Sensor also features a sensitivity adjustment for calibrating the sensor for accuracy. The Logitech Webcam captures the feed and sends it over the Local Area Network to the Pi for processing, and the PI sends an alert to Telegram and the speakers. An uninterruptible power supply (UPS) sits below the Raspberry Pi and powers the Pi via USB-C. The UPS is 4000 milliamp hours and also charges via USB-C. Finally, the WM8960 audio HAT module has a digital-to-analog converter and an amplifier to output alarm audio and text-to-speech messages to the speakers. Each speaker is capable of 5 watts output at 8 ohms, but the audio Hat is only capable of 1 watt per channel, so the speakers will be running at 1 watt. The whole hardware is put in a wooden box to protect it from external elements, as seen in Fig. 3.

D. Software Architecture

The detection system is the core component responsible for processing video frames and identifying potential hazards. It is coded in Python and uses OpenCV, YOLO, and Google's Mediapipe for detections. OpenCV is used for collecting the frames from the remote IP stream, and then it resizes these frames to 320 * 320 and passes them to the YOLODetector, FaceDetector, and Motion Detector classes. These classes then process the frames for predictions. The detection system operates in a continuous loop, capturing frames, processing them, and generating an alarm.

The fire and smoke detection involves taking the resized frames and passing them through the NCNN model in a loop. Every frame is analyzed, and a prediction is given. The confidence threshold for this application is 65% with a 55% IoU threshold. Meaning that the AI should be at least 65% sure before it triggers an alarm, and any prediction below that is discarded. This helps reduce false positives. Intersection over Union (IoU) helps to show the best fit box for the prediction by dropping the lower confident box if boxes overlap by 55%. These configurations help filter out weak and noisy detections while prioritizing high-confidence detections. Specifically, our model is based on the YOLOv8 backbone, which offers improved accuracy and faster inference speed.

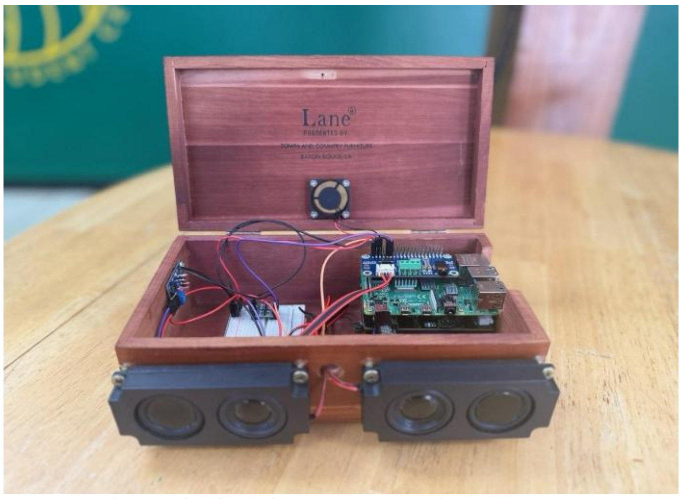
Fig. 3. A hard wooden Case to House the Components

The code in Fig. 4 takes the resized frames and generates predictions, then post-processes the output to extract the bounding boxes, classes, and confidence scores. It applies both confidence and IoU thresholds to filter out low-quality detections and finally returns the refined detection results for further processing.

```
class YOLODetector:
    def __init__(self, model_path, conf_threshold=0.65,
iou_threshold=0.55):
        """Initialize YOLO object detector"""
        self.model = YOLO(model_path)
        self.CONF_THRESHOLD = conf_threshold
        self.IOU_THRESHOLD = iou_threshold

    def detect(self, frame):
        """Detect objects in frame using YOLO"""
        height, width = frame.shape[:2]
        results = self.model.predict(frame, imgsz=width,
conf=self.CONF_THRESHOLD, iou=self.IOU_THRESHOLD)
        return results[0].boxes
```

Fig. 4. The YOLODetector Class Handles Fire and Smoke Detection

For the intruder and face detection component of this project, this project drew inspiration from the growing role of AI in modern security monitoring systems. Previous research has shown that facial feature recognition can accurately track human presence and activities in real time, which motivated this project to adopt a similar approach. Beyond live monitoring, this project can also zoom in, capture intruder faces, and store these images for later reference. This project utilizes the MediaPipe FaceMesh model [12] for detecting facial landmarks, building on its proven applications in prior studies. FaceMesh is a cross-platform framework for multimodal machine learning tasks, offering accurate, locally running face detection models that do not require internet connectivity and are well-suited for deployment on a Raspberry Pi.

As shown in Fig. 5, the face detection process begins by initializing the Mediapipe FaceMesh with an 80% confidence threshold, then takes the image frame in RGB format and applies face detection. It processes the frames and returns the face landmarks whenever faces are successfully detected.

```
# Face detector
class FaceDetector:
    def __init__(self, min_detection_confidence=0.8):
        """
        Initialize MediaPipe Face Mesh for more accurate and fast face
detection.
        """
        self.mp_face_mesh = mp.solutions.face_mesh
        self.face_mesh = self.mp_face_mesh.FaceMesh(
            static_image_mode=False,
            max_num_faces=5,  # Adjustable based on your needs
            min_detection_confidence=min_detection_confidence,
            min_tracking_confidence=0.5
        )

    def detect(self, frame_rgb):
        results = self.face_mesh.process(frame_rgb)
        return results.multi_face_landmarks if results.multi_face_landmarks
else []
```

Fig. 5. The FaceDetector Class Handles Face Detection in Live Camera Streams

The motion detection is implemented using the frame differencing approach, which compares the previous frame with the present frame to tell if movement has happened because if there was motion, there will be a significant difference in the frames. This method is efficient, and it helps detect motion in the video feed.

In Fig. 6, the motion detection process involves converting the image to grayscale and applying a Gaussian filter to reduce noise, computing the absolute difference between the current and previous frames, and applying a threshold to determine whether a significant change has occurred. It then identifies motion regions by filtering based on a minimum area and returns the bounding boxes of the detected motion regions.

The detection system Python file also includes a TelegramService class to handle notifications, a Camera class to handle the incoming stream and a ConfigManager to handle system configuration. The ConfigManager helps to make the system parameters customizable for different deployment environments.

To further solidify INFERNO's reliability, this project implemented a thresholding system. This logic is applied exclusively to fire and smoke detection, leaving intruder detection unaffected. The system is designed so that it does not trigger the alarm at the first sign of smoke or fire; instead, it verifies whether the threat persists for n consecutive detections before activating the alarm. In Fig. 7, which shows a snippet of the JSON configuration file, the 'alarm_threshold' is set to 3. Only after the third detection does the system trigger the alarm sound, immediately sending a notification to the user along with captured images. Although this might seem time-consuming, the detection cycles occur within split seconds, ensuring that the system remains both fast and reliable without compromising accuracy. This approach enhances INFERNO's functionality while reducing the occurrence of false fire alarms.

```
class MotionDetector:
    def __init__(self):
        """Initialize motion detector with better memory and
performance"""
        self.fgbg = cv2.createBackgroundSubtractorMOG2(history=100,
varThreshold=25)
        self.min_area = 1000  # Minimum area to trigger motion detection
        self.last_motion_time = 0
        self.motion_cooldown = 2.0  # Time in seconds before motion is
considered gone

    def detect(self, frame):
        """Detect motion in frame with automatic cooldown"""
        current_time = time.time()

        # Apply background subtraction
        # Convert to grayscale for better performance
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        fgmask = self.fgbg.apply(gray)

        # Apply thresholding to remove noise
        thresh = cv2.threshold(fgmask, 200, 255,
cv2.THRESH_BINARY)[1]

        # Find contours
        contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)

        # Check for significant motion
        motion_detected = False
        for contour in contours:
            if cv2.contourArea(contour) > self.min_area:
                motion_detected = True
                self.last_motion_time = current_time
                break

        # Return true if motion was detected recently (within cooldown
period)
        return motion_detected or (current_time - self.last_motion_time <
self.motion_cooldown)
```

Fig. 6. The MotionDetector Class Detects Motion Using Frame Differencing

```
"system": {
  "camera_index": 0,
  "camera_url": "http://192.168.5.166:5000",
  "detection_interval": 0.5,
  "face_save_interval": 1.0,
  "alarm_threshold": 3,
  "max_saved_faces": 50,
}
```

Fig. 7. The N threshold

The web server is built with Flask, a lightweight Python web framework. It serves the frontend and provides RESTful APIs for communication between the frontend and the detection system backend. The web server has endpoints for handling requests related to configuration, status, and alerts. It also serves HTML, CSS, JavaScript, and image files, and manages text-to-speech (TTS) requests. For TTS, gTTS is implemented to notify people in the vicinity of the device. On the webpage, this project provides a box for sending TTS requests for the Piper model. When a user submits a request, it can be used to deter intruders by playing a warning message.

As shown in Fig. 8, the TTS process begins by retrieving a request from the queue, then uses gTTS to synthesize the text and save the output as temp_tts.mp3 in the voice directory. The generated audio is played using the mpg321 command-line player, the temporary file is deleted afterward, and finally, tts_queue.task_done() is called to indicate that the task has been completed.

```
# Function to process TTS requests from the queue
def process_tts_queue():
    voices_dir = os.path.join(os.getcwd(), 'voices')
    os.makedirs(vices_dir, exist_ok=True)
    while True:
        text = tts_queue.get()
        if text is None:
            break
        try:
            print(f"[TTS] Speaking: {text}")
            # Use gTTS to synthesize audio
            tts = gTTS(text=text, Iang='en')
            temp_mp3 = os.path.join(voices_dir, "temp_tts.mp3")
            tts.save(temp_mp3)

            # Play the audio file using mpg321
            subprocess.run(["mpg321", "-q", temp_mp3])

            # Remove the temporary audio file
            os.remove(temp_mp3)
        except Exception as e:
            print(f"[TTS Error] {str(e)}")
        finally:
            tts_queue.task_done()

# Start a background thread to process the TTS queue
tts_thread = threading.Thread(target=process_tts_queue,
daemon=True)
tts_thread.start()
```

Fig. 8. Asynchronous Text-to-Speech (TTS) Queue Processor Utilizing the gTTS Library and Background Threading

The frontend interface is a responsive web dashboard built with HTML, CSS, and JavaScript, providing a user-friendly platform for monitoring system status, viewing detection results, and configuring settings. It features a status dashboard that displays the system's current condition, a video feed for real-time streaming from the IP camera, and a configuration panel for adjusting detection parameters. Additionally, it includes a Detection Gallery that showcases recent fire and smoke detections with bounding boxes, a Face Gallery for monitoring recent faces for security purposes, and a TTS interface that allows users to send requests for voice announcements using Piper.

The data flow within the system begins with the detection module capturing frames from the IP camera. These frames are then preprocessed by resizing and passed through several detectors, including YOLO, a face detector, and a motion detector. When a hazard is identified, the system generates alerts that are sent to notification services and may also trigger an alarm in the case of fire or smoke. The web server exposes API endpoints that allow the frontend to access status updates and detection results, which the frontend periodically requests to refresh the dashboard. Any configuration changes made by the user through the frontend are sent to the web server and then forwarded to the detection system.

IV. Evaluation And Results

To evaluate the effectiveness of the INFERNO security system, a series of practical experiments were conducted, including live demonstrations in class. The system was deployed on a Raspberry Pi overclocked to 2 GHz to provide additional processing power. Experiments were performed under various lighting conditions, including daylight, dim light, and extremely bright light, to assess the system's robustness.

For fire and smoke detection, we used a diverse set of inputs, including controlled fire samples such as matches and lighters, simulated smoke obtained from a YouTube video, video datasets featuring real fire and smoke incidents, and non-fire or non-smoke objects with similar visual characteristics such as red objects, steam, or clouds.

For face detection, we used a variety of inputs, including multiple individuals with different facial characteristics, various facial orientations, and a range of lighting conditions.

Detection performance encompasses several aspects, including the effectiveness of fire and smoke detection, the accuracy of face detection, and the reliability of motion detection.

In evaluating fire and smoke detection performance, we observed that the system initially struggled with sudden changes in lighting, which could trigger false alarms. To address this, we trained a new model using data augmentation to make it more robust. The latest model achieves a mAP50 of 80.4%, as shown in Fig. 9, representing a significant improvement in detection accuracy and performance. This enhancement ensures that the model can effectively handle abrupt lighting changes without compromising detection reliability.

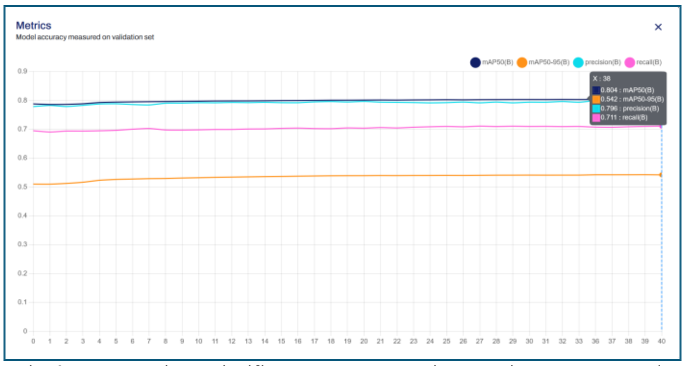

Fig. 9. Representing a Significant Improvement in Detection Accuracy and Performance

We also tested the model across several environments like car fire, wildfire, and home fire,and it performed well, as seen in Fig. 10, 11, and 12 below. It shows that the model is generalised and able to capture fire in different scenarios and environments. Model versatility is crucial for real life applications, ensuring that the system can be deployed in a wide range of settings without requiring environment-specific adjustments.

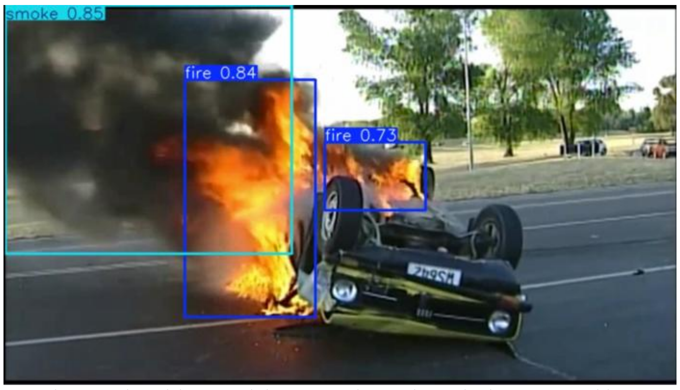

Fig. 10. The Vision Model Detects Fire and Smoke in a Car Accident

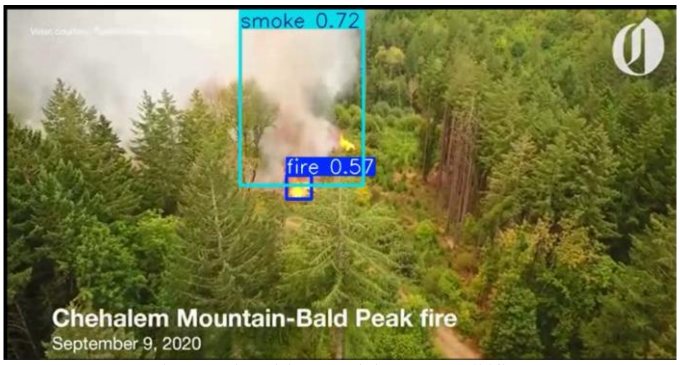

Fig. 11. The Vision Model Detects Wildfires

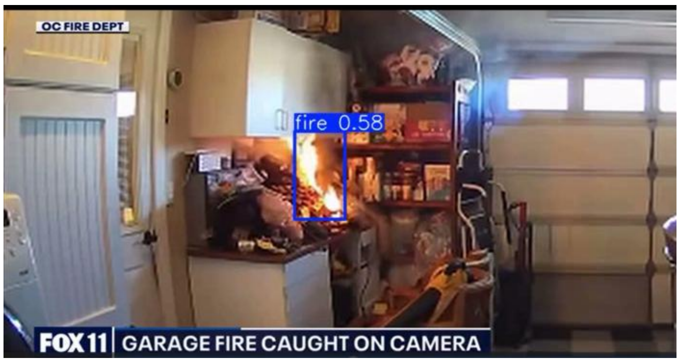

Fig. 12. The Vision Model Detects House Fires

For face detection the performance is subpar as we used the least accurate model due to resource constraint. Due to this the model picks up features that are not faces. The solution to this would be picking a stronger MediaPipe's model but that would be too much for our Pi to handle.

For motion detection the algorithm performed really well as it easily tells movements and alerts the user. It could detect both fast moving and slow-moving objects, which is good, it could even detect light flickering.

In terms of frame processing speed, a crucial requirement for any real-time vision system is its ability to handle enough frames to ensure timely and reliable alarms. Our system's performance data shows that on a Raspberry Pi with 8 GB of

RAM, it processes approximately 400 frames per minute, which is a strong result for an edge device.

In detection processing, this refers to the time required to analyze each frame passed to the fire and smoke detector. The processing time is approximately 270 ms, which is remarkably fast for a Raspberry Pi, and includes the total time spent on pre-processing, processing, and post-processing.

For system reliability, we tested the system by running it continuously for seven hours, during which it operated without any issues. There were no overheating problems, thanks to the use of a fan and heatsink, and network connectivity remained stable due to a reliable internet connection. Additionally, if the camera disconnects, the system automatically attempts to reconnect, ensuring continuous operation.

Our system outperforms traditional detectors, and many other computers vision systems because, being developed by a small group and running on an inexpensive Raspberry Pi, it was carefully optimized and overclocked for speed. It delivers notifications in under three seconds after detection, includes an additional onboard security system for intruder detection, and provides an intuitive web interface for real-time monitoring. Moreover, it can connect to virtually any camera, even an IP camera located in another country, allowing it to detect fire and smoke from remote image feeds.

Commercial systems typically offer higher detection accuracy and performance, but they are usually more expensive and often require paid subscriptions for cloud storage or additional features. They also tend to be less flexible than INFERNO, which is open source. INFERNO provides a balanced combination of performance, cost, and adaptability, making it well-suited for small to medium-sized deployments in homes or offices where commercial solutions may be cost-prohibitive.

V. Conclusion and Future Work

This paper presented INFERNO, a comprehensive real-time fire, smoke, and intruder alarm system with integrated notification capabilities. The system leverages computer vision techniques, with Yolo, Mediapipe, and OpenCV to identify potential hazards in video feeds.

The key highlights of this work include its ability to detect multiple hazards such as fire, smoke, faces, and motion within a single integrated system. It can be deployed almost anywhere because it is designed for the Raspberry Pi, eliminating the need for expensive hardware. The system also features integrated notifications through Telegram, speakers, and text-to-speech to ensure timely alerts. A user-friendly web dashboard simplifies monitoring and configuration. Additionally, the open-source nature of the system encourages community contributions, customization, and adaptation for specific user needs.

Our experimentation demonstrates that our system has high accuracy across various conditions, with good processing speed on edge devices. It maintains stable performance after prolonged running and is reliable.

While the system has limitations related to lighting conditions and processing constraints, it shows a significant advancement in reliable fire safety monitoring solutions. This system could help you build your own personal security system without worrying about how your data is handled, as you will own your data.

As fire safety remains a critical concern in both residential and industrial settings, INFERNO has potential to significantly reduce response time. While minimizing the loss of lives and properties.

Future work will focus on expanding the system to support multiple cameras, further optimizing performance on the Raspberry Pi, and incorporating self-learning capabilities through a feedback mechanism. Additionally, it will include video saving and playback features, integration with smart home systems, and the development of a more advanced neural network to achieve higher detection accuracy. Future expansion could also integrate UAV-IoT frameworks for wildfire detection, offering broader coverage and faster alerting capabilities [13][14]. This project has built our experience with Python and web development, and we cannot wait to work more on this with the support of new contributors to this open-source project.

References

[1] National Fire Protection Association, “Fire Loss in the United States During 2020,” *NFPA Research*, September 2021.

[2] P. Panindre, S. Acharya, N. Kalidindi, and S. Kumar, "Artificial-Intelligence-Integrated Autonomous IoT Alert System for Real-Time Remote Fire and Smoke Detection in Live Video Streams," *IEEE Internet of Things Journal*, vol. 12, issue: 21, pp. 45133–45149, November 2025, DOI: 10.1109/JIOT.2025.3598979.

[3] D. Gragnaniello, A. Greco, C. Sansone, and B. Vento, “Video Fire Recognition Using Zero-shot Vision-Language Models Guided by a Task-aware Object Detector,” *ACM Transactions on Multimedia Computing, Communications and Applications*, vol. 21, issue 10, pp. 1–24, October 2025, DOI: 10.1145/3721291.

[4] K. Muhammad, J. Ahmad, I. Mehmood, S. Rho, and S. W. Baik, "Convolutional Neural Networks Based Fire Detection in Surveillance Videos," *IEEE Access*, ISSN: 2169-3536, vol. 6, pp. 18174–18183, March 2018. DOI: 10.1109/ACCESS.2018.2812835.

[5] P. Kumar, A. Prathyusha, N. HimaBindu, and P. Manaswini, "Intelligent IoT System with AI and ML Integration for Smart Home Automation," *2025 5th International Conference on Trends in Material Science and Inventive Materials (ICTMIM)*, Kanyakumari, India, pp. 1107–1114, April 2025, DOI: 10.1109/ICTMIM65579.2025.10988075.

[6] K.P. Yang, G. Kiepper, B. Henry, and R. Hunter, “Modular Architecture for IoT Home Automation and Security Surveillance,” *Journal of Multidisciplinary Engineering Science and Technology (JMEST)*, ISSN 2458-9403, vol. 5, issue 11, pp. 8978–8982, November 2018.

[7] V. Thananant and C. Mokarat. “An IoT Based Intruder and Smoke Monitoring System,” *In Proceedings of the 2021 7th International Conference on Computing and Artificial Intelligence (ICCAI '21)*, Association for Computing Machinery, New York, NY, USA, pp. 471–478, 2021, DOI: 10.1145/3467707.3467778.

[8] A. Vu-Quoc, B. Nguyen-Phuc, L. Van-Thien, C. Dang-Le-Bao, T. Nguyen-Khanh, and Q. Le-Trung, "Research and Development of Attendance System for Smart Classroom Using Bluetooth Low Energy and Facial Recognition Technologies," *2024 RIVF International Conference on Computing and Communication Technologies (RIVF)*, pp. 266–270, December 2024, DOI: 10.1109/RIVF64335.2024.11009036

[9] K. P. Yang, P. McDowell, P. Dolan, C. Otts, G. Chenevert, and C.Tunstall, “SelfieBot: A Robot to Detect, Photograph, and Tweet Smiles,” *International Journal of Engineering Research & Technology (IJERT)*, ISSN 2278-0181, vol. 8, issue 10, pp. 387–390, October 2019.

[10] M. Wang, P. Yue, L. Jiang, D. Yu, and T. Tuo, “An Open Flame and Smoke Detection Dataset for Deep Learning in Remote Sensing Based Fire Detection.” *Geo-spatial Information Science*, vol. 28, issue 2, pp. 511–526, January 2025, DOI: 10.1080/10095020.2024.2347922.

[11] W. Thomson, N. Bhowmik, and T. P. Breckon, "Efficient and Compact Convolutional Neural Network Architectures for Non-temporal Real-time Fire Detection," *2020 19th IEEE International Conference on Machine Learning and Applications (ICMLA)*, Miami, FL, December 2020, pp. 136–141, DOI:10.1109/ICMLA51294.2020.00030
[12] B. Thaman, T. Cao, and N. Caporusso, "Face Mask Detection Using MediaPipe Facemesh," *2022 45th Jubilee International Convention on Information, Communication and Electronic Technology (MIPRO)*, Opatija, Croatia, pp. 378–382, May 2022, DOI: 10.23919/MIPRO55190.2022.9803531.
[13] K.P. Yang, P. McDowell, P. Devkota, S. Pradhan, R. Bhandari, and Z. Madewell, "Detecting Gas Leaks: A Case Study in IoT Technologies," *European Journal of Engineering and Technology Research (EJ-ENG)*, ISSN 2736-576X, vol. 6, issue 7, pp. 103–106, December 2021.
[14] O. M. Bushnaq, A. Chaaban, and T. Y. Al-Naffouri, "The Role of UAV-IoT Networks in Future Wildfire Detection," *IEEE Internet of Things Journal*, vol. 8, issue. 23, pp. 16984–16999, December 2021, DOI: 10.1109/JIOT.2021.3077593.