Implementing real-time wildfire detection using lightweight object-detection models and machine vision sensor on Raspberry Pi 5: Fireframe, a practical framework

Jukka Joutsalainen, Maxim Vitikainen, Juuso Lehrbäck, Alexander Goldhill, Anna-Maria Raita-Hakola

Faculty of Information Technology, University of Jyväskylä, Mattilanniemi 2, Jyväskylä Finland
(jukka.a.joutsalainen, maxim.t.vitikainen, juuso.e.lehrback, alejgold, anna.m.hakola)@jyu.fi

Keywords: Wildfire smoke detection, Edge-device deployment, Fire surveillance, Framework for real-time smoke detection, Object detection

Abstract

The research field of small, lightweight object-detection models that are capable of real-time monitoring, particularly for the detection of wildfires, is highly popular. However, a quick overview of the literature reveals that while most research suggests lightweight models, it does not report results from tests conducted on platforms with limited computational power or frameworks that might enable practical applicability of the techniques. This leaves the algorithms without real-time usability tests. This study addresses the research gap, aiming to provide a robust and low-cost framework (Fireframe) for edge device deployment for wildfire smoke detection. Fireframe combines hardware (Raspberry 5 computer and Basler machine vision camera) and a trained object detection model with tasks that are performed in an operational loop (main thread). It can simultaneously record a live stream, analyze whether wildfire smoke is present, and display the findings. Fireframe is tested using two lightweight models (YOLOv10 and MobileNetV3), and the findings confirm its suitability for simulations and real-life action.

1. INTRODUCTION

Forest fires are among the most devastating natural disasters, leading to large-scale environmental degradation, loss of biodiversity, and severe public health implications. To highlight human impact, global exposure to landscape fire smoke contributes to an estimated 339,000 premature deaths annually (Johnston et al., 2012). The economic factor caused by the wildfire damage is also substantial. The United Nations Environment Program (UNEP) reports that the cost of wildfire damages, including suppression, property loss, and health impacts, continues to rise due to climate change and increased human encroachment to forested areas (UNEP and GRID-Arendal, 2022). In addition, forest fires have an impactful contribution to greenhouse gas emissions and long-term soil degradation, thus affecting global climate change and hampering reforestation efforts.

Wildfires typically spread rapidly, making early and reliable detection essential for effective mitigation. Traditional approaches such as satellite-based observation and ground-based surveillance often suffer from latency and limited spatial resolution, which can hamper early-stage detection (Park and Ko, 2020). In response, Unmanned Aerial Vehicles (UAVs) combined with machine vision solutions, particularly deep learning models, have become widely proposed in wildfire and smoke detection applications (Chen et al., 2023b). However, the operations beyond visual line of sight (BVLOS) are strictly controlled with legislation (see, for example, (Ariante and Del Core, 2025)). Thus, for these situations, low-cost solutions are needed that could be used both ways as an effective UAV payload as well as an inexpensive small device that could be installed stationary in areas that are not suitable for UAVs.

Although there is a lot of research on the lightweight wildfire detection and monitoring research (Barmpoutis et al., 2020, Moumgiakmas et al., 2021, Abid, 2021, Bouguettaya et al., 2022, Gaur et al., 2020, Surya, 2020, Saleh et al., 2024, Carta et al., 2023), relatively few new technological solutions have been put into practice (Barmpoutis et al., 2020, Moumgiakmas et al., 2021, Bouguettaya et al., 2022, Carta et al., 2023, Bailon-Ruiz and Lacroix, 2020). Besides evaluating review articles from 2020-2024, we searched the web-of-science database to retrieve 148 forest fire detection and monitoring articles from 2010 to 2024. We performed a brief overview of them and separated articles that presented only algorithms from those that put algorithms into practice and made hardware implementations. We found 108 articles presenting only algorithms and 38 articles that tested algorithms in theory and practice. To conclude, the majority of the studies present the models as state-of-the-art, being computationally lightweight for real-time platforms, but without practical testing, these have fallen short of real-world experimentation.

1.1 Detecting smoke with deep learning

Convolutional Neural Networks (CNNs), which are designed for image analysis, are well-suited for this task due to their ability to automatically extract and learn spatial patterns such as smoke plumes and flame textures. Architectures like YOLOv3, YOLOv5, and YOLOv8 have been employed in UAV-based systems for real-time detection, showing improved accuracy and processing efficiency (Zhou et al., 2019, Ha et al., 2018, Yang et al., 2019).

Since smoke is often the first visual signal of a fire, and thus it is not typically present in a machine-learning imagery, there is a limited amount of relevant training data that is suitable for training deep-learning-based object detection methods (Raita-Hakola et al., 2023). To address the challenge of limited datasets and high model complexity, researchers have increasingly turned to transfer learning. Pre-trained CNNs allow efficient adaptation to new fire detection tasks, even with modest data (Best et al., 2020). These models are typically trained on large-scale datasets like ImageNet, which provides millions of labeled images across a wide variety of categories, enabling stronger generalization capabilities (Krizhevsky et al., 2017). However, when

This contribution has been peer-reviewed. The double-blind peer-review was conducted on the basis of the full paper.
https://doi.org/10.5194/isprs-annals-X-2-W2-2025-81-2025 | © Author(s) 2025. CC BY 4.0 License.

fine-tuning models for new datasets, existing knowledge may be lost, a challenge referred to as forgetting. To mitigate this, Sathishkumar et al. (Sathishkumar et al., 2023) introduced a method combining transfer-learning with Learning without Forgetting (LwF). Noteworthy, this was achieved by leveraging relatively lightweight and efficient pre-trained models such as VGG16, InceptionV3, and Xception, reducing training complexity while maintaining performance across chosen datasets.

Integrating deep learning with UAV platforms enables early detection of subtle fire indicators like smoke plumes, even under challenging environmental conditions. These models aim to address key issues such as identifying small and dispersed smoke patches and differentiating them from visually similar backgrounds like clouds or mist (Wu et al., 2023). Nonetheless, accurate smoke detection remains difficult due to its inherent variability in transparency, shape, and motion. Environmental factors such as fog, rain, low light, and interference from industrial emissions can lead to high false positive rates (Chen et al., 2023a, Hosseini and Choi, 2022).

However, many state-of-the-art deep learning architectures are computationally intensive, requiring significant memory and processing power, making them unsuitable for deployment on edge devices such as UAVs, which are constrained by limited battery life, onboard storage, and real-time processing capabilities. Consequently, there is a growing need for lightweight, resource-efficient models that can operate reliably on embedded systems without compromising accuracy (Qu, 2022). In parallel, existing fire detection methodologies often fall short when applied to ecologically complex domains such as boreal forests, where dense canopies, high humidity, and atmospheric variability pose unique challenges. These environmental complexities can degrade the performance of generic models, thus underscoring the domain-specific solutions.

1.2 Hypothesis, research questions and main contributions

Given the increasing prevalence of wildfires as a global problem, we will need low-cost technological solutions in the future that can be rapidly deployed, both static and on-board UAVs, for example. We hypothesize that a low-cost framework can ease the real-life implementations of wildfire detection systems. In the future, UAVs have high potential for autonomous wildfire monitoring, especially in smaller countries such as Finland, but while the legislation is not ready for autonomous UAVs, stationary solutions are equally important. By keeping this in mind, we aim to answer the following research questions:

A) *How to create a low-cost framework that can be used for testing and real-time use (onboard and stationary) of object detection methods for wildfire detection?*
B) *What kind of operational loop (main thread) is needed for Raspberry Pi to run machine vision camera, object detection algorithm and result visualization tasks?*

Our main contribution is a robust and low-cost framework (Fireframe) for edge device deployment for wildfire smoke detection, which addresses the challenges of real-time wildfire smoke detection and can be used for testing novel object detection algorithms as well as be implemented into a real-life scenarios (stationary or UAV-based).

The article is organized as follows: Section 2 presents the requirements and processes of our Fireframe framework. Section 3 shows how we constructed and compared three lightweight object detection methods for it and tested Fireframe in practice. The results of our tests can be found in Section 4, and the discussion and conclusions are in Section 5.

2. MATERIAL AND METHODS

This section contains the Fireframe framework's details of hardware, tasks, workflow, and object detection model architectures that were used in our edge device deployment.

2.1 Requirements for framework's edge device deployment

A wildfire smoke detection system, based on Fireframe can be built using the following hardware:

- Raspberry Pi 5 8 GB single-board computer (64-bit quad-core Arm Cortex-A76 processor running at 2.4GHz, with a basic Ubuntu operating system)
- Basler Dart daA1920-160uc machine vision camera (Sony IMX392 CMOS sensor, 2.3 MP resolution, 160fps)
- Evetar S-mount 4mm F1.6 1/2" lens
- The programming language was Python 3.9, which was used for controlling hardware, analysis and visualizations.
- Two basic computer screens with no specific requirements
- Keyboard and mouse for the phases when the system is deployed

Figure 3 visualizes the Basler Dart camera and Raspberry Pi computer. As can be seen from the images, the computer is connected with a USB-3 and USB-C cables, which can give a hint of the physical size.

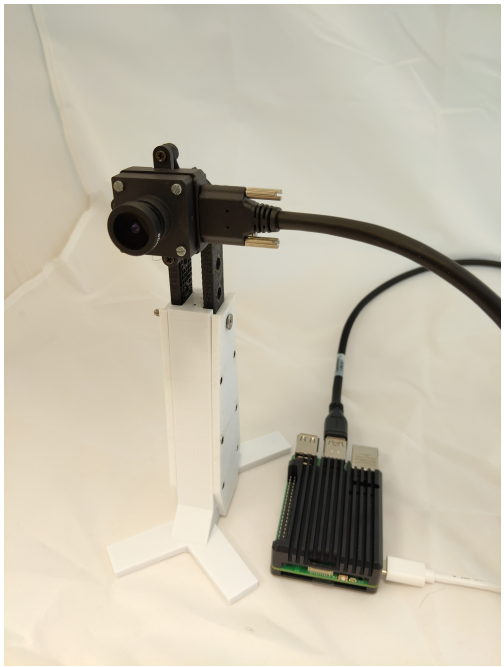

Figure 1. Hardware aspects of the framework: Raspberry Pi 5 single-board computer connected to a Basler RGB machine vision camera with Evetar lens.

2.2 Fireframe tasks

Despite limited computational resources, a Fireframe-based system is capable of performing the complete detection pipeline locally with the following tasks:

This contribution has been peer-reviewed. The double-blind peer-review was conducted on the basis of the full paper.
https://doi.org/10.5194/isprs-annals-X-2-W2-2025-81-2025 | © Author(s) 2025. CC BY 4.0 License.

- Load a pre-trained neural network model into memory
- Capture live video frames from a camera system
- Preprocess the captured data as required by the model
- Perform inference using the preloaded model
- Visualize and display the inference results in near real-time
- Repeat this cycle continuously with low latency.

By performing the entire processing loop on a single device, this approach eliminates the need for additional processing on external systems. This makes the system suitable for real-world deployment scenarios, such as static monitoring stations (e.g., fire posts or surveillance towers). Once legislation permits, such autonomous systems could be integrated into mobile platforms (e.g., drones or autonomous vehicles) that transmit actionable, processed data to remote control centers.

2.3 Framework's process flow

The main thread of the application, illustrated in Figure 2, follows a structured and repeatable pipeline:

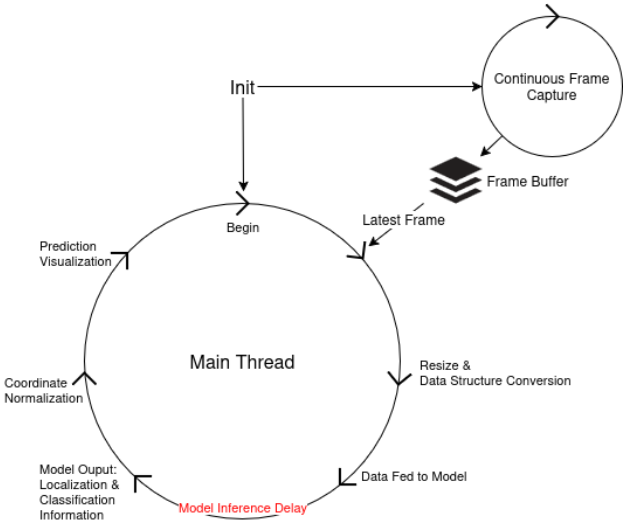

Figure 2. Conceptual overview of the main thread process for edge-based smoke detection.

1. **Initialization:** The system initializes using predefined configuration parameters. This includes setting up the camera interface and loading the neural network model into memory for subsequent inference tasks.
2. **Frame Acquisition:** The main thread continuously fetches the latest frame from the camera system's frame buffer.
3. **Preprocessing and Resizing:** The captured frame is resized to match the input dimensions expected by the network model. YOLO-based libraries handle this internally, but for custom architecture, resizing is implemented manually within the main thread.
4. **Inference:** The preprocessed frame is fed into the model for inference. The model produces detection results indicating object presence and type.
5. **Model Output:** The output includes bounding box coordinates, class probabilities (e.g., "smoke" vs. "no smoke"), and confidence scores, all represented as floating-point values.
6. **Coordinate Normalization and Visualization:** The raw bounding box coordinates are normalized to pixel values relative to the original frame dimensions. A visualization step overlays the detection results—bounding boxes and class labels onto the original image using a visualization library.
7. **Output Utilization:** The annotated output may be displayed on a monitoring interface and/or stored for further analysis, as illustrated in Figure 4. After this step, the process returns to frame acquisition, continuing the loop.

2.4 Data and object detection architectures for tests

A Fireframe system deploys light-weight object detection algorithms. In our study, we used data from Boreal forests, and tested three lightweight algorithms, of which two were deployed into Fireframe framework after comparison.

2.4.1 Data We utilized the Boreal Forest Fire dataset's subset A in our tests, which is available at (Pesonen et al., 2025). Boreal Forests dataset is an UAV-collected wildfire detection and smoke segmentation dataset comprising aerial photographs and video clips of forest fires and non-fire forest regions. The data is sourced from four different municipalities in Finland. The dataset contains UAV-captured images under different lighting conditions and from various angles, being annotated with bounding boxes around the smoke clouds.

2.4.2 Architectures Three different architectures were used in the final testing phase: YOLOv8, YOLOv10, and MobileNetV3, the latter of which had a custom detection head attached. The YOLO models were developed using the Ultralytics library (Ultralytics, 2023b), while the MobileNetV3-based model was implemented with Keras (Chollet et al., 2015). Model optimization, training, and comparison were performed on a 28-core Linux server (x86_64) in a non-parallel computing setup.

2.4.3 Custom detection head A pre-trained MobileNetV3 with ImageNet weights, served as the backbone and was followed by a flatten-layer branching into two heads: one for bounding box coordinates and another for classification and confidence estimation. The architecture is visualized in Figure 3.

The heads consisted of dense layers, from which bounding-box output is achieved with reshape-function, whilst confidence and class predictions are obtained via lambda functions

$$\lambda_1(x) = -x \quad \text{and} \quad \lambda_2(x) = 1 - x \tag{1}$$

where they apply lightweight output transformations. The function $\lambda_1(x)$ inverts the binary class prediction, emphasizing the negative class in the loss function, and $\lambda_2(x)$ complements the confidence score, providing a direct measure of uncertainty.

The architecture of the box head for the Keras model is presented in Table 1, while the class head architecture is provided in Table 2.

Box head		
Neuron count	Type	Activation
256	Dense	Swish
64		
32		
16		
4		Sigmoid

Table 1. Box head architecture for the Keras model

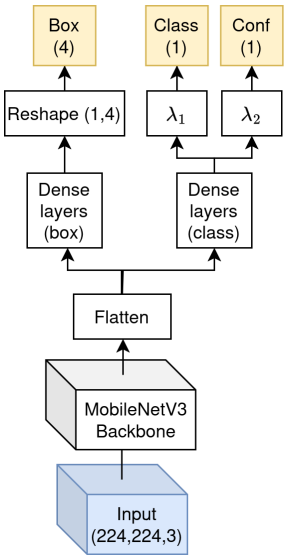

Figure 3. Diagram of MobileNetV3-based architecture

Class head		
Neuron count	Type	Activation
128	Dense	ReLU
	Dropout	
64	Dense	ReLU
	Dropout	
1	Dense	Sigmoid

Table 2. Class head architecture for the Keras model

3. Deployment

After the hardware, tasks, workflow, data and architecture are selected, it is time for deployment. The process phase covers steps from data pre-processing and model training and optimization to model evaluation and finally implementing the algorithms, camera controls and main thread to the Raspberry Pi computers.

3.1 Preprocessing

We used, 4760 Boreal Forests Fire images, in our trials. The images contained frames with and without smoke. Since, they were captured in several municipalities of Finland, we left all Ruokolahti area images into our test set (total 1822), while the remaining images (2938), were randomly divided into training and validation sets, with 20% allocated to validation and the rest to training. Since the Boreal Forest Fire data is annotated in the YOLO-compatible center-based XY, W, H (center coordinate, width, height) format. However, for Keras, the relative XY, XY (upper-left corner, bottom-right corner coordinates) box format was adopted.

Before training, all images were resized to 224 × 224 pixels for the MobileNetV3-based model and to 640 × 640 pixels for the YOLO models. To improve generalization, data augmentation was applied. In Keras, this was done explicitly using KerasCV layers, mainly `RandomFlip` and `JitteredResize`. These performed horizontal or vertical flips and randomized resizing, followed by cropping and padding to the target size.

For YOLO models, Ultralytics' built-in augmentation pipeline was used, which automatically applies a wide range of random transformations. These include color jitter (hue, saturation, brightness), scaling, horizontal translation and flipping, mosaic augmentation, random erasing, and cropping. The specific augmentations and their parameters are documented in Ultralytics' official documentation (Ultralytics, 2023a). These augmentations aimed to simulate diverse real-world conditions with unpredictable noise, and to improve the model's robustness against unseen variations

3.1.1 Hyperparameter selection, training and optimization

Five hyperparameters were chosen to be optimized for YOLO models: optimizing algorithm, initial learning rate (lr0), final learning rate factor (lrf), weight decay and momentum. Learning rate and momentum are mentioned in YOLO documentation as important parameters. The final learning rate factor was chosen due to the direct connection to the initial learning rate, and the optimizing algorithm is assumed to have a notable effect on the result. Weight decay, which basically refers to regularization, is an important parameter to control overfitting.

In Keras, some hyperparameters were the same as in Ultralytics, but due to varying implementations of the optimization algorithms and own custom detection head architecture, some different hyperparameters were chosen: an optimizing algorithm, number of dense layers, dropout probability in chosen layers and activation function in localization part of the detection head, learning rate, global clip norm and weight decay. The Global clip norm limits the maximum norm of gradients when updating the weights, which is essential for preventing exploding gradients. Dropout is another tool for avoiding overfitting ((Srivastava et al., 2014)). When the dropout probability was less than 0.05, it wasn't applied at all.

MobileNetV3	
Optimizer	RMSProp
Layers	5
Activation	Swish
Learning rate	2.61×10^{-5}
Global clipnorm	11.26
Weight decay	9.97×10^{-5}
Dropout	0 / 0.042
YOLOv10	
Optimizer	AdamW
lr0	2.20×10^{-4}
lrf	1.37×10^{-5}
Weight decay	7.83×10^{-5}
Momentum	0.803
YOLOv8	
Optimizer	Adam
lr0	4.25×10^{-4}
lrf	0.042
Weight decay	8.39×10^{-4}
Momentum	0.921

Table 3. Optimized hyperparameters for different models. Dropout was applied only to the classification head in MobileNetV3.

All three models were trained using transfer learning. The YOLO models transfer learning followed (Raita-Hakola et al., 2023), and the Keras-based MobileNetV3 model was pre-trained on ImageNet data (Deng, 2012), with its backbone frozen to preserve well-generalizing feature extraction. To calculate a generic loss metric, a mean squared error (MSE) of classes and

complete intersection over union (CIoU) of coordinates were summed using built-in Keras functions. CIoU was chosen as it offers a more in-depth metric for the accuracy of the coordinates than regular IoU metrics, since it not only considers the coordinates, but also the center point distance and aspect ratio (Zheng et al., 2020).

Training was conducted using Ray Tune with various hyperparameter configurations. YOLO models were optimized using random search and HyperOpt, both of which leveraged the Asynchronous HyperBand Scheduler (ASHA) (Li et al., 2018) for efficient early stopping and parallelization. In all cases, the best model of the hyperparameter optimization was selected based on mAP50-95 results. Early stopping was integrated into the training process to save the model at the optimal number of epochs. The optimal hyperparameters for the best performing three models are presented in Table 3. After optimization and training, the best models were compared using test data and evaluation metrics.

3.1.2 Evaluation metrics for model comparison The model comparison was performed to acquire a more nuanced understanding of the quality of the models. We evaluated the performance using precision, recall, mAP50 and mAP50-95. Precision and recall tell, how accurately the model labels the image to have smoke or not. mAP50 and mAP50-95 allow for varying levels of more precise performance analysis based on the accuracy of the coordinates compared to the ground truth. These metrics are readily available in Ultralytics, but for Keras, custom metric code was required. After evaluations, we selected two best-performing models to be implemented in our hardware framework.

3.2 Edge device deployment and evaluation

3.2.1 Hardware To assess the real-world feasibility of the Fireframe, we deployed two of our models on embedded hardware and tested their inference performance in a simulated operational scenario that follows our framework. For both models, we used a Raspberry Pi 5 device, connected to Basler Dart area-scan camera, equipped with an Evetar S-mount lens. Test videos were obtained from the Boreal Forest Fire data subset B (Pesonen et al., 2025) and displayed on common office monitors (Dell). The cameras, mounted in small stands (Figure 4), captured these displayed frames, simulating real-time wildfire detection conditions.

3.2.2 Software The software for the tasks and main thread, analysis as well as camera controls, was made using basic Python libraries. For camera controls, there are several Python options available. As an example, Basler has an official Python wrapper Pylon (Basler, 2025), and for Basler and other manufacturers, our in-house-backend Camazing can control several types of machine vision sensors utilizing GenICam standard (EMVA, 2025). The Camazing is available at (Jääskeläinen et al., 2019), introduced and used for example in (Rahkonen et al., 2022) and (Trops et al., 2019).

3.2.3 Workflow The captured frames were continuously passed to the deployed model (either YOLOv10 or MobileNetV3), which performed inference in real time. The output detections were rendered using OpenCV (Bradski, 2000) and displayed on separate screens. Figure 4 illustrates the full setup, showing the input screen (left) and the output display pipeline (right). This setup allowed us to measure end-to-end detection performance, including frame capture and rendering on real hardware, reflecting practical constraints and edge deployment viability.

Figure 4. Edge device setup. Left: A machine vision camera acquires a screen that shows UAV-collected video material from the Boreal Forest Fire dataset. Right: The second screen shows the results, where the detected wildfire is surrounded with a blue bounding box, and the prediction confidence is printed on the screen. The Raspberry Pi computer, that processes all tasks, is at the bottom of the image.

4. Results

4.1 Performance comparison

As can be seen in the comparison results (Table 4), all models demonstrated relatively strong performance. YOLOv10 outperformed YOLOv8 in most categories, except for recall, where YOLOv8 retained a slight advantage (0.969 versus 0.965). The MobileNetV3 achieved the highest mAP50-95 score (0.854), surpassing both YOLO variants in terms of overall precision and consistency.

Qualitative results, illustrated in Figure 5, further support these findings. The MobileNetV3-based model produced more refined and stable detections, while the YOLO models offered faster inference times, albeit with slightly lower precision and confidence.

Model	Precision	Recall	mAP-50	mAP50-95
MobileNetV3	**0.984**	**0.984**	**1.0**	**0.854**
YOLOv10	0.979	0.965	0.992	0.809
YOLOv8	0.934	0.969	0.964	0.763

Table 4. Results of the model comparison

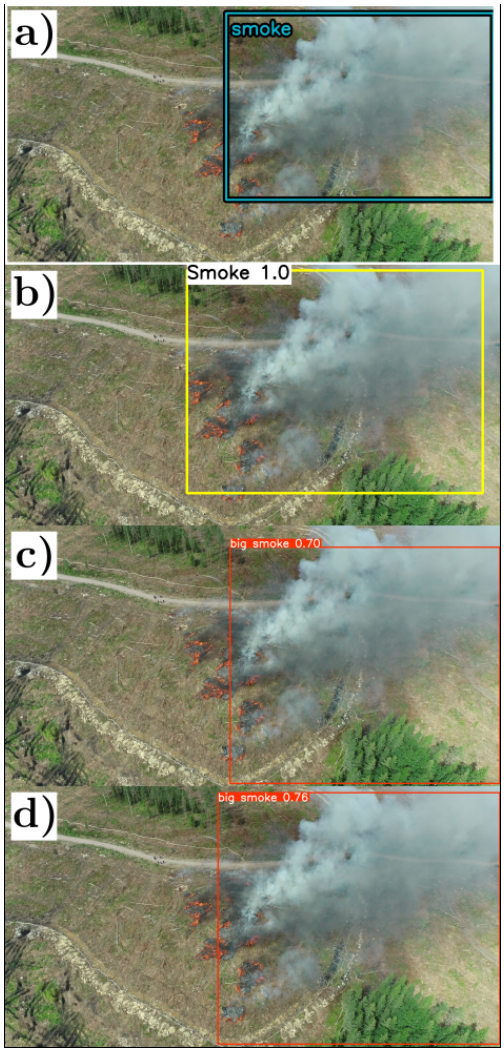

Figure 5. A sample of predictions: (a) Ground truth, (b) MobileNetV3-based model (Confidence 1.0), (c) YOLOv10 (Confidence 0.70), (d) YOLOv8 (Confidence 0.76).

In addition to detection performance, model complexity was analyzed to better understand each model's efficiency and capacity. Table 5 presents the number of layers and parameters for each model. YOLOv10 features significantly more layers (223) than YOLOv8 (129), yet has fewer total parameters (2.7M vs. 3.0M), indicating a more compact and parameter-efficient design. In contrast, the custom MobileNetV3-based model is substantially larger, with approximately 85 identifiable layer blocks and over 11.8 million parameters. This added capacity benefits precision and consistency, as reflected in its mAP50-95 score. However, this comes at the cost of increased model size.

Model	# Layers	Parameters
MobileNetV3	~85	11 823 461
YOLOv10	**223**	**2 707 430**
YOLOv8	129	3 011 043

Table 5. Model complexity comparison: number of layers and parameters. The YOLOv10 had the least trainable parameters.

4.2 Real-time deployment of Fireframe with YOLOv10 and MobileNetV3-based models

To evaluate real-time deployment, two best-performing models were tested in a Fireframe hardware setup described in Section 3.2. Both YOLOv10 and MobileNetV3-based models sustained real-time inference performance, achieving approximately one detection per second, including all phases from the frame capture to result visualization. At this speed, the Raspberry Pi computer could serve without duty-related crashes.

When the camera's frame rate and main thread's operation speed are increased, the direct effect can be seen in the reliability of the system: a small computer can easily be overloaded with tasks and data and crash.

These tests confirm the feasibility of Fireframe in deploying object detection systems in low-power, embedded environments such as UAVs and validate detection accuracy even under challenging visual conditions, including noise and motion artifacts (caused by screen technology).

5. Discussion and Conclusions

5.1 Key contributions and discussion

This study addressed two key research questions regarding low-cost, real-time wildfire smoke detection using embedded systems. The aim was to show, in practice, how scientists can test their algorithms in simple hardware experiments. This was achieved through the development and validation of a complete Fireframe system, based on Raspberry Pi and a machine vision camera. By integrating lightweight object detection models such as YOLOv10 and a custom MobileNetV3-based architecture, we achieved reliable smoke detection performance at approximately one frame per second, which can be considered suitable for real-life warning applications. The results demonstrate that optimized neural networks, even on cost-effective hardware, can enable both prototyping and real-world UAV-based deployments.

As seen, a basic machine vision camera, lens, two computer screens (one for showing unseen forest fire videos and one for visualizations), and a low-cost computational unit, such as a Raspberry Pi with the latest Ubuntu release, are the needed hardware. The needed software is minimal. For example, most of the camera controlling blocks are implemented in the manufacturer's example code, and the machine-learning libraries are as well carefully documented. No advanced programming skills are required, which may ease the implementation threshold.

The second question was how to support image acquisition, detection reasoning and visualization on resource-constrained hardware. A streamlined, single-threaded main loop enabled consistent real-time operation. This confirmed that even minimal setups can maintain practical responsiveness. Notably, the system can provide actionable outputs—visual images of detected smoke—without requiring further post-processing.

At this point, one may wonder why the main loop does not have new innovative building blocks - we agree. The main findings lay elsewhere. A Raspberry Pi computer can easily be overwhelmed when combined with a high-resolution, high-speed RGB machine vision sensor, visualization loop tasks and image analysis algorithms. The insights of our study reveal that

a good camera control backend system can be used to handle camera features and configure the data flow wisely.

Through several iterations of testing the best possible camera settings, we ended up setting the camera's frame rate to one frame per second, limiting its autocorrection functions so that we only used the necessary functions (no fancy pixel correction like automatic color balance was needed), and limiting the width and height (region of interest) of the captured image so that it only focused on the video, instead of the video, monitor and its surroundings. These actions were important for us to achieve a reliable system, without data overwhelm. The Raspberry Pi computer was able to serve without crashing. Therefore, we recommend, not only to switch on the cameras, but also pay attention to the GenICam standard features ((EMVA, 2025)) of the camera, and its acquisition control possibilities via Python interface.

5.2 Limitations and future work

While the Fireframe shows promise for embedded wildfire monitoring, some limitations remain. Real-time broadcasting to end-users or authorities was outside the study's scope, as implementation depends on specific operational contexts and infrastructure.

Our main focus was not to introduce a new novel object detection model, but to create a flexible framework for evaluating existing models in wildfire scenarios. Evaluation centered on mAP and energy efficiency; practitioners should also consider dataset variability and avoid including visualization in performance measurements unless necessary.

The current test dataset, although containing 1822 images, includes a high number of samples with clearly visible smoke and relatively few negative examples. This imbalance can affect metrics such as precision and recall. To mitigate this, we emphasized mean Average Precision (mAP) and enforced consistent labeling for bounding boxes.

Regarding detection speed tests. These camera stream limitations, producing eventually result visualizations once per second, may sound slow, but that is deliberate. In an action, the firefighters don't need tens of warnings per second, but they benefit from a visualization, which, of course, takes energy and computational capacity in this system. Therefore, the version we tested, was solid, and performed at a rate that enables visual inspection and interpretations of the result visualizations. However, if one wishes to push boundaries, we recommend removing the visualization loop from the framework, adding energy consumption measurement devices into the framework and increasing the frame rate of the camera to high-speed, and then evaluating the detection rate.

References

Abid, F., 2021. A survey of machine learning algorithms based forest fires prediction and detection systems. *Fire Technology*, 57(2), 559–590.

Ariante, G., Del Core, G., 2025. Unmanned Aircraft Systems (UASs): Current State, Emerging Technologies, and Future Trends. *Drones*, 9(1), 59.

Bailon-Ruiz, R., Lacroix, S., 2020. Wildfire remote sensing with uavs: A review from the autonomy point of view. *2020 International Conference on Unmanned Aircraft Systems (ICUAS)*, IEEE, 412–420.

Barmpoutis, P., Papaioannou, P., Dimitropoulos, K., Grammalidis, N., 2020. A review on early forest fire detection systems using optical remote sensing. *Sensors*, 20(22), 6442.

Basler, 2025. Pypylon, the official python wrapper for the Basler pylon Camera Software Suite.

Best, N., Ott, J., Linstead, E. J., 2020. Exploring the efficacy of transfer learning in mining image-based software artifacts. *Journal of Big Data*, 7(1), 1–10.

Bouguettaya, A., Zarzour, H., Taberkit, A. M., Kechida, A., 2022. A review on early wildfire detection from unmanned aerial vehicles using deep learning-based computer vision algorithms. *Signal Processing*, 190, 108309.

Bradski, G., 2000. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*.

Carta, F., Zidda, C., Putzu, M., Loru, D., Anedda, M., Giusto, D., 2023. Advancements in Forest Fire Prevention: A Comprehensive Survey. *Sensors*, 23(14). https://www.mdpi.com/1424-8220/23/14/6635.

Chen, G., Cheng, R., Lin, X., Jiao, W., Bai, D., Lin, H., 2023a. LMDFS: A Lightweight Model for Detecting Forest Fire Smoke in UAV Images Based on YOLOv7. *Remote Sensing*, 15(15). https://www.mdpi.com/2072-4292/15/15/3790.

Chen, L., Gao, H., Li, Y., Zhang, X., 2023b. Application of Deep Transfer Learning on UAV-Based Aerial Images for Forest Fire Detection. *Proceedings of SPIE*, 13442, 134421W. https://www.spiedigitallibrary.org/conference-proceedings-of-spie/13442/134421W.

Chollet, F. et al., 2015. Keras. `https://keras.io`. Accessed: 2025-06-25.

Deng, J., 2012. Large Scale Visual Recognition. PhD thesis, Princeton University.

EMVA, 2025. GenICam standard. The Generic Interface for Cameras standard is the base for plug play handling of cameras and devices.

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(5), 1943–1980.

Ha, V. K., Ren, J., Xu, X., Zhao, S., Xie, G., Vargas, V. M., 2018. Deep learning based single image super-resolution: a survey. *Deep learning based single image super-resolution: a survey*, Springer, 106–119.

Hosseini, S., Choi, Y., 2022. Deep Learning and Transformer Approaches for UAV-Based Wildfire Detection and Segmentation. *Sensors*, 22(5), 1977. https://www.mdpi.com/1424-8220/22/5/1977.

Jääskeläinen, S., Annala, L., Eskelinen, M. A., Raita-Hakola, A.-M., 2019. Spectral Imaging Laboratory's Machine vision library for GenICam-compliant cameras. Developed at the University of Jäskylä, Spectral Imaging Laboratory. Released under MIT-licence.

Johnston, F. H., Henderson, S. B., Chen, Y., Randerson, J. T., Marlier, M., DeFries, R. S., Kinney, P., Bowman, D. M., Brauer, M., 2012. Estimated global mortality attributable to smoke from landscape fires. *Environmental Health Perspectives*, 120(5), 695–701.

Krizhevsky, A., Sutskever, I., Hinton, G. E., 2017. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6), 84–90.

Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., Talwalkar, A., 2018. Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization. *Journal of Machine Learning Research*, 18(185), 1–52. http://jmlr.org/papers/v18/16-558.html.

Moumgiakmas, S. S., Samatas, G. G., Papakostas, G. A., 2021. Computer vision for fire detection on UAVs—From software to hardware. *Future Internet*, 13(8), 200.

Park, S. J., Ko, Y. T., 2020. A study on forest fire detection system based on image processing using drone. *International Journal of Computer Science and Network Security (IJCSNS)*, 20(4), 167–172.

Pesonen, J., Raita-Hakola, A.-M., Joutsalainen, J., Hakala, T., Akhtar, W., Karjalainen, V., Koivumäki, N., Markelin, L., Suomalainen, J., de Oliveira, R. A. et al., 2025. Boreal forest fire: Uav-collected wildfire detection and smoke segmentation dataset. `https://doi.org/10.23729/fd-72c6cf74-b8eb-3687-860d-bf93a1ab94c9`. National Land Survey of Finland, FGI Dept. of Remote sensing and photogrammetry.

Qu, Z., 2022. Enabling Deep Learning on Edge Devices. PhD thesis, ETH Zurich. Ph.D. dissertation.

Rahkonen, S., Lind, L., Raita-Hakola, A.-M., Kiiskinen, S., Pölönen, I., 2022. Reflectance Measurement Method Based on Sensor Fusion of Frame-Based Hyperspectral Imager and Time-of-Flight Depth Camera. *Sensors*, 22(22). https://www.mdpi.com/1424-8220/22/22/8668.

Raita-Hakola, A.-M., Rahkonen, S., Suomalainen, J., Markelin, L., Oliveira, R., Hakala, T., Koivumäki, N., Honkavaara, E., Pölönen, I., 2023. Combining YOLO V5 and transfer learning for smoke-based wildfire detection on boreal forests. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XLVIII-1/W2-2023, 1771–1778. https://isprs-archives.copernicus.org/articles/XLVIII-1-W2-2023/1771/2023/.

Saleh, A., Zulkifley, M. A., Harun, H. H., Gaudreault, F., Davison, I., Spraggon, M., 2024. Forest fire surveillance systems: A review of deep learning methods. *Heliyon*, 10(1), e23127. https://www.sciencedirect.com/science/article/pii/S2405844023103355.

Sathishkumar, V. E., Cho, J., Subramanian, M., Naren, O. S., 2023. Forest fire and smoke detection using deep learning-based learning without forgetting. *Fire Ecology*, 19(1), 9. https://doi.org/10.1186/s42408-022-00165-0.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R., 2014. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1), 1929–1958.

Surya, L., 2020. Fighting fire with ai: Using deep learning to help predict wildfires in the us. *International Journal of Creative Research Thoughts (IJCRT), ISSN*, 2320–2882.

Trops, R., Hakola, A.-M., Jääskeläinen, S., Näsilä, A., Annala, L., Eskelinen, M. A., Saari, H., Pölönen, I., Rissanen, A., 2019. Miniature MOEMS hyperspectral imager with versatile analysis tools. W. Piyawattanametha, Y.-H. Park, H. Zappe (eds), *MOEMS and Miniaturized Systems XVIII*, 10931, International Society for Optics and Photonics, SPIE, 109310W.

Ultralytics, 2023a. Configuration - ultralytics yolo documentation.

Ultralytics, 2023b. Ultralytics github repository. `https://github.com/ultralytics`. Accessed: 2025-06-25.

UNEP, GRID-Arendal, 2022. Spreading like wildfire: The rising threat of extraordinary landscape fires. United Nations Environment Programme.

Wu, Z., Zhang, Y., Li, Y., Wang, S., Zhang, Y., 2023. Forest Fire Smoke Detection Based on Deep Learning Approaches and UAV Imagery. *Sensors*, 23(12), 5702. https://www.mdpi.com/1424-8220/23/12/5702.

Yang, H., Jang, H., Kim, T., Lee, B., 2019. Non-temporal lightweight fire detection network for intelligent surveillance systems. *IEEE Access*, 7, 169257–169266.

Zheng, Z., Wang, P., Liu, W., Li, J., Ye, R., Ren, D., 2020. Distance-IoU Loss: Faster and Better Learning for Bounding Box Regression. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(07), 12993-13000. https://ojs.aaai.org/index.php/AAAI/article/view/6999.

Zhou, Y., Cheng, H., Jiang, S., Jiang, S., 2019. A deep learning based forest fire detection approach using uav and yolov3. *Proceedings of the 1st International Conference on Industrial Artificial Intelligence (IAI)*, 1–5.