Drone-Based Fire Hazard Detection and Alert System

🎯 Project Aim

To develop a drone-mounted, context-aware fire hazard detection system capable of identifying potential fire outbreaks in **buildings or forests** and alerting responders in **real time**, even in **mobile network dead zones**.

🚨 1. Problem Statement

- **Delayed fire detection** results in massive property and ecological damage.
- **Remote areas** or disaster-hit zones often **lack cellular coverage**, delaying alerts.
- Traditional smoke detectors are localized and unsuitable for open environments like forests.

💡 Innovative Solution

Use a **mobile, autonomous drone** equipped with a:

- **Raspberry Pi (or ESP32-CAM)** for onboard edge processing.
- **Thermal camera** (like MLX90640) or **RGB camera** to detect fire signatures.
- **Gas sensor** (MQ-2 or more efficient one) to analyze ambient air quality.
- **GPS sensor** (via smartphone or dedicated module) to geo-tag hazard locations.

And establish a **mesh network** using LoRa or ESP-NOW for:

- Long-range communication
- Operation in **no-internet environments**

🧠 . Context Awareness

This system intelligently integrates multiple data types:

- **Thermal data or RGB inference**: Detecting heat signatures, flames, or smoke.
- **Gas levels**: Air quality confirmation via MQ-2 sensor.
- **Location (GPS)**: Identifies exact coordinates of potential hazard.

The goal is to **reduce false positives** and trigger alerts **only in verified cases**.

. Workflow Overview

A[Drone Flight Path] --> B[Frame Capture (Thermal or RGB)] --> C[Edge ML Inference (eim model)] C -->|Fire| D[Read MQ-2 Gas Sensor] D -->|Gas Level > Threshold| E[Fetch GPS Coordinates] E --> F[Send Alert via SMS/App]

🔗 . Communication Architecture

In Internet-Available Areas:

- Use **Wi-Fi / 4G module** on Pi
- Alert via **Pushbullet, SMS, or app**

In Internet-Denied Areas:

- Use **Mesh Network** with:
- **ESP-NOW** (for ESP32 nodes)
- **LoRa** for ultra-long range low-power communication
- Base station on ground relays alerts once network is found

🧰 . Hardware Components

Component	Purpose
Raspberry Pi 3 / ESP32-CAM	Main controller, image inference
Thermal Camera / IP Webcam in smartphone	Capture input image
MQ-2 Gas Sensor	Detect flammable gas or smoke
GPS module / Smartphone GPS	Geo-tagging hazard location
Pushbullet / GSM / LoRa	Alert delivery
Drone Frame + Battery	Mobility and power
ESP-NOW or LoRa	Mesh network in offline regions

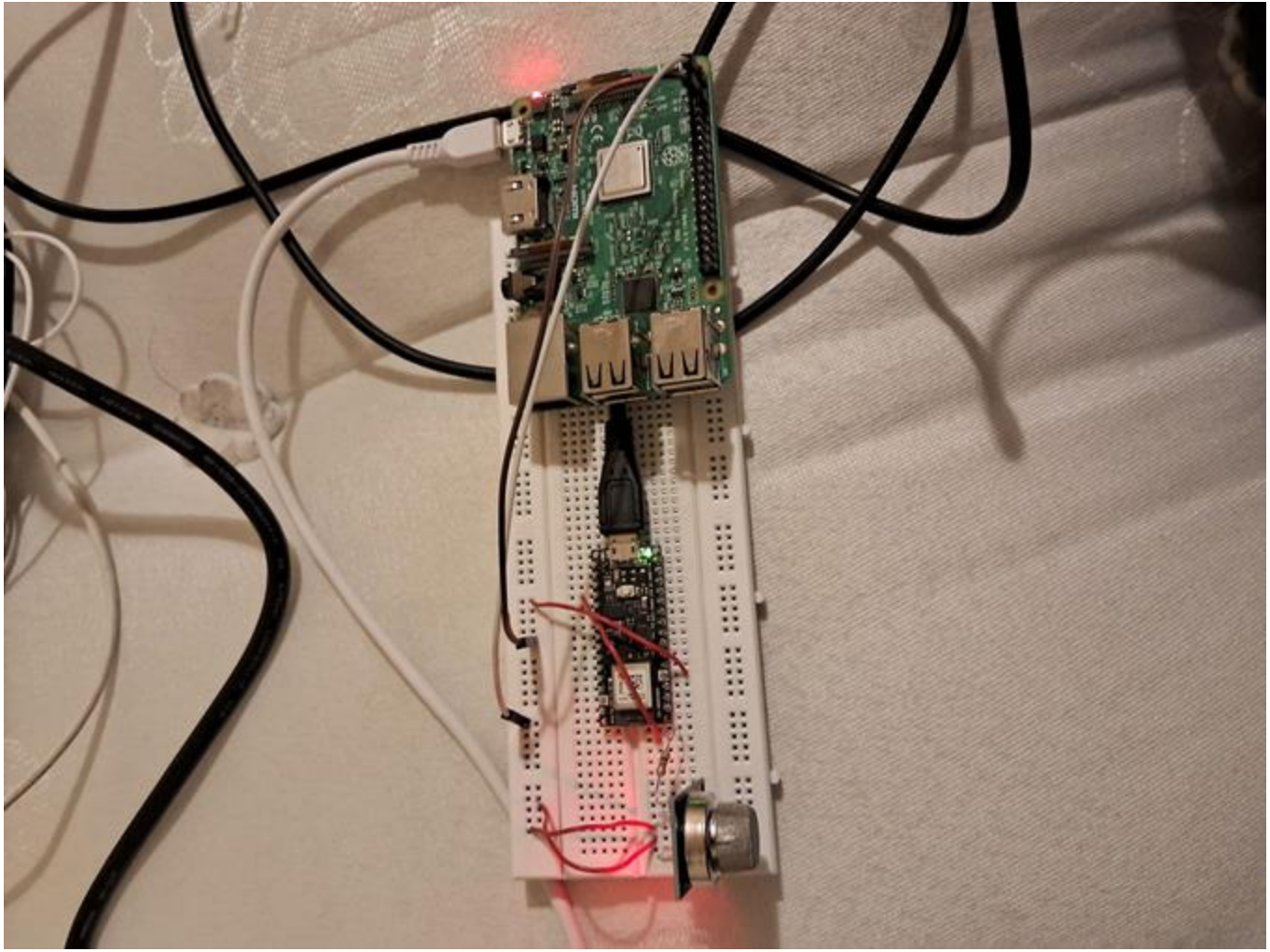
🧪 . Software Flow

Onboard Inference Code (Pi)

- Load .eim model (trained in Edge Impulse)
- Capture frame from IP camera or thermal camera
- Run frame through ML model
- If label is Wild_Fire or Normal_Fire:
- Read gas value via BLE
- If gas value exceeds threshold:
- Fetch GPS via Termux API
- Send location-tagged alert via Pushbullet (or SMS in real deployment)

✅ What was Achieved

- Trained and deployed a working **RGB-based fire classification model** in raspberry pi 3
- Replaced thermal module with smartphone camera stream (IP Webcam) since MLX90640 was damaged due to reverse wiring
- Model inference with IP Webcam live stream
- BLE-based MQ-2 sensor data collection
- Smartphone GPS via **Termux API**
- Alert dispatch via **Pushbullet**
- Built a working demo on **Raspberry Pi 3 Model B**



https://studio.edgeimpulse.com/studio/693568/acquisition/training?page=1

EDGE IMPULSE

apremgolickal / Fire detection model PERSONAL

Target: Raspberry Pi 4

- Dashboard
- Devices
- Data acquisition
- Experiments
- EON Tuner
- Impulse design
- Create impulse
- Image
- Transfer learning
- Retrain model
- Live classification
- Model testing
- Deployment

Upgrade Plan

Get access to higher job limits and more collaborators.

View plans

Dataset | Data explorer | Data sources | Synthetic data | AI labeling | CSV Wizard

DATA COLLECTED
1,802 items

TRAIN / TEST SPLIT
80% / 20%

Dataset

Training (1,441) Test (361)

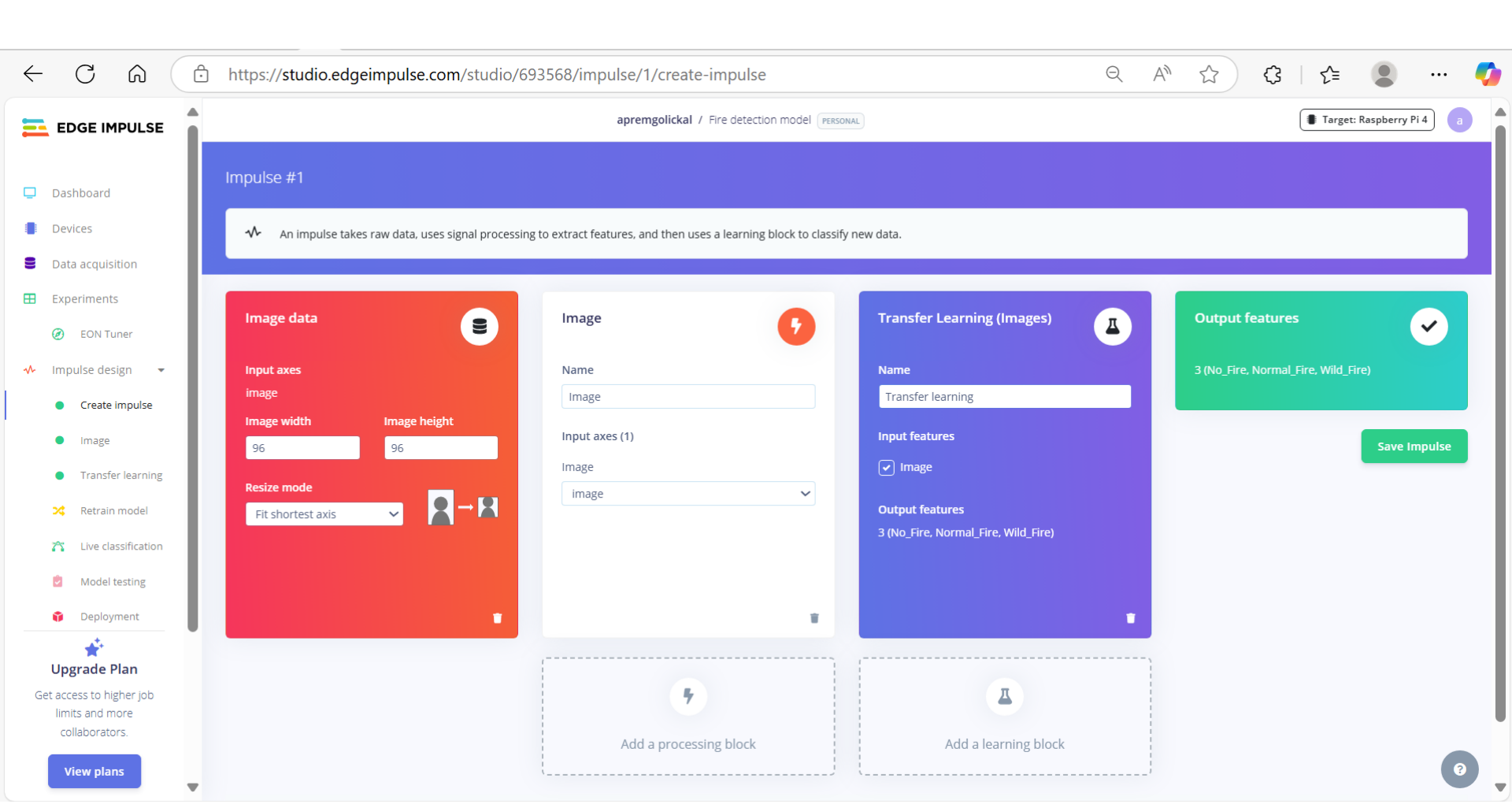
SAMPLE NAME	LABEL	ADDED
00616	Wild_Fire	Yesterday, 01:28:43
00613	Wild_Fire	Yesterday, 01:28:43
00618	Wild_Fire	Yesterday, 01:28:43
00615	Wild_Fire	Yesterday, 01:28:43
00619	Wild_Fire	Yesterday, 01:28:43
00622	Wild_Fire	Yesterday, 01:28:43
00612	Wild_Fire	Yesterday, 01:28:43
00614	Wild_Fire	Yesterday, 01:28:43
00617	Wild_Fire	Yesterday, 01:28:43
00608	Wild_Fire	Yesterday, 01:28:43

Collect data

Connect a device to start building your dataset.

RAW DATA

Click on a sample to load...


https://studio.edgeimpulse.com/studio/693568/impulse/1/create-impulse
EDGE IMPULSE
apremgolickal / Fire detection model PERSONAL
Target: Raspberry Pi 4
Dashboard
Devices
Data acquisition
Experiments
EON Tuner
Impulse design
Create impulse
Image
Transfer learning
Retrain model
Live classification
Model testing
Deployment
Upgrade Plan
Get access to higher job limits and more collaborators.
View plans
Impulse #1
An impulse takes raw data, uses signal processing to extract features, and then uses a learning block to classify new data.
Image data
Input axes
image
Image width
96
Image height
96
Resize mode
Fit shortest axis
Image
Name
Image
Input axes (1)
Image
image
Transfer Learning (Images)
Name
Transfer learning
Input features
Image
Output features
3 (No_Fire, Normal_Fire, Wild_Fire)
Output features
3 (No_Fire, Normal_Fire, Wild_Fire)
Save Impulse
Add a processing block
Add a learning block

https://studio.edgeimpulse.com/studio/693568/impulse/1/dsp/image/3

EDGE IMPULSE

apremgolickal / Fire detection model PERSONAL

Target: Raspberry Pi 4

- Dashboard
- Devices
- Data acquisition
- Experiments
- EON Tuner
- Impulse design
- Create impulse
- Image
- Transfer learning
- Retrain model
- Live classification
- Model testing
- Deployment

Upgrade Plan

Get access to higher job limits and more collaborators.

View plans

Parameters | Generate features

Raw data

Show: All labels | 00616 (Wild_Fire)

Raw features

0xa0d9f6, 0x9fd8f5, 0x9fd8f5, 0x9fd8f5, 0x9ed7f4, 0x9cd5f2, 0x9cd5f3, 0x9cd5f3, 0x9bd4f2, 0x9bd4f2, 0x9bd3f4…

Parameters

Image

Color depth: RGB

Save parameters

DSP result

Image

Processed features

0.6275, 0.8510, 0.9647, 0.6235, 0.8471, 0.9608, 0.6235, 0.8471, 0.9608, 0.6235, 0.8471, 0.9608, 0.6196, 0.84…

On-device performance

PROCESSING TIME

PEAK RAM USAGE

EDGE IMPULSE

- Dashboard
- Devices
- Data acquisition
- Experiments
- EON Tuner
- Impulse design
- Create impulse
- Image
- Transfer learning
- Retrain model
- Live classification
- Model testing
- Deployment
- Versioning

GETTING STARTED

- Documentation
- Forums

Transfer learning settings

Training settings

Training processor	GPU
Number of training cycles	30
Learning rate	0.001
Model size	B0 - 4M params, 16 MB
Use pretrained weights	☑
Freeze % of layers	85
Last layers	dense: 32, dropout: 0.1
Data augmentation	flip, crop, brightness

Valid options: flip, crop, brightness

Advanced training settings

Validation set size	20 %
Split train/validation set on metadata key	
Profile int8 model	☑
Batch size	16
Early stopping	☑
Early stopping patience	10
Early stopping min. delta	0.001

Neural network architecture

Input layer (27,648 features)

Edge Impulse Inc. / EfficientNet

Training output

(0)

Model

Model version: Quantized (int8)

Last training performance (validation set)

ACCURACY	LOSS
97.6%	0.17

Confusion matrix (validation set)

	NO_FIRE	NORMAL_FIRE	WILD_FIRE
NO_FIRE	95.9%	2.7%	1.4%
NORMAL_FIRE	0.9%	99.1%	0%
WILD_FIRE	2.9%	0%	97.1%
F1 SCORE	0.95	0.99	0.98

Metrics (validation set)

METRIC	VALUE
Area under ROC Curve	1.00
Weighted average Precision	0.98
Weighted average Recall	0.98
Weighted average F1 score	0.98

Data explorer (full training set)

On-device performance

Engine: TensorFlow Lite

INFERENCING TIME	PEAK RAM USAGE	FLASH USAGE
207 ms	1.1M	4.8M

EDGE IMPULSE

Dashboard
Devices
Data acquisition
Experiments
EON Tuner
Impulse design
Create impulse
Image
Transfer learning
Retrain model
Live classification
Model testing
Deployment

This lists all test data. You can manage this data through Data acquisition.

Test data

Classify all

Set the 'expected outcome' for each sample to the desired outcome to automatically score the impulse.

SAMPLE NAME	EXPECTED OUTCOME	ACCURACY	RESULT
00620	Wild_Fire	100%	1 Wild_Fire
00621	Wild_Fire	100%	1 Wild_Fire
00610	Wild_Fire	100%	1 Wild_Fire
00604	Wild_Fire	100%	1 Wild_Fire
00600	Wild_Fire	100%	1 Wild_Fire
00601	Wild_Fire	100%	1 Wild_Fire
00589	Wild_Fire	100%	1 Wild_Fire
00585	Wild_Fire	100%	1 Wild_Fire
00582	Wild_Fire	100%	1 Wild_Fire
00574	Wild_Fire	100%	1 Wild_Fire

Model testing output

(0)

Results

Model version: Quantized (int8)

ACCURACY
95.84%

Metrics for Transfer learning

METRIC	VALUE
Area under ROC Curve	1.00
Weighted average Precision	0.97
Weighted average Recall	0.97
Weighted average F1 score	0.97

Confusion matrix

	NO_FIRE	NORMAL_FIRE	WILD_FIRE	UNCERTAIN
NO_FIRE	93.3%	0%	2.5%	4.2%
NORMAL_FIRE	0%	100%	0%	0%
WILD_FIRE	0%	0%	94.4%	5.6%
F1 SCORE	0.97	1.00	0.96	

Feature explorer

https://studio.edgeimpulse.com/studio/693568/impulse/1/deployment

EDGE IMPULSE

apremgolickal / Fire detection model PERSONAL

Target: Raspberry Pi 4

- Dashboard
- Devices
- Data acquisition
- Experiments
- EON Tuner
- Impulse design
  - Create impulse
  - Image
  - Transfer learning
  - Retrain model
  - Live classification
  - Model testing
  - Deployment

Upgrade Plan

Get access to higher job limits and more collaborators.

View plans

Configure your deployment

You can deploy your impulse to any device. This makes the model run without an internet connection, minimizes latency, and runs with minimal power consumption. Read more.

Linux (ARMv7) ×

SELECTED DEPLOYMENT

Linux (ARMv7)

An EIM binary for Linux (ARMv7) that implements the Edge Impulse Linux protocol.

DEPLOY TO ANY LINUX-BASED DEVELOPMENT BOARD

Edge Impulse for Linux lets you run your models on any Linux-based development board, with SDKs for Node.js, Python, Go and C++ to integrate your models quickly into your application.

1. Install the Edge Impulse Linux CLI
2. Run `edge-impulse-linux-runner` (run with `--clean` to switch projects)

See the documentation for more information and setup instructions. Alternatively, you can download your model for Linux (ARMv7) below.

MODEL OPTIMIZATIONS

Model optimizations can increase on-device performance but may reduce accuracy.

Quantized (int8)

Selected ✔

	IMAGE	TRANSFER LEARNING	TOTAL
LATENCY	1 ms.	207 ms.	**208 ms.**
RAM	4.0K	1.3M	**1.3M**
FLASH	-	4.5M	-
ACCURACY			**95.84%**

Latest build

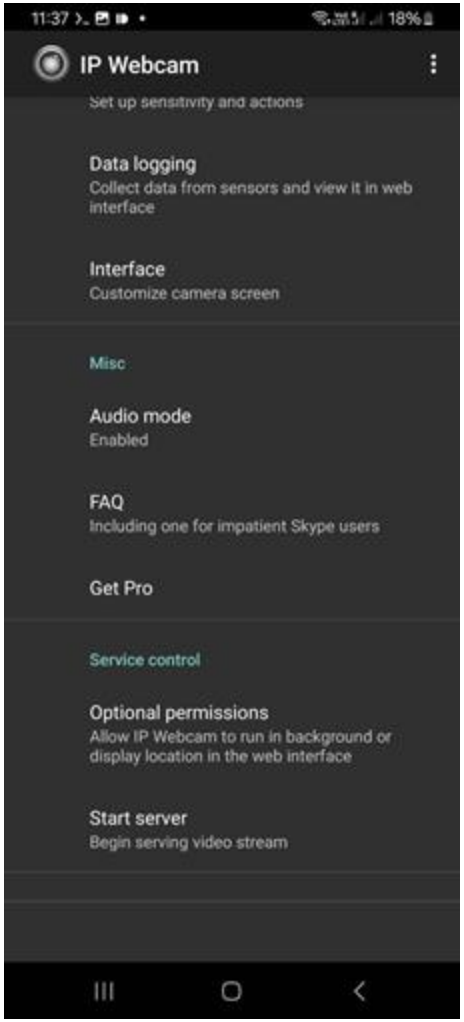
v8 (Linux (ARMv7))
Yesterday, 12:48:38

View docs

Run this model

Scan QR code or launch in browser to test your prototype

Launch in browser


11:37
18%
IP Webcam
Set up sensitivity and actions
Data logging
Collect data from sensors and view it in web interface
Interface
Customize camera screen
Misc
Audio mode
Enabled
FAQ
Including one for impatient Skype users
Get Pro
Service control
Optional permissions
Allow IP Webcam to run in background or display location in the web interface
Start server
Begin serving video stream

Here's a **step-by-step guide** to fully set up and run the **GPS server on your Android phone using Termux**, so your **Raspberry Pi can request your phone's location automatically when fire is detected**:

✅ Step-by-Step Setup for Termux GPS Server on Android

📱 1. **Install Termux & Termux:API (Both Required)**
🔹 a. Install Termux:
Go to F-Droid from your phone.
Download and install **Termux**.
🔹 b. Install Termux:API Add-on:
Go to Termux:API on F-Droid.
Install it.
Give it location permission when asked.

🛠️ 2. **Initial Setup in Termux**
Open **Termux** and run the following commands **one by one**:

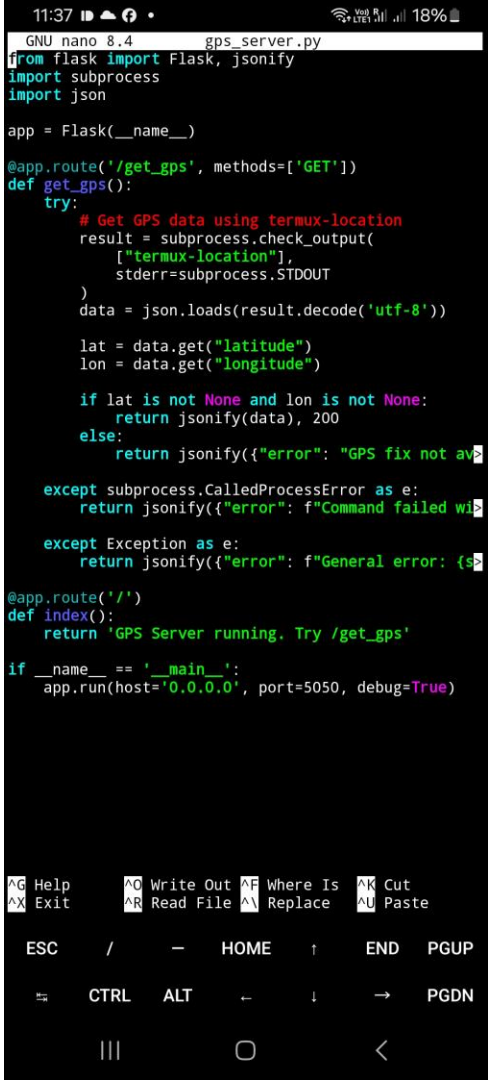
```
pkg update && pkg upgrade -y
pkg install termux-api python -y
termux-setup-storage
```

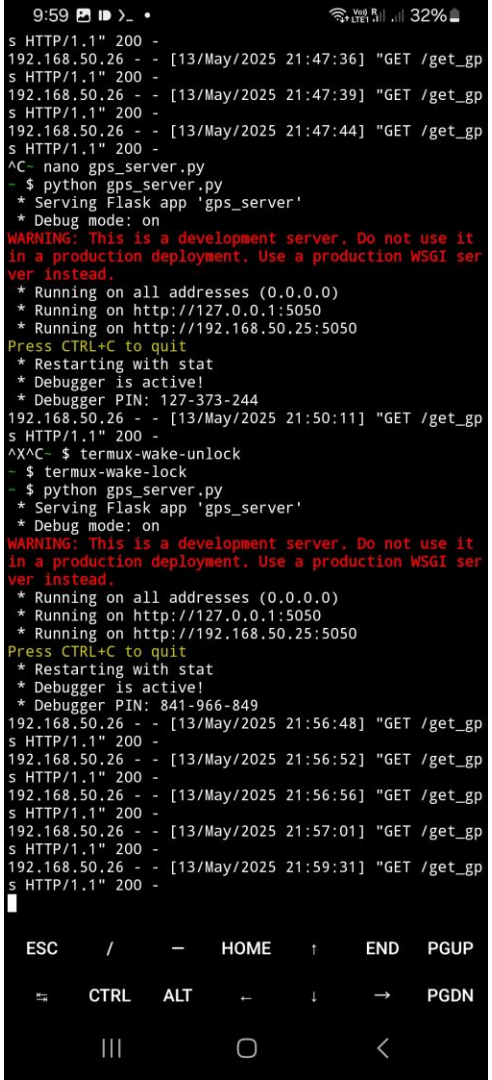
✔️ This sets up Python and the Termux APIs (including `termux-location`).

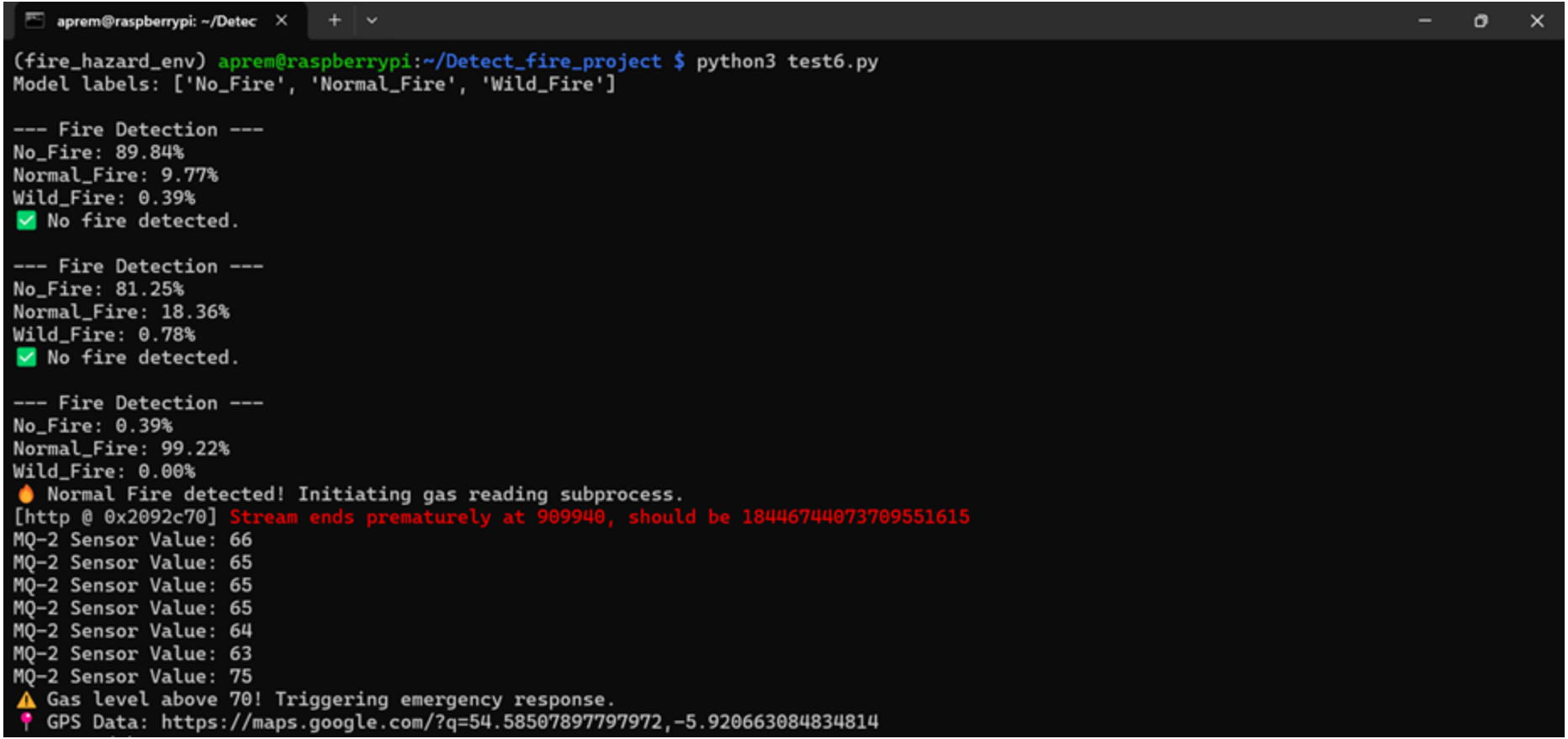
🧠 3. **Create the GPS Server Script**
🔸 a. Create a file called `gps_server.py`:

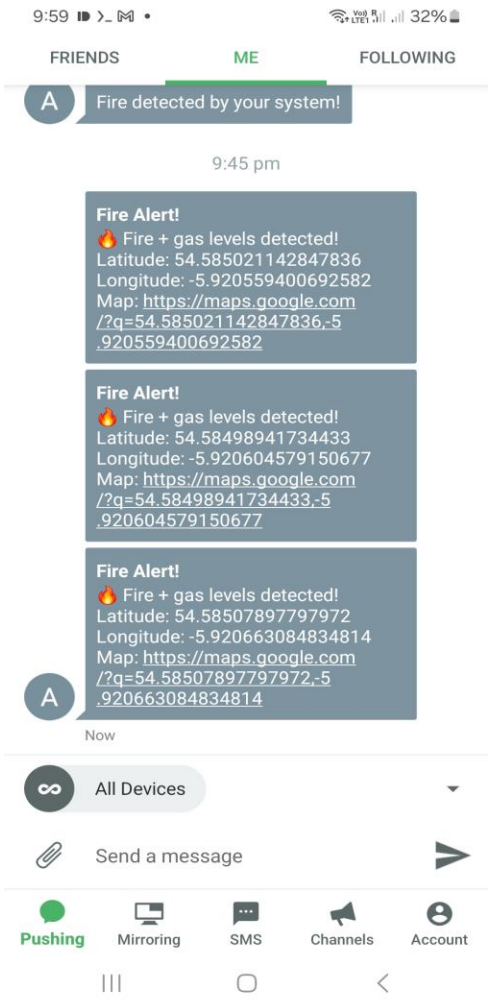
```
nano gps_server.py
```

- 🧱 4. **Install Flask (for the web server)**
- `pip install flask`
-
- **Find Your Phone's IP (on Wi-Fi)**
- In **Termux**, run:
- `ip a`


11:37
18%
GNU nano 8.4 gps_server.py
from flask import Flask, jsonify
import subprocess
import json
app = Flask(__name__)
@app.route('/get_gps', methods=['GET'])
def get_gps():
try:
# Get GPS data using termux-location
result = subprocess.check_output(
["termux-location"],
stderr=subprocess.STDOUT
)
data = json.loads(result.decode('utf-8'))
lat = data.get("latitude")
lon = data.get("longitude")
if lat is not None and lon is not None:
return jsonify(data), 200
else:
return jsonify({"error": "GPS fix not av>
except subprocess.CalledProcessError as e:
return jsonify({"error": f"Command failed wi>
except Exception as e:
return jsonify({"error": f"General error: {s>
@app.route('/')
def index():
return 'GPS Server running. Try /get_gps'
if __name__ == '__main__':
app.run(host='0.0.0.0', port=5050, debug=True)
^G Help ^O Write Out ^F Where Is ^K Cut
^X Exit ^R Read File ^\ Replace ^U Paste
ESC / — HOME ↑ END PGUP
CTRL ALT ← ↓ → PGDN


9:59
32%
s HTTP/1.1" 200 -
192.168.50.26 - - [13/May/2025 21:47:36] "GET /get_gp
s HTTP/1.1" 200 -
192.168.50.26 - - [13/May/2025 21:47:39] "GET /get_gp
s HTTP/1.1" 200 -
192.168.50.26 - - [13/May/2025 21:47:44] "GET /get_gp
s HTTP/1.1" 200 -
^C~ nano gps_server.py
~ $ python gps_server.py
* Serving Flask app 'gps_server'
* Debug mode: on
WARNING: This is a development server. Do not use it
in a production deployment. Use a production WSGI ser
ver instead.
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:5050
* Running on http://192.168.50.25:5050
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 127-373-244
192.168.50.26 - - [13/May/2025 21:50:11] "GET /get_gp
s HTTP/1.1" 200 -
^X^C~ $ termux-wake-unlock
~ $ termux-wake-lock
~ $ python gps_server.py
* Serving Flask app 'gps_server'
* Debug mode: on
WARNING: This is a development server. Do not use it
in a production deployment. Use a production WSGI ser
ver instead.
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:5050
* Running on http://192.168.50.25:5050
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 841-966-849
192.168.50.26 - - [13/May/2025 21:56:48] "GET /get_gp
s HTTP/1.1" 200 -
192.168.50.26 - - [13/May/2025 21:56:52] "GET /get_gp
s HTTP/1.1" 200 -
192.168.50.26 - - [13/May/2025 21:56:56] "GET /get_gp
s HTTP/1.1" 200 -
192.168.50.26 - - [13/May/2025 21:57:01] "GET /get_gp
s HTTP/1.1" 200 -
192.168.50.26 - - [13/May/2025 21:59:31] "GET /get_gp
s HTTP/1.1" 200 -
ESC / — HOME ↑ END PGUP
CTRL ALT ← ↓ → PGDN


aprem@raspberrypi: ~/Detec
(fire_hazard_env) aprem@raspberrypi:~/Detect_fire_project $ python3 test6.py
Model labels: ['No_Fire', 'Normal_Fire', 'Wild_Fire']
--- Fire Detection ---
No_Fire: 89.84%
Normal_Fire: 9.77%
Wild_Fire: 0.39%
✅ No fire detected.
--- Fire Detection ---
No_Fire: 81.25%
Normal_Fire: 18.36%
Wild_Fire: 0.78%
✅ No fire detected.
--- Fire Detection ---
No_Fire: 0.39%
Normal_Fire: 99.22%
Wild_Fire: 0.00%
🔥 Normal Fire detected! Initiating gas reading subprocess.
[http @ 0x2092c70] Stream ends prematurely at 909940, should be 18446744073709551615
MQ-2 Sensor Value: 66
MQ-2 Sensor Value: 65
MQ-2 Sensor Value: 65
MQ-2 Sensor Value: 65
MQ-2 Sensor Value: 64
MQ-2 Sensor Value: 63
MQ-2 Sensor Value: 75
⚠️ Gas level above 70! Triggering emergency response.
📍 GPS Data: https://maps.google.com/?q=54.58507897797972,-5.920663084834814


9:59
32%
FRIENDS
ME
FOLLOWING
A
Fire detected by your system!
9:45 pm
Fire Alert!
🔥 Fire + gas levels detected!
Latitude: 54.585021142847836
Longitude: -5.920559400692582
Map: https://maps.google.com/?q=54.585021142847836,-5.920559400692582
Fire Alert!
🔥 Fire + gas levels detected!
Latitude: 54.58498941734433
Longitude: -5.920604579150677
Map: https://maps.google.com/?q=54.58498941734433,-5.920604579150677
Fire Alert!
🔥 Fire + gas levels detected!
Latitude: 54.58507897797972
Longitude: -5.920663084834814
Map: https://maps.google.com/?q=54.58507897797972,-5.920663084834814
A
Now
All Devices
Send a message
Pushing
Mirroring
SMS
Channels
Account

Future Expansion

- Re-introduce **thermal camera** for better night/fire heat detection
- Migrate camera system to onboard **ESP32-CAM** (lightweight)
- Implement **LoRa-based mesh network** for resilient communication
- Add **flight autonomy** using GPS waypoints and object avoidance

Critical Evaluation of Performance

The performance evaluation of the fire hazard detection system was carried out under simulated room conditions using real-time inputs from a smartphone camera, BLE-connected MQ-2 gas sensor, and GPS data via the Termux API. The image classification model was built using **Edge Impulse's EfficientNet architecture**, optimized for low-resource edge devices like the **Raspberry Pi 3 Model B**. While the model achieved a **high classification accuracy on the test dataset**, its real-time performance in practical scenarios revealed a mismatch between **lab-trained accuracy and field reliability**.

Specifically, in live inference sessions, the system frequently returned **inaccurate predictions**, often misclassifying No_Fire as Normal_Fire or Wild_Fire or vice versa. This limitation was likely due to **insufficient training diversity**—with the model struggling to generalize across varying lighting, angles, and cluttered environments. Performance dropped noticeably under **low-light or visually noisy backgrounds**, highlighting that even lightweight, high-performing models like EfficientNet can falter when image contrast or clarity degrades in real-world conditions.

Continued.

From a systems integration standpoint, the **BLE communication** between the **Arduino Nano 33 BLE Sense Lite** and Raspberry Pi worked well in most cases. The Pi was able to detect the Nano and retrieve gas readings within 5 seconds of fire detection. However, **BLE delays and inconsistencies** were occasionally encountered, not due to signal issues or hardware repositioning, but due to **abrupt termination of the BLE handler on the Nano**, which caused connection timeouts or stale data reception. This highlights the need for more robust **BLE error handling and graceful disconnection logic**.

A significant strength of the system lies in its **context-aware multi-sensor logic**, which reduced false positives by ensuring that alerts were only triggered when both visual fire evidence and elevated gas levels were detected. This dual-condition logic improved reliability and credibility of alerts. The integration of **GPS-based location via Termux API** and **Pushbullet alert delivery** worked seamlessly, enabling real-time hazard alerts with Google Maps links to be dispatched within approximately 12 seconds of detection—demonstrating effective end-to-end system responsiveness.

Continued.

In addition to these core strengths, the **modular architecture** of the system offers substantial upgrade potential. For instance, reintroducing the originally planned **MLX90640 thermal camera** would allow for heat-based fire detection, critical for low-light or obscured settings, enhancing accuracy where RGB fails. Further, incorporating **LoRa modules** could enable long-range, low-power communication in forested or rural zones lacking internet, while a **GSM module** could support SMS-based alerting in complete absence of Wi-Fi. Such upgrades would extend the system’s deployment capability to **autonomous drone platforms** and **remote environments**, aligning with the project's original vision.

Continued.

Despite its current limitations, the system successfully demonstrates the **viability of low-cost, edge-based pervasive computing** for disaster response. It integrates visual, environmental, and location data in real time, and shows clear promise for real-world deployment with continued refinement in data collection, BLE communication robustness, and sensor fusion strategies.

It effectively met the core objectives of pervasive computing by combining edge intelligence, mobility, and environmental awareness into a deployable hazard response tool.

Concluding Remarks

This project successfully demonstrated a modular, edge-based fire hazard detection and alert system that leverages image classification, gas sensing, GPS tracking, and wireless communication to identify and respond to fire-related risks. Key components that worked well included the seamless integration of BLE-based gas sensor communication, real-time GPS retrieval via the Termux API, and reliable alert delivery through Pushbullet. The multi-modal approach—requiring both visual and environmental confirmation before triggering an alert—significantly improved the credibility of detections and aligned well with the principles of pervasive computing. However, certain aspects of the system fell short of expectations. The image classification model, while accurate in test environments, struggled with generalization during live inference, particularly in low-light or cluttered scenes. Additionally, the BLE communication was occasionally disrupted due to improper handler termination on the Nano side, indicating a need for more robust error handling.

Continued.

Despite these challenges, the foundational architecture proved effective and scalable. The solution could be improved through enhanced data collection strategies to better train the model on real-world conditions, including diverse lighting, camera angles, and background complexity. Reintegration of the thermal camera would significantly strengthen detection in visually limited environments. Furthermore, adding LoRa for long-range communication and GSM modules for SMS-based alerts would expand the system’s applicability in remote or disaster-affected regions with no mobile or Wi-Fi connectivity. Overall, this project lays a strong groundwork for a deployable, intelligent fire detection system capable of autonomous monitoring and resilient alerting in pervasive computing environments.

References

[1] Y. Yu, Y. Luo, L. Wang, X. Xu and G. Liu, “A UAV-Based Real-Time Wildfire Detection and Monitoring System Using YOLOv5 and Thermal Imaging,” Sensors, vol. 21, no. 22, pp. 7618, Nov. 2021. doi: 10.3390/s21227618

[2] K. A. Ghamry, Y. Zhang and A. A. Farag, “Cooperative Forest Monitoring and Fire Detection Using a Team of UAVs,” in Procedia Computer Science, vol. 95, pp. 475–482, 2016. doi: 10.1016/j.procs.2016.09.360

[3] G. Manogaran, C. Thota, D. Lopez, R. Sundarasekar and V. Balas, “Big Data Knowledge System in Fire Disaster Management Using Internet of Things,” Computers & Electrical Engineering, vol. 65, pp. 111–121, Jan. 2018. doi: 10.1016/j.compeleceng.2017.05.013