Predictive wildfire risk intelligence

An IoT-enabled machine learning framework for pre-ignition hazard forecasting

[1]Pavithra Suresh, [2]Chaithanya G.R

[1,2]Department of Computer Science and Engineering,
[1,2]Sir M. Visvesvaraya Institute of Technology, Bengaluru, Karnataka, India

***Abstract*—Forest-fire management benefits from continuous, location-specific estimates of ignition risk, yet many deployed systems either watch a single point or react only after smoke or heat has been sensed. This paper describes WMS, an IoT and machine-learning platform that estimates wildfire risk for a primary site and for any number of locations registered by the operator. An ESP32 board with a DHT22 sensor posts temperature and humidity every 5 s to a Flask service, which completes each record with wind speed and rainfall from the Open-Meteo forecast API and evaluates a logistic-regression model trained on the 244-record Algerian Forest Fires dataset using four weather inputs. On a stratified hold-out set of 49 records the model reached an accuracy of 0.816 and an ROC-AUC of 0.861; the fire probability is mapped to LOW, MEDIUM or HIGH risk at 0.40 and 0.70. A background poller refreshes every registered location every 30 s, and a token-protected React dashboard presents live cards, a risk gauge, a numbered OpenStreetMap view and one history chart per location. The paper documents the data flow, equations, caching and fallback rules, the multi-location data model, access control and the defects found during development. Results are reported for the model and for a four-location demonstration; the screenshots were captured with the ESP32 offline, so the sensor path is described but not field-validated. The work is a prototype and makes no claim of operational accuracy beyond its training data.**

***Index Terms*—Wildfire risk, IoT, ESP32, DHT22, logistic regression, Open-Meteo, multi-location monitoring, Flask, React, Leaflet**

I. Introduction

Whether a forest fire starts, and how far it travels, depends heavily on the weather of the preceding hours and days. Warm air, low relative humidity, wind and the absence of rain dry out surface fuel, and the Canadian Fire Weather Index System was built around exactly these quantities [1]. Because they can be measured before any flame exists, a monitoring system can raise a warning from conditions instead of waiting until smoke or heat has been sensed.

Much of the sensing literature takes the second route. Wireless sensor networks have been proposed to detect fires in real time [2]–[4], and data-driven studies have fitted statistical models to meteorological records to predict fire occurrence or burned area [5]–[7]. Two gaps remain for a small team that wants something deployable. Detection networks need dense hardware, while most prediction studies stop at an offline model and do not supply a running service, a map or an operator interface. In addition, few of them let an operator watch several places at once.

This paper describes WMS (Wildfire Monitoring System), built as a final-year project. WMS follows a hybrid sensing principle. An ESP32 with a DHT22 sensor supplies local temperature and humidity; a Flask service completes each record with wind speed and rainfall from the Open-Meteo forecast API and evaluates a logistic-regression model. The same service can fetch a full record for any latitude–longitude pair, so additional places can be monitored without additional hardware. The contributions are:

- a four-input risk model whose features are restricted to what the live system can actually provide, with every equation and learned parameter reported;
- a data-fusion rule with a per-site cache, a request timeout and a two-level fallback, so that a prediction is always returned;
- a multi-location registry with independent weather caches and histories, one polling thread, and a dashboard that shows each location on a numbered map, as a card and as a chart;
- token-based protection of configuration actions that still leaves the sensor endpoint open to a microcontroller; and
- an account of the defects found while building the system, with an explicit statement of what has and has not been validated.

Section II reviews related work, Section III states the problem, Section IV describes the architecture and hardware, Section V the dataset and Section VI the learning pipeline. Sections VII and VIII describe the backend and the dashboard, Section IX reports results, Section X lists limitations and Section XI concludes.

II. Related Work

A. Sensor-Based Fire Detection

Yu *et al.* [2] proposed a wireless-sensor-network paradigm for real-time forest-fire detection in which nodes gather environmental readings and forward them to a base station. Hefeeda and Bagheri [3] treated early detection as a node-coverage problem and evaluated a network design for it, and Dampage *et al.* [4] combined a sensor network with machine learning and reported trials in tropical forest sites with lower alert latency than existing systems. These systems respond to the physical effects of a fire at the node. WMS instead estimates risk from the weather state and therefore needs only modest hardware.

B. Weather-Based Risk Estimation

The Fire Weather Index System converts daily weather into fuel-moisture codes and spread indices [1]; its codes depend on the codes of earlier days, which a stateless sensing node does not have. Cortez and Morais [5] applied data-mining techniques to meteorological records from a Portuguese park, and Abid and Izeboudjen [6] trained a decision tree on the Algerian dataset that is also used here. Jain *et al.* [7] survey machine-learning applications in wildfire science and call for high-quality shared data. WMS adopts a deliberately simple classifier, logistic regression, because its coefficients can be read directly and its inference cost is negligible on a small server. It therefore sits between detection networks and offline prediction studies: a running, location-aware risk service assembled from low-cost parts.

III. Problem Formulation

Let $\mathcal{L} = \{\ell_0, \ell_1, \ldots, \ell_n\}$ be the set of monitored locations, where ℓ_0 is the primary site. Each location has a name and coordinates (φ_k, λ_k) with $-90° \le \varphi_k \le 90°$ and $-180° \le \lambda_k \le 180°$. At any instant the weather state of a location is described by the vector

$$\mathbf{x} = (T,\ H,\ W,\ R)^{\top} \qquad (1)$$

where T is air temperature in °C, H is relative humidity in %, W is wind speed in km/h and R is precipitation in mm. For ℓ_0, T and H may come from the DHT22; every other component, and every component of ℓ_k ($k \ge 1$), comes from the weather API. The system must return for each location the probability $p = \Pr(\text{fire} \mid \mathbf{x})$ and a categorical risk level $\rho(p) \in \{\text{LOW, MEDIUM, HIGH}\}$, refresh both periodically and expose them to an operator interface. The mapping from $\mathbf{x}$ to p is learned in Section VI, and ρ is defined in Eq. 7.

IV. System Architecture and Hardware

A. Layered Architecture

WMS has four layers (Fig. 1). The sensing layer is the ESP32 with the DHT22. The backend is a single Flask process that exposes a REST interface and contains a weather module, the machine-learning engine, a polling thread and token authentication. The data layer holds readings and configuration, and the presentation layer is a React application that talks to the backend over HTTP and JSON. Data enter along two independent paths: the ESP32 posts a reading about every 5 s, and the poller builds a complete weather-based reading for every registered location every 30 s. The poller never marks the ESP32 as online, so the interface cannot show a connected device that is not there.

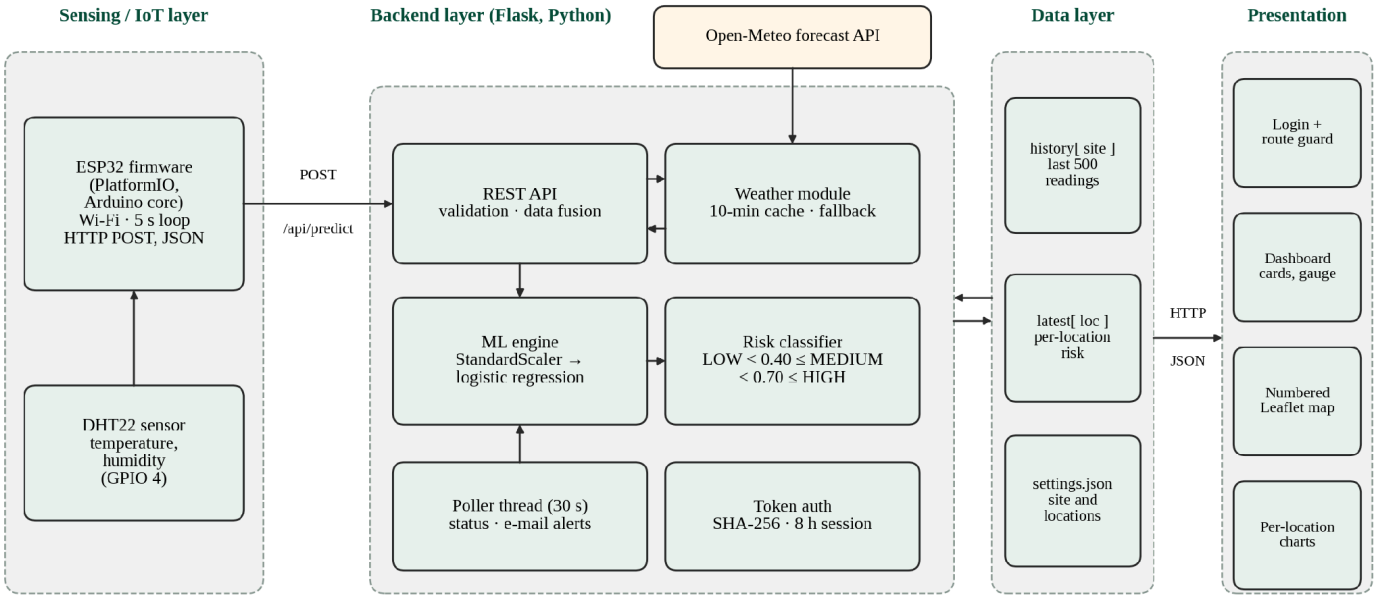

Fig. 1. Layered architecture of WMS. Solid arrows show request flow; the poller drives a second, hardware-free data path.

B. Sensing Node and Firmware

The node uses a standard ESP32 development board (the esp32dev profile) [8] and a DHT22 temperature–humidity sensor [9] whose data line is wired to GPIO 4. The firmware targets the Arduino core and is built and uploaded with PlatformIO [10]; it depends on the Adafruit DHT sensor library (version 1.4.4 or later), and the serial monitor runs at 115200 baud. A non-blocking timer based on millis() starts a cycle every READ_INTERVAL_MS = 5000 ms. In each cycle the firmware reads humidity and temperature. If either value is not a number it reports the failure on the serial port and skips the cycle; otherwise it builds a JSON body such as {"temperature":28.6,"humidity":40.0}, with one decimal place, and sends it by HTTP POST to the /api/predict endpoint of the Flask host. Before each transmission it checks the Wi-Fi state and, if the link has dropped, restarts the connection with a retry every 400 ms. The HTTP status and response body are echoed on the serial port for debugging. Wind speed and rainfall are deliberately absent from the payload.

Two design choices are worth recording. An MQ-2 gas sensor was part of an early design but was dropped, because smoke is diluted in open air and a single point sensor is unreliable outdoors. A pulse-counting anemometer and a tipping-bucket rain gauge were also considered; the training data record wind and rain as daily weather values and a regional forecast was judged adequate for those two quantities, so no wind or rain hardware is fitted.

C. Software Stack

Table 1 lists the components and their roles. Versions are those pinned in the project manifests; the HTTP client library is used unpinned.

Table 1 Software and Hardware Components

Layer	Technology	Role
Node	ESP32 (esp32dev), DHT22	T, H; Wi-Fi client
Firmware	Arduino core, PlatformIO, Adafruit DHT ≥ 1.4.4	5 s loop, HTTP POST
Backend	Python, Flask 3.0.3, Flask-CORS 4.0.1, requests	REST API, polling
ML	scikit-learn 1.5.1, pandas 2.2.2, NumPy 1.26.4, joblib 1.4.2	training, inference
Weather	Open-Meteo forecast API	T, H, W, R
Front end	React 19, React Router 7, Vite 5.4, Recharts 3.10	SPA, charts
Map	Leaflet 1.9, react-leaflet 5, OpenStreetMap tiles	numbered markers
Storage	process memory, settings.json	histories; configuration

V. Dataset and Feature Design

The model is trained on the Algerian Forest Fires dataset [11], which was also used in [6]. It contains 244 daily records, 122 for each of two regions of northern Algeria (Bejaïa and Sidi Bel-Abbès), covering June to September 2012. Each record carries the date, four weather measurements (temperature, relative humidity, wind speed, rain), six indices of the Fire Weather System (FFMC, DMC, DC, ISI, BUI, FWI) and a binary class, fire or not fire. There are 138 fire days (56.6 %) and 106 non-fire days (43.4 %).

Preprocessing is short. Column names are stripped of white space; the class text is stripped, lower-cased and encoded as fire = 1; the four weather columns are coerced to numbers, and rows with a missing value would be dropped (none were, so all 244 records are used). The date and the six index columns are discarded. The indices are accumulated from the moisture codes of earlier days, which WMS cannot compute from one instantaneous reading, and training on them would make the model depend on inputs that do not exist at prediction time.

Table 2 summarizes the four retained features. Fire days are on average 3.8 °C warmer, 13.0 percentage points drier and almost free of rain (0.10 mm against 1.62 mm), whereas mean wind speed is nearly the same in both classes (15.3 and 15.7 km/h). Rain is strongly skewed: its median is 0 mm and its 75th percentile 0.5 mm.

Table 2 Retained Features of the Training Data

Feature	Range	Mean	Mean, fire	Mean, no fire
Temperature (°C)	22–42	32.17	33.82	30.03
Humidity (%)	21–90	61.94	56.28	69.31
Wind speed (km/h)	6–29	15.50	15.34	15.72
Rain (mm)	0–16.8	0.76	0.10	1.62

VI. Machine-Learning Pipeline

A. Split and Standardization

The 244 records are divided into 195 training and 49 test records by a stratified split (test fraction 0.2, random seed 42). The training set holds 110 fire and 85 non-fire days and the test set 28 and 21. Each feature is standardized with statistics from the training set only, as in Eq. 2, where μ_j and s_j are the mean and standard deviation of feature j. The fitted scaler is stored with joblib (scaler.pkl) and used unchanged at prediction time.

$$z_j = \frac{x_j - \mu_j}{s_j} \tag{2}$$

B. Logistic Regression

The classifier is a logistic regression [12], [13] implemented in scikit-learn [14] with default settings (L2 penalty, $C = 1$, L-BFGS solver). For a standardized vector $\mathbf{z}$ the model forms the logit u of Eq. 3 and the fire probability of Eq. 4, where $\mathbf{w}$ and b are the weights and the intercept.

$$u = \sum_{j=1}^{4} w_j z_j + b \tag{3}$$

$$p = \sigma(u) = \frac{1}{1 + e^{-u}} \tag{4}$$

Training minimizes the L2-regularized logistic loss of Eq. 5 over the $n = 195$ training records, with labels coded as $y_i \in \{-1, +1\}$.

$$J(\mathbf{w}, b) = \tfrac{1}{2}\|\mathbf{w}\|^2 + C \sum_{i=1}^{n} \ln\left(1 + e^{-y_i(\mathbf{w}^\top \mathbf{z}_i + b)}\right) \tag{5}$$

The fitted model (model.pkl) has the parameters in Table 3. The odds ratio of Eq. 6 is the factor by which the odds of fire change when a feature rises by one standard deviation and the others stay fixed.

$$\mathrm{OR}_j = e^{w_j} \tag{6}$$

Table 3 Standardization and Learned Parameters

Feature	μ	s	w	e^w
Temperature	32.2513	3.659	+1.1601	3.19
Humidity	61.4718	15.0153	−0.5429	0.58
Wind speed	15.3949	2.8035	+0.2232	1.25
Rain	0.6456	1.7863	−3.4333	0.03
Intercept b	—	—	−0.2685	—

Higher temperature and wind raise the odds of fire; higher humidity and, above all, rain lower them. The rain weight is large because the feature is concentrated at zero, so a wet day of several millimetres is a several-standard-deviation event. The magnitudes should be read as properties of this dataset rather than universal sensitivities.

C. Risk Bands and Displayed Confidence

The probability is converted to a risk level by Eq. 7. The thresholds 0.40 and 0.70 were fixed at design time and were not tuned on the test set. The dashboard value labelled “AI Confidence” is $p \times 100$ rounded to one decimal place, that is, the model’s fire probability, not a calibrated certainty about the displayed level.

$$\rho(p) = \begin{cases} \text{LOW}, & p < 0.40 \\ \text{MEDIUM}, & 0.40 \le p < 0.70 \\ \text{HIGH}, & p \ge 0.70 \end{cases} \tag{7}$$

D. Evaluation Measures

Performance on the held-out set is summarized by precision P, recall R, the F1 score and accuracy (Eqs. 8–9), where TP, FP, FN and TN are the entries of the confusion matrix and $N = 49$, and by the area

under the ROC curve [15] (Eq. 10), the probability that a random fire day receives a higher p than a random non-fire day.

$$P = \frac{TP}{TP + FP}, \qquad R = \frac{TP}{TP + FN} \tag{8}$$

$$F_1 = \frac{2PR}{P + R}, \qquad \mathrm{Acc} = \frac{TP + TN}{N} \tag{9}$$

$$\mathrm{AUC} = \Pr(p_{\text{fire}} > p_{\text{no fire}}) \tag{10}$$

E. Worked Example

For the primary-site reading shown in Fig. 4 (T = 28.6 °C, H = 40 %, W = 17 km/h, R = 0 mm), standardization with Table 3 gives $\mathbf{z}$ = (−0.998, −1.430, 0.573, −0.361). The weighted terms are −1.158, +0.776, +0.128 and +1.241, which sum to 0.987; adding b = −0.269 gives u = 0.719 and hence $p = 1/(1 + e^{-0.719}) = 0.672$. Because $0.40 \le p < 0.70$ the level is MEDIUM, and the dashboard shows 67.2 %.

VII. Backend Implementation

A. REST Interface

Table 4 lists the endpoints. Sensor ingestion and all read endpoints are open; the endpoints that change configuration need a session token (Section VII-G).

Table 4 REST Endpoints

Endpoint	Auth	Purpose
POST /api/predict	none	Accept a reading (any of T, H, W, R), fuse, predict, store
GET /api/latest	none	Latest primary-site prediction
GET /api/history (limit, location_id)	none	Last n records of the primary site or of a monitored location
GET /api/status	none	ESP32 online flag, age of last post, filled fields
GET /api/predict-locations	none	Primary site and all locations with latest prediction
GET /api/locations	none	List monitored locations
POST, DELETE /api/locations	token	Add or remove a location
GET, POST /api/settings	POST	Read or set primary coordinates
POST /api/login, GET /api/verify-token, POST /api/logout	—	Session handling

B. Data Fusion and Weather Retrieval

A POST to /api/predict may contain any subset of the four fields. Each missing field is filled by the weather module, and the record is tagged with its origin: “sensor” if all four fields came from the client, “api” if at least one was filled in, and “api_auto” if the poller created it. The four values are then cast to floating point (a non-numeric value returns HTTP 400), scaled, evaluated and appended to the primary history (Fig. 2a).

The weather module calls the Open-Meteo forecast endpoint [16], [17] with the latitude and longitude of the site and the current variables temperature_2m, wind_speed_10m, precipitation and relative_humidity_2m, returned in °C, km/h, mm and %. The request has a 5 s timeout. Each site owns a cache entry that is reused

while Eq. 11 holds, with τ_c = 600 s. If a request fails, the last cached tuple is returned; if the site has never been fetched successfully, the defaults 25 °C, 0 km/h, 0 mm and 50 % are used, so a prediction is always produced. Under this rule a site causes at most 3600/600 = 6 API calls per hour although the poller evaluates it 120 times per hour.

$$t - t_{\text{fetch}} < \tau_c, \qquad \tau_c = 600 \text{ s} \tag{11}$$

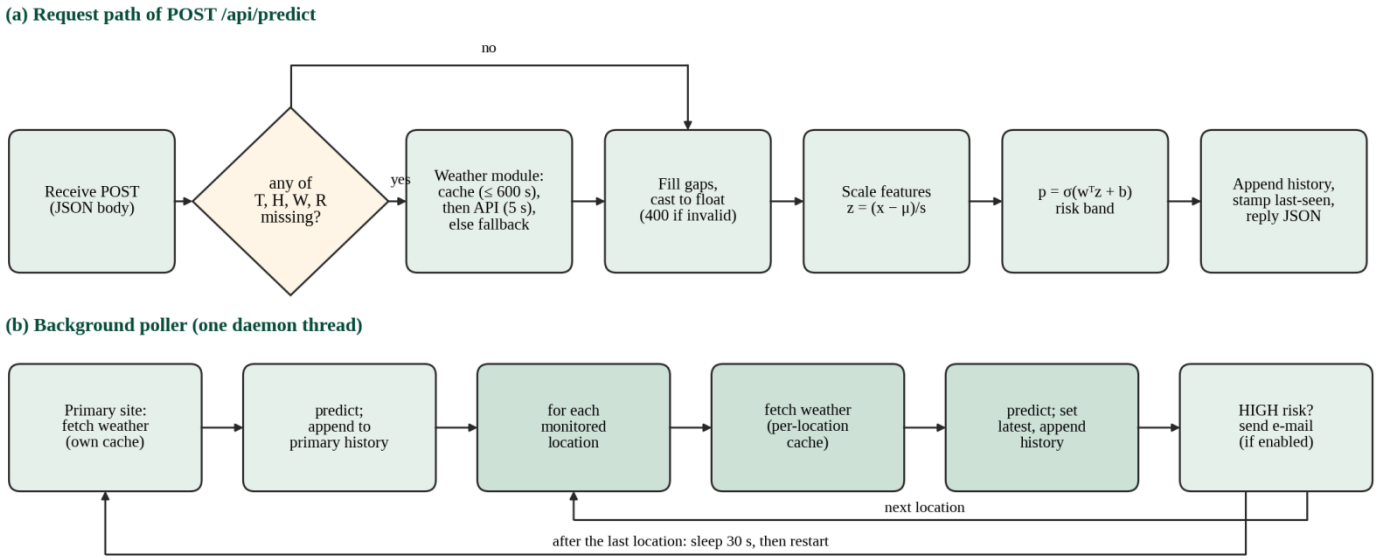

Fig. 2. (a) Processing of a POST to /api/predict. (b) Behaviour of the background poller.

C. Device Status

The backend records the time of the last real POST. The ESP32 is reported as online while Eq. 12 holds, that is, while its last post is not older than three posting intervals; the poller does not update this time. The status endpoint also returns the number of non-empty fields in the latest record, out of four.

$$\text{online} \Leftrightarrow t_{\text{now}} - t_{\text{last}} \leq 15 \text{ s} \tag{12}$$

D. Background Poller

One daemon thread starts with the server and repeats the cycle of Fig. 2(b) every 30 s. It fetches and evaluates the primary site and then every monitored location in turn, each with its own cache. For monitored locations it stores the latest result and appends it to a per-location history; a failure for one location is caught and logged, so it cannot stop the others. The loop iterates over a copy of the location list because locations can be added or removed while it runs. When a result is HIGH and alerts are enabled, an e-mail is sent through SMTP with STARTTLS on port 587; alerts are disabled by default. Flask's debug reloader is switched off (use_reloader=False), because with the reloader the module ran in two processes and two pollers appeared.

E. History Storage

Histories are Python lists capped at 500 records per site, the oldest being discarded, and are guarded by a threading lock. The latest-result cache, the per-location history and the per-location weather cache are dictionaries keyed by the location identifier. Readings are therefore lost when the server stops, whereas configuration is persisted as described next.

F. Multi-Location Registry

A location is a record {id, name, latitude, longitude}. On creation the name must be non-empty and both coordinates numeric and within the ranges of Section III, otherwise HTTP 400 is returned. The identifier is the lower-cased name with every run of non-alphanumeric characters replaced by a hyphen, followed by a hyphen and four hexadecimal digits from secrets.token_hex(2), which keeps identifiers

readable and distinct for repeated names. Locations are stored in settings.json under the key monitored_locations, next to the primary latitude and longitude. Every write first reads the file, updates only its own keys and writes it back under a lock, so saving the primary coordinates cannot erase the location list or the reverse. Changing the primary coordinates also resets the primary cache time so that the next request fetches new weather.

The endpoint /api/predict-locations returns an array whose first element is the primary site (identifier "primary", flagged is_primary) followed by the monitored locations in insertion order, each with its latest prediction. The position in this array defines the number shown on the map, in the cards and in the charts. /api/history accepts location_id to select the history of a monitored location and defaults to the primary site.

G. Access Control

Access control is intentionally light-weight. A single administrator account is defined; the submitted password is hashed with SHA-256 and compared with the stored digest, so the plain password is not kept in the source. A successful login issues a session token of 32 random bytes (64 hexadecimal characters, 256 bits) from secrets.token_hex(32), valid for 8 h and held in server memory. A decorator checks the Authorization: Bearer header of the protected routes (adding or deleting a location, and changing the primary coordinates) and returns 401 otherwise; logout deletes the token. The ingestion endpoint is not protected because the microcontroller cannot perform a login, and read endpoints are open in this prototype.

VIII. Dashboard Implementation

A. Application Structure

The front end is a single-page React application built with Vite. React Router defines four routes: the landing page (Fig. 3), the login page, the dashboard and a page that explains the architecture. The dashboard route is wrapped in a guard that reads the token from the browser's local storage, asks /api/verify-token whether it is still valid and redirects to the login page if not. The landing page shows a demonstration gauge that cycles through preset values; it is a simulated reading and is labelled as such.

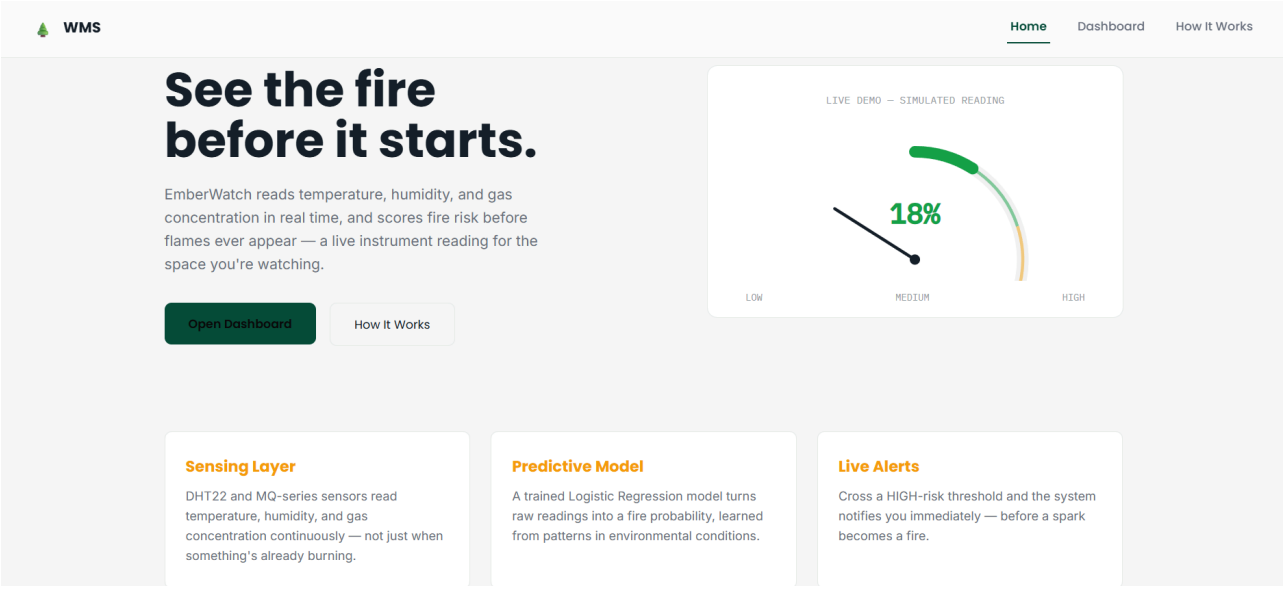

Fig. 3. Home page of the WMS web application with the demonstration risk gauge.

B. Primary-Site Dashboard

The dashboard (Fig. 4) has a fixed sidebar whose entries scroll to the corresponding sections, a header with a live clock and a status label derived from the ESP32 status and, below a label that identifies them as the primary site, four sensor cards for temperature, humidity, wind speed and rain. A prediction card shows the risk level with a colour-coded shield, the probability and a recommendation that depends on the level. A device-status panel lists the ESP32, the DHT22, the weather API and Wi-Fi. The panel in Fig. 4

shows the weather API online and the three hardware items offline, because the screenshot was taken while the ESP32 was powered off; the readings come from the poller. The page polls /api/latest, /api/history and /api/status every 5 s.

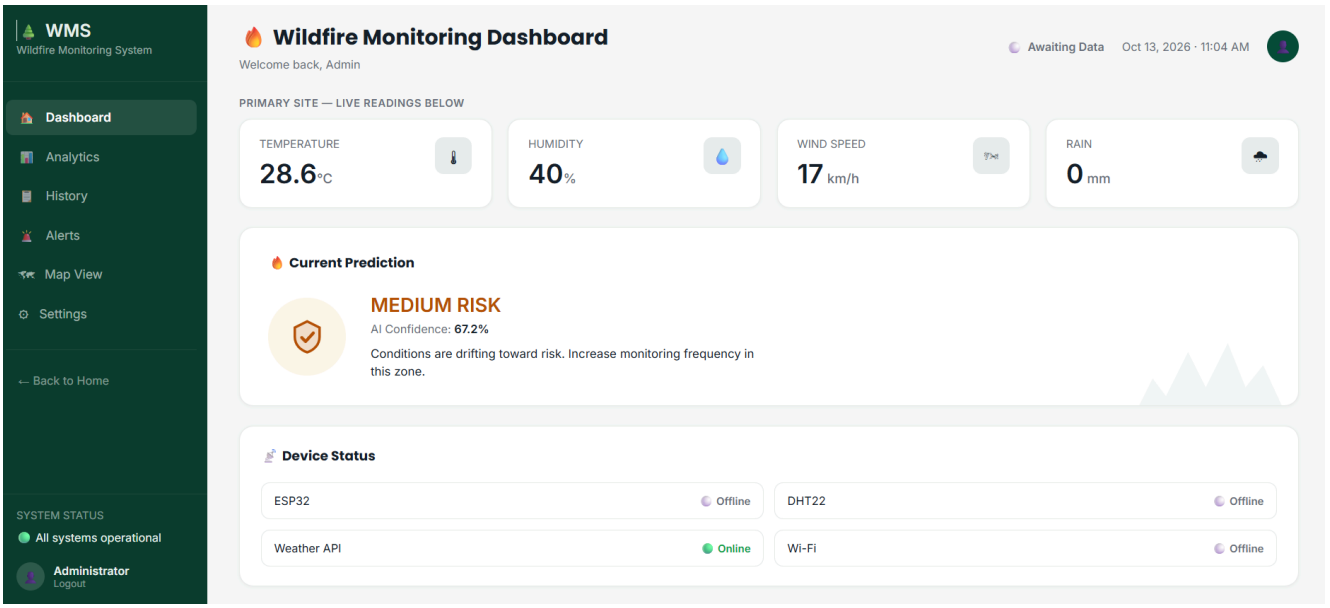

Fig. 4. Primary-site dashboard: sensor cards, MEDIUM-risk prediction (67.2 %) and device status (ESP32 offline, weather API online).

The gauge on the landing page is an SVG semicircle. With p in percent, the needle angle and tip are given by Eqs. 13 and 14, where (c_x, c_y) is the pivot and r the radius; the arc colour changes at 40 and 70 to match Eq. 7.

$$\theta = -90^\circ + 1.8\,p \tag{13}$$

$$(c_x + 0.88\,r\sin\theta,\;\; c_y - 0.88\,r\cos\theta) \tag{14}$$

C. Location Entry

The Settings section (Fig. 5) has two forms. The first edits the primary latitude and longitude and posts them to /api/settings. The second, “Monitored Locations”, takes a name, a latitude and a longitude and posts them to /api/locations with the bearer token. Each stored location is listed with its number (the primary site is number 1, so the first added location is number 2) and a Remove button. Network failures and server errors are shown beside the form.

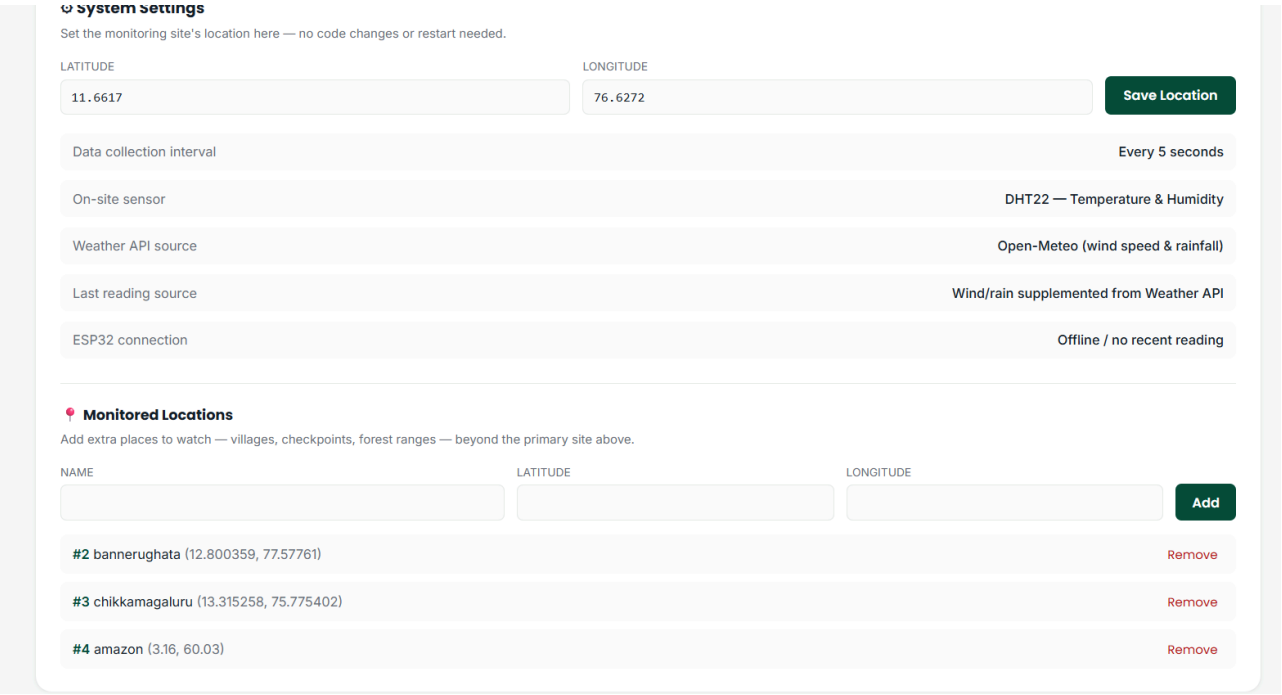

Fig. 5. Settings section: primary-site coordinates and the Monitored Locations form with three added locations (#2–#4).

D. Numbered Map

The map (Fig. 6) uses Leaflet [18] with OpenStreetMap tiles [19]. Each location returned by /api/predict-locations becomes a circular marker whose label is its array position plus one and whose colour is green, amber or red for LOW, MEDIUM or HIGH (grey before the first reading). A popup gives the name, level and probability. The list under the map repeats the numbers, names and probabilities, and selecting a row flies the map to that marker (zoom 11, 0.8 s). The map is centred on the primary site at zoom 9 and refreshes every 15 s. Fig. 6 is zoomed on the primary site, which lies near the Karnataka–Tamil Nadu border, and the list gives all four locations.

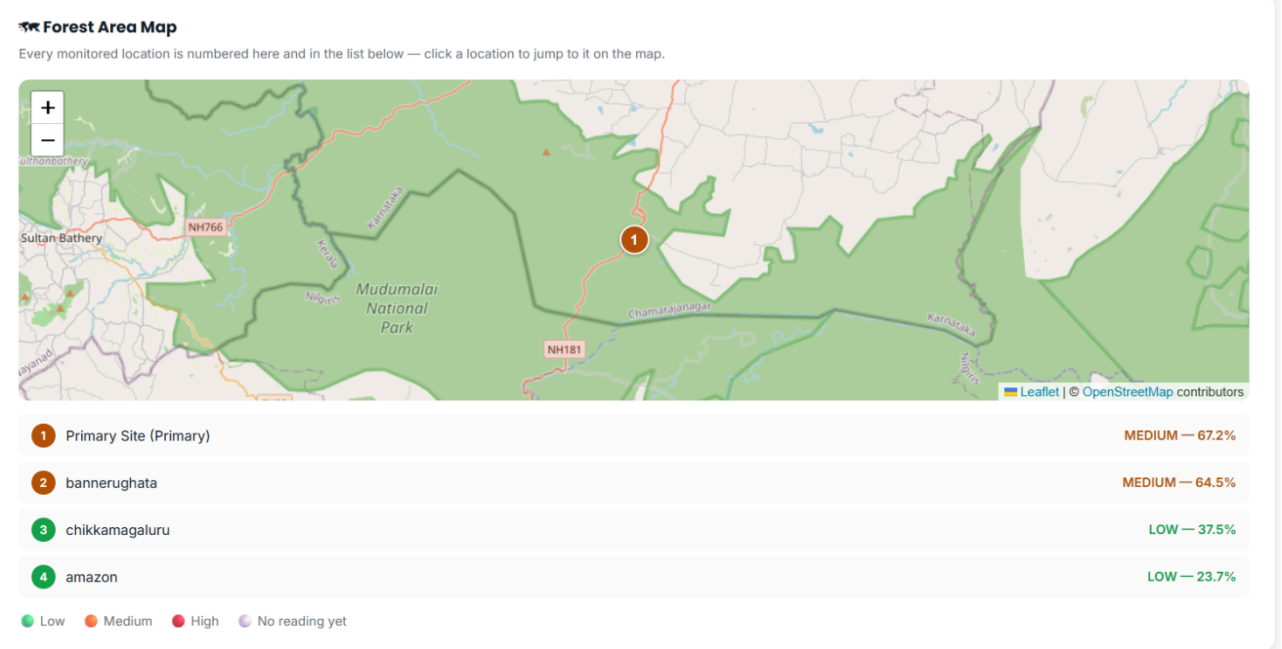

Fig. 6. Forest-area map with a numbered marker for the primary site and the numbered location list with risk level and probability.

E. Location Cards and History Charts

Below the map, one card per location shows the four weather values as compact tiles, a risk badge and the probability; the cards use the same numbers as the markers. A second block draws one line chart per

location with the four series for the latest 30 readings of /api/history?location_id=…, refreshed every 20 s (Fig. 7). The series use muted green, slate and tan tones so that red, amber and green stay reserved for risk levels. The interface otherwise follows a light layout with a single forest-green accent (#064E3B).

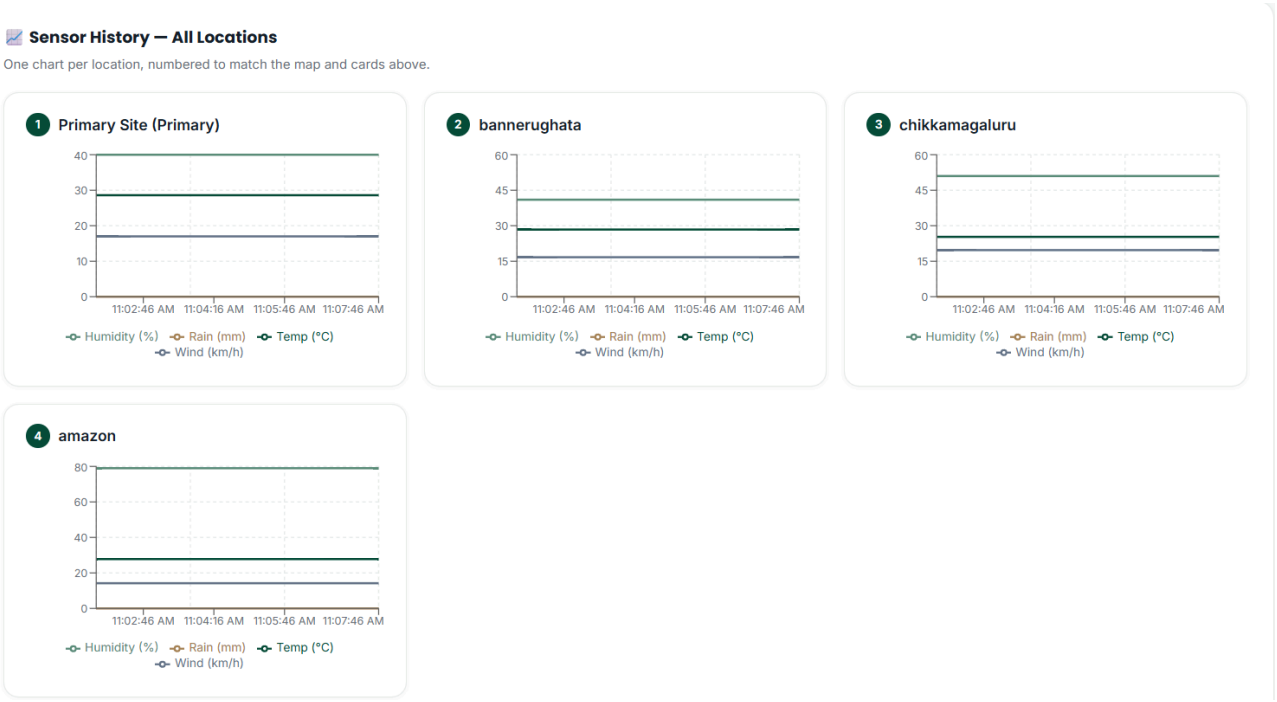

Fig. 7. History charts, one per location and numbered as in Fig. 6 (temperature, humidity, wind speed, rain).

IX. Results and Discussion

A. Classification Performance

Table 5 and Fig. 8 give the results on the 49 test records. Accuracy is 0.816 (40 of 49 correct) and the ROC-AUC is 0.861; training accuracy is 0.831, so there is no large generalization gap. Fire days are retrieved with recall 0.821 and precision 0.852. Five of the 28 fire days were predicted as no fire and four of the 21 non-fire days as fire (Fig. 8b). With only 49 records one case changes accuracy by about two percentage points and a normal-approximation 95 % interval is roughly ±11 points, so the numbers indicate feasibility rather than a precise accuracy.

Table 5 Test-Set Performance (n = 49)

Class	Precision	Recall	F1	Support
not fire	0.773	0.810	0.791	21
fire	0.852	0.821	0.836	28
accuracy	—	—	0.816	49
macro average	0.812	0.815	0.814	49
weighted average	0.818	0.816	0.817	49
ROC-AUC	—	—	0.861	—

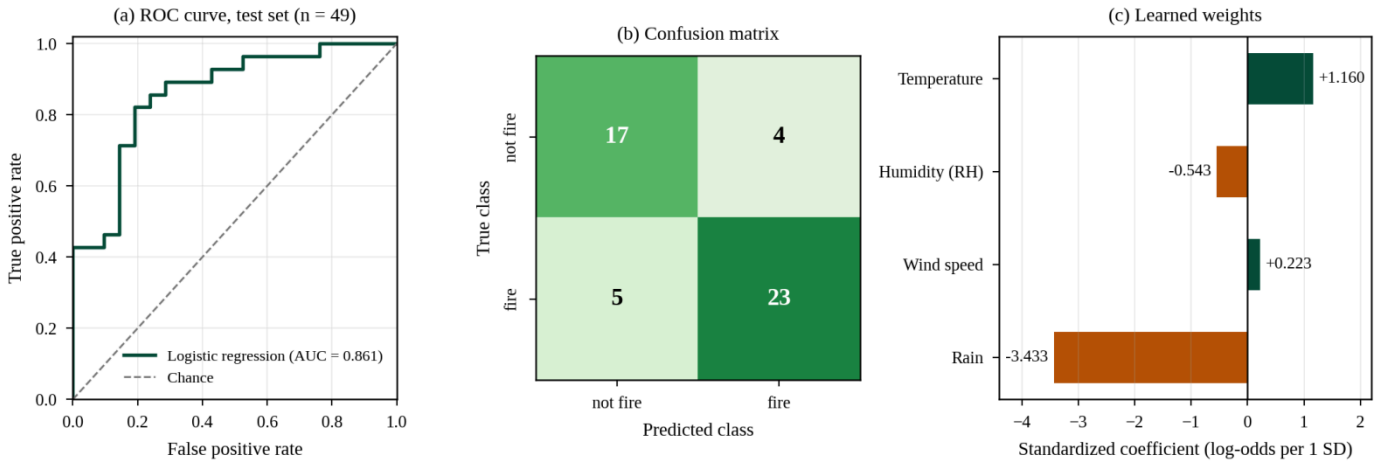

Fig. 8. Model evaluation on the test set: (a) ROC curve, (b) confusion matrix, (c) standardized weights.

B. Behaviour of the Risk Bands

Table 6 classifies the test records with the 0.40 and 0.70 thresholds. The share of fire days rises from 21.1 % in the LOW band to 57.1 % in MEDIUM and 87.0 % in HIGH, so the bands order the cases sensibly. The LOW band still contains four of the 28 fire days (14.3 %), a miss rate that a deployment would have to accept or counter with a lower threshold.

Table 6 Fire Share per Risk Band, Test Set

Band	Records	Fire days	Fire share
LOW	19	4	21.1 %
MEDIUM	7	4	57.1 %
HIGH	23	20	87.0 %

C. Response to Controlled Inputs

To check the direction of the response, the model was evaluated on constructed inputs (Table 7). Cool, wet air gives a probability of 0.0 % to one decimal, hot, dry and windy air gives 99.0 %, and intermediate conditions fall in the MEDIUM or LOW band, as the signs in Table 3 predict.

Table 7 Model Response to Constructed Inputs

Scenario	T	H	W	R	p (%)	Level
Cool, wet	22	85	8	6.0	0.0	LOW
Hot, dry, windy	38	25	22	0	99.0	HIGH
Borderline	30	55	14	0.3	45.1	MEDIUM
Moderate	29	60	15	0.5	26.9	LOW
Fig. 4 reading	28.6	40	17	0	67.2	MEDIUM

D. Four-Location Demonstration

Four locations were registered through the interface of Fig. 5: the primary site and three additional places entered as bannerughata, chikkamagaluru and amazon (Table 8). The levels are those displayed in Fig. 6. The two Karnataka locations differ. Bannerughata is MEDIUM (64.5 %), close to the primary site, whereas chikkamagaluru is LOW (37.5 %) because the weather API reported higher humidity there (about 51 % against 40–41 %; Fig. 7). The fourth point, in the Amazon basin, has the highest humidity (about 79 %) and the lowest probability, 23.7 %, in agreement with the negative humidity weight. Re-evaluating the

model on values read from the charts reproduces the displayed probabilities to within two percentage points, which is about the precision with which the charts can be read.

Table 8 Registered Locations and Displayed Risk

#	Location	Latitude, longitude	Level	p (%)
1	Primary Site	11.6617, 76.6272	MEDIUM	67.2
2	bannerughata	12.800359, 77.57761	MEDIUM	64.5
3	chikkamagaluru	13.315258, 75.775402	LOW	37.5
4	amazon	3.16, 60.03	LOW	23.7

Two observations about Fig. 7 follow from the design rather than from the weather. First, each trace is flat: the poller calls the model every 30 s, but the weather of each site is cached for 600 s and the plotted window is about five minutes, so consecutive points repeat one cached observation and a trend can appear only after the cache expires. Second, the charts differ from one another, which confirms that each location keeps its own history and cache. The Amazon point lies far outside the climate of the training data, so its probability is an extrapolation, shown only to demonstrate that locations are handled independently.

E. Functional Verification

Functional checks were run during development with Flask's test client and a mocked weather service. They confirmed that (i) adding or deleting a location without a token returns 401 and succeeds with one, and that a logged-out token is rejected; (ii) invalid coordinates return 400 and an unknown identifier 404; (iii) saving the primary coordinates leaves the location list intact; (iv) the histories of different locations start empty, fill independently and are selected by location_id; (v) the ESP32 is reported offline when only the poller has run and online after a POST; and (vi) a weather failure falls back to cached values without raising an error. These are scripted spot checks, not a complete automated test suite.

F. Defects Found and Fixed

Table 9 lists defects that appeared during construction. Several are instructive because they arise from the interaction of components rather than from any single one.

Table 9 Implementation Defects and Fixes

Symptom	Cause	Fix
Prediction failed: scaler expects 3 features	Stale model files from an earlier three-input design	Retrain after any feature change
Duplicate poller messages	Debug reloader ran the module in two processes	use_reloader=False
Adding a location returned 400	Client sent lat/lon, server required latitude/longitude; an older, unused module used the former	One field-name contract
Location list vanished after saving coordinates	Settings file overwritten, not merged	Merge-safe write under a lock
All charts showed the primary site	Server kept history for the primary site only	Per-location history; location_id parameter
Front-end build failed on Windows	Experimental rolldown-based Vite lacked a native binding	Standard Vite 5.4 and matching React plugin
Weather API shown offline at start	Status derived from the presence of a reading; none existed yet	Poller creates readings without hardware

X. Limitations

Data. The model rests on 244 daily records from one season and two Algerian regions. Its transfer to Karnataka or to other climates is untested; the label marks a fire occurring on that day in the region, not ignition at a point; and the weights and thresholds have not been checked against Indian fire records.

Validation. Results come from a single 80/20 split with 49 test records, without cross-validation or probability calibration. The bands are design values, and "AI Confidence" is an uncalibrated probability.

Inputs. Open-Meteo returns model-based estimates on grids of roughly 1–11 km [17], which cannot represent canopy, valley or ridge effects. The sensor path (ESP32 to /api/predict) is implemented and uses the same endpoint as the manual and poller paths, but the ESP32 was offline in every screenshot. Outdoor operation, the network configuration between board and server and long-run stability were not evaluated, and the Wi-Fi credentials are compiled into the firmware.

Security and persistence. The password digest is a single unsalted SHA-256 hash, weaker than the salted, deliberately slow schemes recommended in [20]; traffic is plain HTTP, cross-origin access is open, tokens live in memory and there is one account. Readings are kept in memory only.

Scale. Locations are polled one after another and each fetch can wait up to 5 s, so a cycle can take up to $5n$ s in the worst case; the design suits tens of locations, not hundreds. Risk is estimated per point, and no spatial interpolation, terrain or vegetation is modelled.

XI. Conclusion and Future Work

WMS shows that a small, inexpensive chain—an ESP32 with one sensor, a public weather API, a four-input logistic regression and a web dashboard—can produce a continuously updated, location-aware fire-risk estimate for several places at once. On a 49-record test set the model reached 0.816 accuracy and 0.861 ROC-AUC, the probability bands ordered fire prevalence from 21 % to 87 %, and a four-location demonstration exercised the registry, the poller, the numbered map and the per-location charts. The system is a prototype, and its value lies in the documented pipeline and in the candid account of its limits.

Future work should (i) add cross-validation and a temporal split; (ii) collect fire records for the target region and recalibrate the probabilities, for example by Platt scaling or isotonic regression; (iii) test the sensor path outdoors and add on-site wind and rain sensors; (iv) replace the in-memory store with a database and the token scheme with salted hashing, HTTPS and standard JSON Web Tokens; (v) add push or SMS alerts; and (vi) poll locations concurrently and interpolate risk between them.

Acknowledgment

The authors thank the Department of Information Science and Engineering, Sir M. Visvesvaraya Institute of Technology, Bengaluru, for the laboratory facilities and guidance during this work.

References

[1] C. E. Van Wagner, "Development and structure of the Canadian Forest Fire Weather Index System," Canadian Forestry Service, Ottawa, ON, Canada, Forestry Tech. Rep. 35, 1987.

[2] L. Yu, N. Wang, and X. Meng, "Real-time forest fire detection with wireless sensor networks," in *Proc. Int. Conf. Wireless Commun., Netw. Mobile Comput.*, vol. 2, Wuhan, China, 2005, pp. 1214–1217.

[3] M. Hefeeda and M. Bagheri, "Forest fire modeling and early detection using wireless sensor networks," *Ad Hoc Sensor Wireless Netw.*, vol. 7, pp. 169–224, 2009.

[4] U. Dampage, L. Bandaranayake, R. Wanasinghe, K. Kottahachchi, and B. Jayasanka, "Forest fire detection system using wireless sensor networks and machine learning," *Sci. Rep.*, vol. 12, Art. no. 46, 2022, doi: 10.1038/s41598-021-03882-9.

[5] P. Cortez and A. Morais, "A data mining approach to predict forest fires using meteorological data," in *Proc. 13th Portuguese Conf. Artif. Intell. (EPIA)*, Guimarães, Portugal, 2007, pp. 512–523.

[6] F. Abid and N. Izeboudjen, "Predicting forest fire in Algeria using data mining techniques: Case study of the decision tree algorithm," in *Advanced Intelligent Systems for Sustainable Development (AI2SD'2019)*. Cham, Switzerland: Springer, 2020.

[7] P. Jain, S. C. P. Coogan, S. G. Subramanian, M. Crowley, S. Taylor, and M. D. Flannigan, "A review of machine learning applications in wildfire science and management," *Environ. Rev.*, vol. 28, no. 4, pp. 478–505, 2020.

[8] Espressif Systems, "ESP32 series datasheet," Shanghai, China.

[9] Aosong Electronics, "Digital-output relative humidity and temperature sensor/module DHT22 (AM2302)," datasheet, Guangzhou, China.

[10] PlatformIO Labs, "PlatformIO documentation." [Online]. Available: https://docs.platformio.org (accessed Oct. 2026).

[11] "Algerian Forest Fires Dataset," UCI Machine Learning Repository, 2019. [Online]. Available: https://archive.ics.uci.edu/dataset/547 (accessed Oct. 2026).

[12] D. R. Cox, "The regression analysis of binary sequences," *J. Roy. Statist. Soc. B*, vol. 20, no. 2, pp. 215–242, 1958.

[13] D. W. Hosmer, S. Lemeshow, and R. X. Sturdivant, *Applied Logistic Regression*, 3rd ed. Hoboken, NJ, USA: Wiley, 2013.

[14] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in Python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, 2011.

[15] T. Fawcett, "An introduction to ROC analysis," *Pattern Recognit. Lett.*, vol. 27, no. 8, pp. 861–874, 2006.

[16] P. Zippenfenig, "Open-Meteo.com weather API," Zenodo, 2023, doi: 10.5281/zenodo.7970649.

[17] Open-Meteo, "Free open-source weather API and documentation." [Online]. Available: https://open-meteo.com (accessed Oct. 2026).

[18] V. Agafonkin, "Leaflet: An open-source JavaScript library for interactive maps." [Online]. Available: https://leafletjs.com (accessed Oct. 2026).

[19] OpenStreetMap contributors, "OpenStreetMap." [Online]. Available: https://www.openstreetmap.org (accessed Oct. 2026).

[20] OWASP Foundation, "Password storage cheat sheet." [Online]. Available: https://cheatsheetseries.owasp.org (accessed Oct. 2026).