Operation AIRFlag: ESP32-Based Risk Monitoring with Local LLM-Assisted Diagnostics

Ankit Kumar Mishra , Arya Singha
Ch. Sri Sai Lohit , Malhar Sanjaybhai Mer
Dr.Aravinda H L, Associate professor, Department of CSE, Dr.AIT
Department of Computer Science and Engineering, Dr. Ambedkar Institute of Technology, Bengaluru, India

Abstract—Operation AIRFlag is an educational monitoring tool that uses an ESP32 sensor node, a Python-based telemetry gateway, deterministic risk scoring, a Streamlit dashboard, and an optional locally hosted large language model for diagnostic explanation. The hardware used includes DHT11, BMP280, MPU6050, and an infrared (IR) obstacle input. The ESP32 sends newline-delimited JSON over USB serial and a separate UART2 display packet. The main telemetry session has 450 records over 898.063 seconds (14.97 minutes). All records are valid, and all four sensor statuses are reported as OK. There are 439 records, two tilt-related GUARDED records, and nine IR-related HIGH records. The temperature is between 28.2 and 28.6 degrees Celsius, so this session does not reach the 40 degree Celsius alert limit. An earlier session with 46 records is kept separately and has three IR-related HIGH records. A separate SQLite database holds 162 LLM benchmark tests across three models. The benchmark checks execution and some software checks, not the accuracy of the diagnostic explanations. These results show the behavior of the prototype only. They do not establish sensor calibration, validated aviation hazard detection, or correctly timed pilot-crew-ATC escalation.

Keywords—*ESP32, telemetry monitoring, deterministic risk scoring, local large language model, Ollama, sensor validation, event logging, embedded alerting.*

I. INTRODUCTION

Small embedded monitoring systems are becoming more available because cheap microcontrollers can gather motion data and send it to higher-level software. In aviation research, anomaly detection has been examined using flight data and data-driven techniques. Li et al. studied cluster-based detection of operations using flight recorder data, while Coelho e Silva and Murça created a flight-data analytics framework that combines detection with explanation [5], [6]. These studies work on aviation datasets and use domain-specific evaluation procedures. AIRFlag has a purpose: it is a laboratory prototype designed to show a sensor-to-dashboard pipeline and local diagnostic help, not an aviation-certified monitor.

The current AIRFlag uses one ESP32 node. The sensor set kept by AIRFlag is DHT11, BMP280, MPU6050, and an IR obstacle module. The ESP32 firmware gathers values, shows an indicator/buzzer interface, and sends JSON telemetry to a Python gateway. A separate UART2 output is ready for an Arduino UNO TFT display. The Python application checks inputs, computes a risk score, saves telemetry and state-transition events in SQLite, and shows the data on a Streamlit dashboard. An optional local LLM can create a diagnostic explanation, but AIRFlag validates that output and cannot change the official Python risk state.

The purpose of this paper is to describe the architecture that AIRFlag uses and to report the evidence found in the project files, the supplied telemetry CSV, and the retained benchmark database. Therefore, the results are framed as functional observations. No claim is made that the readings observed by AIRFlag represent aircraft flight conditions or verified hazards. I hope that the observations of AIRFlag provide insights.

A. Scope and Contributions

The contribution is an end-to-end prototype. It has four elements. First, it includes multi-sensor acquisition and a versioned ESP32 JSON packet. Second, there is a contract validator and a deterministic, explainable risk score. Third, the system logs state-transition events and provides a dashboard for telemetry review. Fourth, it features an LLM explanation tool and a synthetic benchmark facility. A further contribution of this paper is an evidence-bounded evaluation. This evaluation separates the live telemetry session from LLM benchmark inputs. It also separates features that remain unverified.

The design does not interface with aircraft controls. It does not connect to avionics, flight management systems, or certified warning systems. All thresholds are application-specific demonstration values. The ESP32 and UNO display functions are for laboratory use only.

II. RELATED WORK

Flight-data anomaly detection has been approached using clustering, kernel methods, supervised classifiers, and explanatory analytics. Li et al. compared cluster-based abnormal-flight detection with multiple-kernel anomaly detection and exceedance-based screening on B777 and A320 datasets [5]. Coelho e Silva and Murça proposed an analytics framework for flight operations. This framework includes anomaly detection and explanatory models across airline and air-traffic-management use cases [6]. These approaches use richer operational data than the short sensor session evaluated here.

AIRFlag does not implement an anomaly detector. It does not use a trained classifier. Its main risk output is a threshold score. The local LLM is used to produce structured hypotheses. This separation is intentional. Deterministic code retains control of the monitoring state. The local LLM is not permitted to act as a safety authority. The design is closer to a rule-based laboratory monitoring demonstrator than to flight-data anomaly detection research.

III. SYSTEM ARCHITECTURE

The provided ESP32 sketch sets up two communication paths. One path uses USB serial to send a JSON object per line at 115200 baud. This stream is intended for the laptop-side ingestion utility. The second path uses UART2 with GPIO17 as the transmit pin operating at 9600 baud. This stream sends comma-separated packets designed for the Arduino UNO TFT endpoint. The two data streams—USB and display—are kept separate. The laptop gateway does not receive the UNO display packet as part of its telemetry record.

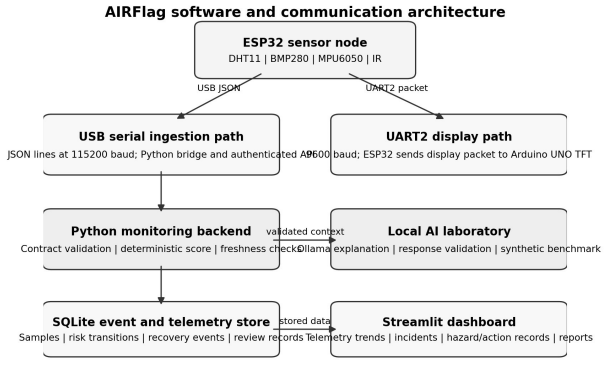


Fig. 1. AIRFlag architecture derived from the supplied firmware and Python project modules.

The USB JSON object identifies the node as ESP32_ENV. It includes the following fields: schema_version, sequence, timestamp, timestamp_ms, sensor_status, fourteen numerical values, IR status, data_valid, alert flags, acknowledgement state, and a time-synchronization flag. The Python ESP32 contract checks the structure of the packet. It ensures all expected fields are present, normalizes the data, keeps the raw payload metadata, and adds a timestamp at the host gateway. The authenticated API

accepts the packet and recalculates the risk score on the server. It does not trust the risk score sent by the device.

A. Hardware and Sensor Roles

The DHT11 sensor provides temperature and relative humidity readings. The BMP280 sensor gives barometric pressure and its own internal temperature. The firmware estimates altitude by calling readAltitude(1013.25), which uses a fixed reference sea-level pressure of 1013.25 hPa. This means the altitude is pressure-based, not derived from GPS. The MPU6050 supplies three-axis acceleration and angular velocity along with a temperature reading. The firmware calculates roll and pitch from the accelerometer data. The IR module returns an input indicating the presence of an obstacle. Details about the sensors' capabilities and interfaces can be found in their respective manufacturer specifications [1]-[4].

TABLE I. HARDWARE AND SOFTWARE ELEMENTS

Element	Implemented role
ESP32	Sensor acquisition; local LEDs, buzzer and acknowledgement button; USB JSON and UART2 outputs
DHT11	Ambient temperature and relative humidity
BMP280	Pressure, sensor temperature and pressure-derived altitude
MPU6050	Acceleration, gyroscope, internal temperature, accelerometer-derived roll/pitch
IR module	Digital obstacle-detection input
Laptop Python application	Packet validation, risk scoring, storage, event generation and API
Ollama local runtime	Optional structured diagnostic explanation and benchmark inference [7]
Streamlit / SQLite	Monitoring dashboard and persistent telemetry/event records

IV. EMBEDDED FIRMWARE AND DATA PROCESSING

A. Sampling and Transmission

The firmware constants define the following sampling and transmission rates: DHT11 every 2500 ms, BMP280 every 2000 ms, MPU6050 every 100 ms, JSON transmission every 2000 ms, and UNO UART packet transmission every 500 ms. The freshness windows are set to 7500 ms for DHT11, 6000 ms for BMP280, and 1000 ms for MPU6050. In the session with 450 records, the average interval between gateway records is 2.0001 seconds, which matches the configured two-second USB JSON transmission. The CSV file does not track each sensor-bus transaction separately.

The firmware includes a time_synced field in its JSON packets. In the session of 450 records, time_synced is reported as true in all 450 client payloads. In the 46-record session, the field shows TIME_NOT_SYNCED and time_synced as false in every record. The timestamps when the gateway receives the data are consistently used for calculating session duration and sampling intervals. The synchronized flag is a device-reported value and does not independently confirm the accuracy of the device clock.

B. Acceleration-Derived Vibration Proxy

The Python telemetry adapter calculates a vibration proxy from the acceleration vector using the formula:

$$v = | \, sqrt(ax^2 + ay^2 + az^2) - 1 \, | \; g \qquad (1)$$

This value measures how much the magnitude of the acceleration vector deviates from one gravity unit. It is not a calibrated RMS vibration value. It does not provide a spectral analysis of vibration and does not serve as an indicator of aircraft structural health. The paper uses the term vibration_g but treats it as an approximate proxy for the acceleration vector.

C. Validation and Timestamp Handling

For the ESP32_ENV backend, the system requires finite and within-range values for ambient temperature, humidity, BMP280 temperature, pressure, altitude, MPU6050 temperature, all three acceleration axes, all three gyroscope axes, roll, and pitch. It also checks four sensor-status fields and the IR Boolean. If any of these inputs are missing, non-finite, out of range, or unhealthy, the data-validity path fails closed, and the effective state transitions to MONITORING DEGRADED. A sensor status of OK reflects the firmware's internal status report. It does not guarantee calibration accuracy or physical health of the sensor.

V. DETERMINISTIC RISK AND LOCAL ALERT LOGIC

A. Backend Demonstration Score

The ESP32_ENV backend branch gives points for threshold exceedances. Let T stand for the DHT11 temperature, RH for the relative humidity, v for the acceleration-vector proxy, theta equal to the maximum of the absolute values of roll and pitch, and IR for the binary obstacle state. The raw score is calculated as:

$$Sraw = 40I(T>=45) + 15I(40<=T<45) + 10I(RH>=85) + 35I(v>=1.8) + 20I(1.0<=v<1.8) + 8I(0.6<=v<1.0) + 35I(theta>=45) + 20I(30<=theta<45) + 45I(IR=1), \quad S=min(100,Sraw) \qquad (2)$$

I(.) represents an indicator function. In the implementation, each tier within a variable does not overlap. If there are validation faults, the backend reports MONITORING DEGRADED rather than treating missing data as a low-risk measurement. Pressure and altitude are checked for validity but do not contribute directly to the score in this ESP32 scoring branch.

TABLE II. ESP32 BACKEND SCORE CONTRIBUTIONS

Input	Condition	Points
DHT11 temperature	T >= 45 C / 40 C <= T < 45 C	+40 / +15
Humidity	RH >= 85%	+10
Vibration proxy	v >= 1.8 / 1.0-<1.8 / 0.6-<1.0 g	+35 / +20 / +8
Tilt	theta >= 45 / 30-<45 deg	+35 / +20
IR	Obstacle detected	+45

The score bands are defined as follows: 0-19: LOW, 20-39: GUARDED, 40-69: HIGH, and 70-100: CRITICAL. These score bands are configuration constants used in the prototype. They are not regulatory limits, aircraft operating limits, or validated hazard thresholds.

B. ESP32 Local Indicator Logic

ESP32 local indicator logic is implemented separately in the firmware. IR detection adds directly to activeAlert. A DHT11 temperature warning is set when the temperature stays at or above forty degrees Celsius for five seconds. The critical temperature flag turns on when the warning is active and the temperature reaches forty-five degrees Celsius. The temperature alert is cleared when the temperature drops below thirty-eight degrees Celsius. Tilt alerting occurs when either roll or pitch reaches thirty degrees or more for one second; tilt is cleared when both angles fall below twenty-five degrees. A debounced physical button can mark the alert as acknowledged while the alert is active. The buzzer sounds while an alert remains active and unacknowledged; the red LED also signals sensor degradation. The authors note a beep when the temperature goes past forty degrees Celsius during bench tests. However, neither of the provided CSV sessions shows a temperature above forty degrees Celsius, and the main session reports temperature_alert = temperature_critical = false in all 450 packets. Thus, the temperature-triggered beep is an observation from the bench, not a quantitative result in the supplied telemetry data.

The local device logic differs from the Python backend. For instance, a steady reading between forty and forty-five degrees Celsius can trigger the ESP32 warning, whereas the backend adds fifteen points and stays in the LOW band if no other term appears. A thirty-degree tilt can trigger the alert, but the backend rates its twenty-point contribution as GUARDED. In the session, two large-pitch samples give backend scores of twenty and thirty-five, both marked GUARDED; the firmware reports the tilt_alert flag as true for the second sample. Nine infrared-positive samples receive a rating of forty-five/HIGH and have local_alert_active = true. These data show some agreement between the backend paths, yet each maintains its own logic and status fields.

C. Event Logging Is Not Timed Notification Escalation

The Python database logs event changes into HIGH, CRITICAL, or MONITORING DEGRADED and logs a

recovery change when the state leaves this configured alert-state set. Repeated samples in an alert state do not create a new event for each sample. The application also stores incident-investigation records and follow-up actions. However, the supplied Python code does not implement a 15-second pilot acknowledgement timer, a 30-second cabin-crew timer, or an ATC/ground-support notification channel. Device-side button acknowledgement is buzzer muting and is not a pilot notification workflow. The current event-state set also excludes GUARDED, so a change from HIGH to GUARDED may be logged as a recovery even though the state is not LOW.

VI. EXPERIMENTAL METHOD

A. Telemetry Sessions

The primary dataset is the supplied filtered ESP32 telemetry CSV. It contains 450 ESP32_ENV records from API:SERIAL:ESP32 with sequence numbers 7186-7635. Gateway timestamps span 2026-11-01 19:28:33.475 to 19:43:31.538 UTC, a duration of 898.063 s (14.97 min). The 449 gaps between records have an average of 2.0001 s, a standard deviation of 0.0183 s, a minimum of 1.950 s, and a maximum of 2.048 s. These are observations from one session, not 450 independent trials.

All 450 primary-session rows have data_valid=1 and all four reported sensor statuses (dht11, bmp280, mpu6050, ir) are OK. The client time-synchronization flag is true in every primary-session packet. Temperature_alert and temperature_critical are false in all 450 packets; measured DHT11 temperature is 28.2-28.6 °C. The file contains no reference measurements, calibration records, controlled tilt fixture, temperature-chamber protocol, or independent ground-truth labels. An earlier filtered CSV is retained as a separate secondary session: 46 records over 90.001 s, mean interval 2.0000 s (SD 0.0192 s), with 43 LOW records and three IR-associated HIGH records at score 45. Its time_synced and alert_acknowledged flags are false in all 46 records. The two CSV sessions are not treated as one continuous experiment.

B. Local LLM Benchmark

A second evidence set was recovered from the project SQLite database. It contains three saved benchmark runs. Each run uses six cases (E01-E06) and three trials per case for each model. The pooled result is 54 trials per model and 162 trials overall. The cases represent tilt, sensor failure, stale telemetry, high temperature, multiple simultaneous hazards, and a separate nominal-condition snapshot. E01 and E06 are isolated readings rather than a time-ordered hysteresis/recovery sequence. Expected scores and freshness labels are derived from the application risk and validation functions.

The benchmark used version v1.0-asram-lab and the saved generation options temperature=0.1 and seed=42 with JSON output mode. The evaluated models were gemma3:4b, llama3.2:latest, and qwen2.5:7b. This benchmark does not include qwen3:8b, which was installed later in the development environment. The reported development workstation uses an Intel Core i7-13700HX, an NVIDIA RTX 4060 Laptop GPU with 8 GB VRAM, and 16 GB system memory. The benchmark records do not preserve an environment manifest for each run, so latency remains machine- and configuration-dependent. The Ollama local API supports JSON-format responses and reproducible generation options [7].

VII. RESULTS AND DISCUSSION

A. Recorded Hardware Telemetry

TABLE III. SUMMARY OF THE PRIMARY 450-RECORD TELEMETRY SESSION

Measure	Observed value
Records / duration	450 / 898.063 s (14.97 min)
Mean interval	2.0001 s (SD 0.0183 s)
Validity / sensor status	450/450 valid; all four statuses OK
Risk-state counts	439 LOW; 2 GUARDED; 9 HIGH; 0 CRITICAL
Tilt-associated samples	2; scores 20 and 35, both GUARDED
IR detections / acknowledgement	9 HIGH records; ack flag true in 7/9
Temperature alert flags	0/450; DHT11 range 28.2-28.6 °C

TABLE IV. SENSOR AND MOTION RANGES IN THE PRIMARY SESSION

Variable	Min to max	Mean
Ambient temperature (°C)	28.20 to 28.60	28.38
Relative humidity (%)	58.90 to 60.20	59.29
Pressure (hPa)	918.58 to 918.74	918.67
Pressure-derived altitude (m)	818.33 to 819.84	819.01
Acceleration-vector proxy (g)	0.015 to 0.103	0.029
Roll (deg)	-3.27 to 10.41	-2.06
Pitch (deg)	-45.32 to 5.20	0.38

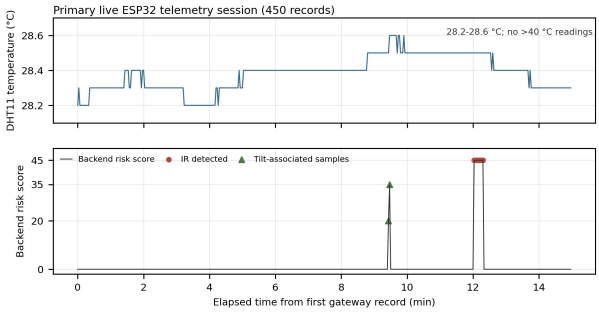


Fig. 2. DHT11 temperature and deterministic backend risk score in the primary 450-record session; the nine IR detections correspond to the nine HIGH-score records.

In the session, 439 records (97.6%) were LOW with score 0. Two consecutive records, sequences 7469 and 7470, recorded pitch values of -42.64 and -45.32 degrees and backend scores of 20 and 35 (GUARDED). The firmware-reported tilt_alert flag was true in sequence 7470. Nine consecutive records, sequences 7547-7555, had ir_detected=1 and a score of 45/HIGH, with local_alert_active=true. At sequence 7556, IR returned to 0. The backend score returned to 0/LOW. These observations show the recorded backend response to pitch and IR-positive inputs in this session. They do not establish calibrated orientation or independent obstacle-detection accuracy.

TABLE V. THE THREE HIGH-RISK RECORDS IN THE EARLIER 46-RECORD SESSION

Seq.	Gateway time (UTC)	IR	Score/state
41	12:17:37.605	1	45 / HIGH
42	12:17:39.596	1	45 / HIGH
43	12:17:41.603	1	45 / HIGH

The primary session reports alert_acknowledged=true in seven of the nine IR-active packets; the earlier 46-record session reports false in all records. This is a firmware-reported state flag, not a measurement of button contact, buzzer output, or LED output. Neither dataset contains a pilot response, cabin-crew notification, ATC notification, or timed escalation record. The reported 40 °C temperature beep is not visible in these CSVs because the

recorded ambient temperature never exceeds 28.9 °C across the two sessions.

B. Saved Synthetic LLM Benchmark

TABLE VI. POOLED RESULTS FROM THREE SAVED SYNTHETIC BENCHMARK RUNS

Model	Success	JSON	Validator	Kwd. check	Mean latency
gemma3:4b	54/54	54/54	54/54	45/54	6.76 s
llama3.2:latest	54/54	54/54	54/54	45/54	4.43 s
qwen2.5:7b	54/54	54/54	54/54	54/54	8.09 s

All 162 benchmark executions were recorded as successful model calls. Each model produced valid JSON in 54/54 trials and passed the configured safety validator in 54/54 trials. The scenario keyword check was 45/54 (83.3%) for Gemma 3 4B and Llama 3.2, and 54/54 (100%) for Qwen 2.5 7B. The restricted-term/evidence-grounding filter passed 54/54 for each model. Mean recorded inference latencies were 6.76 seconds, 4.43 seconds, and 8.09 seconds, respectively.

These values are not model accuracy. The technical-correctness evaluator primarily checks for the presence of scenario-related words in the summary and technical-interpretation fields; it does not require agreement with all expected sensor values, thresholds, score, or state. The grounding filter screens for a set of sensor-role confusions and disallowed/fabricated terms, not every unsupported statement. All 162 unsupported_claims fields remain unevaluated (NULL). Human review status is 160 Pending, one Approved, and one Rejected. The results therefore describe execution, JSON conformance, and configured rule checks only; they do not establish hallucination-free output or clinical/aviation diagnostic accuracy.

C. Interpretation

The main 450-record session shows a demonstration: all packets passed the data-validity check, and all four sensor-status fields were OK. Two large-pitch samples produced GUARDED backend scores, and nine IR-positive samples produced HIGH scores. The earlier 46-record session separately contains three HIGH records. Neither CSV contains a temperature-threshold crossing; the beep that the authors observed when the temperature rose is still qualitative. Neither dataset has reference measurements or calibration evidence. The synthetic LLM benchmark is separate from both hardware sessions, and its heuristic evaluation does not prove accuracy.

VIII. SOFTWARE VERIFICATION AND IMPLEMENTATION STATUS

The project includes automated tests for telemetry normalization, risk assessment, API ingestion, event persistence, duplicate-event suppression, recovery-event logging, benchmark execution, AI-response validation, database migration, and dashboard components. The project's saved integration tests specifically verify that a repeated same-state alert does not create one event per sample and that a transition from a high-risk input to a low-risk input produces both RISK_ALERT and RECOVERY records. The project's software tests use synthetic packets and temporary SQLite databases; they are not independent trials on the physical ESP32 sensors.

The project also provides a hazard register, an action tracker, incident-investigation records, telemetry replay, and reporting/export functions. The supplied project database contains 1,523 event records, one saved AI explanation, and no saved safety-action or incident-review rows. The project's counts describe the database snapshot. They are not evidence that every dashboard workflow was exercised during the hardware session.

The supplied ESP32_M13_JSON_USB_TFT_UART.ino sketch defines USB JSON transmission and a separate UART2 display packet. The project ZIP also contains a Milestone 13 README describing the USB JSON bridge as pending and a MicroPython file explicitly retained as an archived template. The README and supplied sketch therefore represent different revision states. The CSV packet structure is consistent with the JSON contract. The exact firmware binary loaded during capture cannot be independently established from the supplied files. The sketch itself states that it has not been compiled against the board package and library versions. The project does not supply an UNO TFT receiver sketch or UNO-side test log.

IX. LIMITATIONS AND THREATS TO VALIDITY

The hardware evidence consists of two consecutive telemetry sessions: 450 records over 898.063 s and an earlier 46 records over 90.001 s. The project's evidence does not include repeated trials. The project does not provide a reference instrument, sensor calibration certificate, controlled tilt fixture, temperature chamber, humidity reference, or documented IR stimulus protocol. The project's primary session reports synchronized client time, whereas the earlier session does not. The project's sessions do not record temperature above 40 °C, so the temperature-triggered buzzer observation reported by the authors is not quantitatively represented. Empty GPS, distance, and light columns must remain unreported as measurements. The project's sensor status of OK reflects firmware/contract reporting and does not prove sensor accuracy or wiring integrity.

The project's pressure-derived altitude uses a fixed sea-level pressure reference, and the project's dataset does not establish altitude accuracy. The project's roll and pitch are computed from accelerometer readings without a documented calibration or sensor-fusion validation procedure. The project's acceleration-vector deviation is not a calibrated vibration measurement. The project's local ESP32 alert thresholds differ from the project's Python scoring thresholds, which can produce different local and dashboard states for the same input.

The project's event logger is not an escalation service. The project's physical acknowledgement button silences the buzzer while an alert is active, but no timed recipient escalation is implemented in the project's supplied Python backend. The project does not contain pilot/cabin/ATC delivery records. The project's LLM benchmark uses six cases, a single prompt version, and a heuristic evaluator; the project's saved data do not assess unsupported claims. Finally, the project is a laboratory prototype and must not be used as an aircraft alerting, navigation, or flight-decision system.

X. CONCLUSION AND FUTURE SCOPE

This paper presented Operation AIRFlag as an ESP32-based educational monitoring prototype. It combines data from the DHT11, BMP280, MPU6050, and IR sensors. The backend uses Python for validation and deterministic scoring. Data is stored in SQLite, and a Streamlit dashboard provides visualization. Optional local LLM diagnostics were also included. In the session with 450 records, all samples passed validation. All four sensor-status fields showed OK. Of these, 439 records were LOW, two samples with pitch were GUARDED, and nine IR-positive samples were HIGH with a score of 45. A separate earlier session of 46 records had 43 LOW and three IR-associated HIGH records. The primary session also included seven IR-active packets with firmware-reported acknowledgement flags. No temperature exceeded 40 °C in either CSV file. As a result, the buzzer behavior triggered by temperature remains a qualitative observation from bench testing. The separate benchmark with 162 trials showed successful local model execution and JSON/safety-validator acceptance. However, the keyword-based scenario metric does not reflect model accuracy.

Future work should first align device timestamps and include firmware and build identifiers with each dataset. Controlled experiments should use reference instruments and consistent, pre-defined conditions for each sensor and threshold. The ESP32 firmware and backend alert rules need to be reconciled and tested against the boundary cases. The UNO receiver firmware and the physical display path must be verified. A real-time escalation system would require state tracking, acknowledgment, recipient details, timeout settings, retry logic, and delivery records. These must be validated with end-to-end measurements. The LLM benchmark should use expected-value comparisons, include independent human review of claims, expand the number of test

cases, and document the full hardware and software environment. Until these steps are completed, AIRFlag remains a proof-of-concept monitoring and explanation platform. It is not yet a validated aviation safety system.

REFERENCES

[1] Espressif Systems, “ESP32 Series Datasheet,” Version 5.3, Espressif Systems. [Online]. Available: https://documentation.espressif.com/esp32_datasheet_en.html.

[2] Bosch Sensortec, “BMP280 Data Sheet,” document BST-BMP280-DS001-26, rev. 1.26, Oct. 2021. [Online]. Available: https://www.bosch-sensortec.com/media/boschsensortec/downloads/datasheets/bst-bmp280-ds001.pdf.

[3] InvenSense Inc., “MPU-6000 and MPU-6050 Product Specification,” document PS-MPU-6000A-00, rev. 3.4, Aug. 19, 2013. [Online]. Available: https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-6000-Datasheet.pdf.

[4] Guangzhou Aosong Electronics Co., Ltd., “DHT11 Product Manual,” version V1.3, Mar. 31, 2017. [Online]. Available: https://aosong.com/userfiles/files/media/DHT11%E6%95%B0%E5%AD%97%E6%B8%A9%E6%B9%BF%E5%BA%A6%E4%BC%A0%E6%84%9F%E5%99%A8%E6%A8%A1%E5%9D%97.pdf.

[5] L. Li, S. Das, R. J. Hansman, R. Palacios, and A. N. Srivastava, “Analysis of flight data using clustering techniques for detecting abnormal operations,” *Journal of Aerospace Information Systems*, vol. 12, no. 9, pp. 587–598, 2015, doi: 10.2514/1.I010329.

[6] L. Coelho e Silva and M. C. R. Murça, “A data analytics framework for anomaly detection in flight operations,” *Journal of Air Transport Management*, vol. 110, Art. no. 102409, 2023, doi: 10.1016/j.jairtraman.2023.102409.

[7] Ollama, “API reference,” Ollama Documentation. [Online]. Available: https://docs.ollama.com/api. Accessed: Nov. 2, 2026.