

Air System Monitoring

Matteo Comini

Academic Year 2024/2025



[GitHub Repository](#)

Contents

1 Introduction

Indoor air quality is critical for human health, considering that people spend approximately 90% of their time indoors. Numerous studies highlight how indoor air pollution (IAP) is linked to chronic respiratory diseases, cardiovascular issues, and other health problems significantly impacting occupants' well-being (Tran et al., 2020). Major sources of IAP include emissions from human activities such as cooking, heating, and the use of building materials and household chemicals.

To effectively address poor air quality, the use of Internet of Things (IoT) technologies combined with predictive machine learning models appears promising (Gawade et al., 2020). This project aims to develop an advanced IoT system employing two sensors to monitor key parameters such as temperature, humidity, carbon monoxide, volatile organic compounds, and particulate matter. Specifically, the system integrates two predictive models based on machine learning techniques to forecast temperature variations and pollution levels, using real-time collected data.

The dataset utilized in this project was gathered from continuous 24-hour environmental monitoring conducted using a custom-built system equipped with sensors specifically for CO, overall air quality, temperature, humidity, estimated CO₂ levels, volatile organic compounds, and dust density. This approach allows for timely interventions using targeted actuators, significantly improving indoor comfort and safety, while simultaneously optimizing energy consumption through accurate predictions of occupancy and space utilization.

1.1 Dataset Description

The dataset used in this project originates from a custom-built air quality monitoring system deployed in an indoor environment over a 24-hour period. It includes real-time sensor readings aimed at evaluating various aspects of indoor air quality. The dataset consists of seven distinct fields:

- **MQ7 Value:** Measures Carbon Monoxide (CO) levels in parts per million (ppm) using the MQ7 sensor.
- **MQ135 Value:** Represents a general air quality index derived from the MQ135 sensor, sensitive to gases such as ammonia (NH₃), nitrogen oxides (NO_x), and benzene.
- **Temperature:** Captures ambient temperature in degrees Celsius.
- **Humidity:** Measures relative humidity as a percentage.
- **eCO2:** Provides estimated CO₂ concentration based on data from the CCS811 sensor (ppm).
- **TVOC:** Indicates the concentration of Total Volatile Organic Compounds in parts per billion (ppb), also measured by the CCS811 sensor.
- **Dust Density:** Reports the density of airborne particulate matter, measured in micrograms per cubic meter ($\mu\text{g}/\text{m}^3$) using the GP2Y1010 dust sensor.

This Kaggle dataset provides the essential ground truth for training and validating the predictive models implemented in both air quality classification and temperature forecasting tasks.

2 Application Logic

The Air System Monitoring relies on sensor nodes designed to continuously monitor various environmental conditions within an indoor environment. Each sensor integrates a machine learning model to analyze sampled data and provide predictions or classifications regarding future conditions. These insights are communicated to corresponding actuators, which then execute targeted responses to maintain optimal environmental safety and comfort levels.

The system classifies the environmental conditions into three general risk levels: Low Risk (LR), Moderate Risk (MR), and Critical Risk (CR). Actuators respond differently based on these classifications, activating appropriate measures to manage risks effectively and ensure worker safety.

All collected sensor data, along with predictions, are sent to a central server application and stored in a cloud-based database. Users have the ability to interact directly with the Air System Monitoring system via a user interface, enabling adjustments to sensor thresholds and actuator responses to better suit the specific needs of their environment. This approach ensures proactive management and continuous monitoring of environmental safety.

3 Architecture

The Air System Monitoring architecture comprises sensor nodes and actuators connected within a CoAP network, which communicates through a Border Router to a centralized Java Cloud Application. This cloud application manages data storage within a MySQL database and provides a user interface for direct user interaction and control of the system.

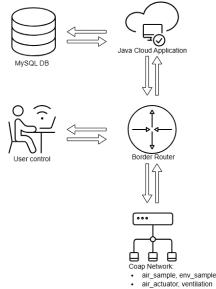


Figure 1: Architecture representation.

3.1 Coap Network

Before becoming operational, all sensors and actuators in the Air System Monitoring enter an initialization phase where a red LED is illuminated to indicate their unregistered status. This visual signal serves as a clear indicator that the device is not yet integrated into the CoAP communication framework. Upon user interaction (specifically, pressing the onboard registration button) the LED

turns off, and the device sends a registration request to the network. This action transitions the node from an inactive to an active state, enabling it to participate in data exchange and respond to system-level events as part of the distributed architecture.

3.1.1 Air Sample Sensor

The ‘air_sample’ sensor node is a critical component of the Air System Monitoring CoAP network, responsible for continuously monitoring air quality parameters, including Carbon Monoxide (CO), total volatile organic compounds (TVOC), and a general air quality index. The sensor periodically samples the environment every 10 seconds, categorizing air quality conditions into risk levels using a machine learning model integrated within the node. The results and classifications are communicated via CoAP to the corresponding actuators through observable resources (‘res-danger’, ‘res-monitoring-air’). The sensor node also supports remote shutdown through the ‘res shutdown’ resource, ensuring controlled management within the network. Notifications and data payloads are structured using JSON format for clear and efficient communication across the system.

3.1.2 Env Sample Sensor

The ‘env sample’ sensor node plays a crucial role within the Air System Monitoring CoAP network by continually monitoring environmental conditions such as temperature, humidity, and particulate matter (dust density). This sensor samples the environment every 2 seconds, maintaining a recent history of samples to facilitate predictions using an integrated machine learning model. The predictive model specifically estimates future temperature values, which are communicated to the actuators through observable CoAP resources (‘res-predict-temp’, ‘res-monitoring’). The system supports remote shutdown functionality via the ‘res-shutdown’ resource, enabling controlled deactivation when necessary. Data communications and notifications utilize structured JSON payloads for clear, standardized interactions within the network.

3.1.3 Air System Actuator

The ‘air system actuator’ (like a purification air system) is a crucial node within the Air System Monitoring CoAP network, primarily responsible for responding to air quality risk levels communicated by the ‘air sample’ sensor. This actuator registers itself to observe the risk notifications sent from the sensor and reacts by triggering corresponding responses based on three defined risk categories: Low Risk, Moderate Risk, and Critical Risk. The actuator visually indicates the current risk level through LEDs (red for higher risks, yellow for moderate risks, and green for safe conditions). Additionally, it supports dynamic adjustment of risk thresholds via the observable resource ‘res-threshold’, and a remote shutdown capability through the ‘res-shutdown’ resource. All interactions and communications within this node are structured and managed through CoAP messages, ensuring efficient and standardized communication across the network.

3.1.4 Ventilation Actuator

The ‘ventilation_actuator’ is an essential node within the Air System Monitoring CoAP network, specifically designed to manage ventilation systems based on temperature predictions from the ‘env_sample’ sensor. The actuator registers

itself to receive continuous temperature notifications and utilizes these predictions to determine the appropriate ventilation response. The ventilation control logic classifies temperature levels into categories, activating different ventilation intensities ranging from off, medium, to maximum based on predefined thresholds. These operational states are visually indicated by LEDs: green for safe conditions (ventilation off), yellow for moderate intensity, and red for maximum risk. The actuator's thresholds can be dynamically adjusted through the observable CoAP resource ‘res threshold’, while it also supports remote shutdown via the ‘res shutdown’ resource. All interactions and notifications occur through structured CoAP communications, ensuring seamless integration and consistent responsiveness within the network.

3.2 Machine Learning Models

3.2.1 Air Quality Classification Model

The ‘air_sample’ node integrates a lightweight classification model designed for embedded execution using the Emlearn framework. The model is implemented as a decision tree and compiled directly into C for real-time prediction. It operates on three environmental features acquired from onboard sensors: carbon monoxide (CO), total volatile organic compounds (TVOC), and a composite air quality index.

At runtime, the model receives a feature vector $\mathbf{x} = [x_1, x_2, x_3]$, where each x_i corresponds respectively to CO, TVOC, and general air quality. The decision tree performs a series of hierarchical threshold evaluations to return a predicted class $y \in \{0, 1, 2\}$, representing the qualitative air risk levels: Safe (0), Warning (1), and Danger (2).

The core function ‘predict_class()’ internally invokes ‘eml.trees_predict()’ on the model structure ‘danger_air_classification’, passing the feature vector. The result is used to trigger actuation logic in ‘air_system_actuator’, which differentiates reactions based on severity: from simple LED alerts in safe or warning states to more aggressive responses in danger situations.

This setup ensures high reactivity with minimal computation, which is critical for constrained embedded systems. The integration with CoAP resource notifications (‘res-danger’) also enables timely propagation of risk assessments across the network.

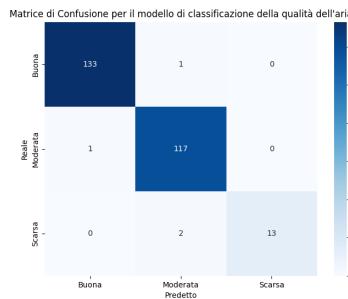


Figure 2: Confusion Matrix Quality Air Prediction.

3.2.2 Temperature Forecasting Model

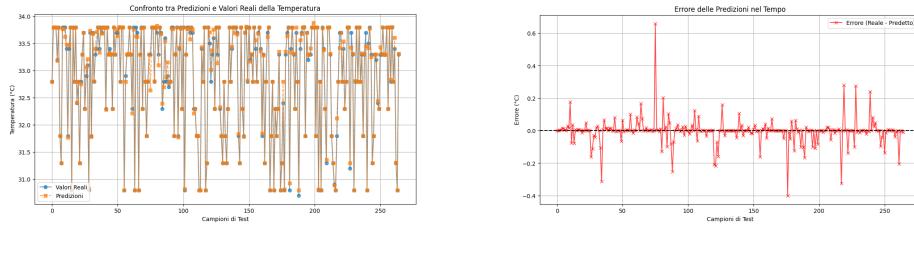
The ‘env_sample’ node implements a regression model to predict short-term temperature trends based on recent sensor history. The model is constructed using

Emlearn’s ‘eml_trees_regress1’ interface and deployed as a compact decision-tree regressor.

The input vector $\mathbf{x} \in R^{30}$ comprises 10 most recent samples of each environmental variable: normalized temperature, relative humidity, and dust concentration. These values are arranged in a fixed-order sequence and normalized within predefined min-max ranges before being passed to the predictor.

The model produces a single floating-point output that represents the predicted ambient temperature at the next sampling instant. This prediction is denormalized and sent to the ‘ventilation_actuator’, which interprets the result in relation to a configurable temperature threshold. The actuation logic defines three ventilation states: off ($T < \tau$), medium ($\tau \leq T < \tau + 2$), and maximum ($T \geq \tau + 2$).

This predictive architecture allows for proactive thermal control, optimizing energy efficiency, and user comfort. The local resource ‘res-threshold’ also enables runtime tuning of the decision boundary τ , enhancing adaptability to varying environmental conditions.



(a) Temperature Prediction

(b) Prediction Error

Figure 4: Temperature model output and error.

3.3 Cloud Application and MySQL Database

3.3.1 Registration Phase

The Cloud Application is responsible for managing the lifecycle of all devices in the system. At startup, the CoAP server is initialized via the ‘Main.java’ and ‘MyServer.java’ classes, which register the CoAP resources ‘/registration/sensor’ and ‘/registration/actuator’ handled by ‘CoapResourceRegistrationSensor’ and ‘CoapResourceRegistrationActuator’, respectively.

1. **Sensor Registration:** During the registration phase, the sensor sends a lightweight JSON payload: the sensor identifier (“s”), the list of sensing features (“ss”), and the sampling period (“t”). Note that actual sensor readings (samples) are not included in this phase. Upon reception, the Cloud Application enriches this data: it extracts the source IPv6 address from the network header and assigns the device type (‘sensor’) based on the endpoint accessed. Finally, it stores the fully resolved triplet (name, ipv6, type) into the ‘ipv6_addresses’ table, effectively mapping the compact CoAP message to the structured database schema.
2. **Actuator Registration:** Once at least one sensor is registered, actuators send a POST request to ‘/registration/actuator’. The server registers the actuator and retrieves all registered sensor IPv6 addresses to forward them to the actuator.

3. **Retry Logic:** If registration fails, the node retries until acknowledgment is received or a maximum retry count is reached.

3.3.2 Monitoring Phase

After successful registration, an observer is attached to each sensor using the ‘CoapObserver’ class:

1. **Observation Setup:** The observer subscribes to observable resources such as ‘/monitoring’ via CoAP’s observe mechanism.
2. **Notification Handling:** Upon receiving a new data notification, the payload is parsed, and relevant values are extracted.
3. **Data Storage:** The extracted values are inserted into the dynamically created sensor-specific tables via ‘IPv6DatabaseManager’, forming a timestamped history of environmental conditions.
4. **Database Schema:** Apart from the shared ‘ipv6Addresses’ table for device info, each sensor has a dedicated table for storing its time-series data.

This architecture allows real-time monitoring, efficient device tracking, and modular data persistence, ensuring clear separation of sensor data and robust integration between edge devices and the cloud.

3.3.3 Database Structure

The database used by the Cloud Application is a MySQL relational database designed to support dynamic registration and time-series monitoring. It is composed of one static table and a variable number of dynamically generated tables, created at runtime during the registration phase.

1. ipv6Addresses Table This is the central registry table where all registered devices are stored. Each entry includes:

- **id** (INT, PRIMARY KEY, AUTO_INCREMENT): Unique identifier for the device.
- **name** (VARCHAR): Device name as declared in the JSON payload (e.g., `air_sample`).
- **ipv6** (VARCHAR): Full IPv6 address of the node.
- **type** (VARCHAR): Device type (`sensor` or `actuator`).

This table is populated by both `CoapResourceRegistrationSensor` and `CoapResourceRegistrationActuator`, and accessed by `databaseHelper` in the user interface for lookup operations.

2. Dynamically Created Sensor Tables For each sensor that registers, a dedicated table is created to store its observations. The name of the table is constructed by concatenating the device name and a sanitized version of its IPv6 address (e.g., `sensor_air_sample_aaaa_212_4b00_abcd_1234`).

Each of these tables has the following structure:

- **timestamp** (DATETIME): Time of the received observation.
- **value1, value2, ...**: Sensor-specific data fields depending on the resource being monitored (e.g., temperature, humidity, dust, risk level).

The creation and population of these tables is handled in the `IPv6DatabaseManager` class. It includes methods to:

- Check if a table exists.
- Generate the table schema dynamically upon sensor registration.
- Insert values as new CoAP observations arrive via `CoapObserver`.

3. Entity-Relationship Overview At the ER level, the schema can be conceptualized as follows:

- **One-to-One**: Between each row in `ipv6Addresses` and the sensor-specific table created for it.
- **One-to-Many**: Each device (row in `ipv6Addresses`) may be related to many timestamped measurements in its corresponding time-series table.

This design ensures that all device metadata and historical data are organized and accessible for analysis, remote control, and audit purposes, while maintaining modularity and scalability.

3.4 User Control

The Air System Monitoring provides a dedicated Java-based remote control interface, implemented in the ‘`RemoteControlApp`’ class. This console application allows users to interact in real time with the IoT nodes deployed in the network by issuing CoAP commands and reading live data directly from the MySQL database.

The remote interface, supported by the ‘`databaseHelper`’ class, first loads all devices present in the ‘`ipv6Addresses`’ table—both sensors and actuators—allowing users to interact with devices based on their current registration status. The user is then presented with a menu of available control functions:

- 1 **Threshold Adjustment**: The user can set a custom temperature threshold for both the ‘`ventilation_actuator`’ and the ‘`air_system_actuator`’. Once a threshold is selected, the application sends a CoAP POST request with the threshold value as plain text to the ‘/threshold’ resource of each actuator.
- 2 **Device Status Check**: The application allows the user to verify if a specific device (either a sensor or an actuator) is active. This is done by scanning the current IPv6 table and comparing it with the user-provided device name.
- 3 **Danger Counter Monitoring**: For the ‘`air_sample`’ node, users can monitor how many times the node has predicted a ”Danger” class. This is done by sending a CoAP GET request to the ‘/dangerCount’ resource of the node. The response is parsed and printed in the console, providing insight into the system’s recent air quality assessments.
- 4 **Remote Shutdown**: The user can remotely shut down any device—either sensor or actuator—by sending a CoAP GET request to the ‘/shutdown’ resource of the selected node. Upon successful shutdown, the device’s entry is also removed from the ‘`ipv6Addresses`’ table to maintain consistency. Each operation is validated through dynamic lookups of the IPv6 database, ensuring that commands are only sent to devices that are currently online and correctly registered.

3.5 Data Encapsulation and Representation

The Air System Monitoring uses JSON-formatted payloads for structured data exchange between sensor nodes and the cloud application. This format is particularly employed in the registration and monitoring phases, and it is implemented on embedded nodes using the lightweight cJSON library.

Registration Phase: Both sensor nodes (air_sample and env_sample) encapsulate their identity and sensing capabilities in a JSON object. The payload is constructed with fields:

- **s**: a string indicating the sensor name (e.g., air_sample).
- **ss**: a string array listing the sensor features (e.g., co, tvoc, air).
- **t**: an integer representing the sampling period.

This structure is created via cJSON_CreateObject(), serialized with cJSON_PrintUnformatted(), and transmitted using a CoAP POST request to the registration endpoint.

Monitoring Phase: Each sensor constructs JSON payloads for the monitoring CoAP resources using real-time sampled data:

- air_sample transmits co, air_quality, tvoc, and a predicted danger level.
- env_sample transmits temperature, humidity, and dust density.

In both cases, the payload includes:

- **ss**: a numeric array containing the latest sensed values (and predicted class, if applicable).
- **t**: an integer sample ID or timestamp.

The JSON is constructed using cJSON_AddItemToArray() and cJSON_AddNumberToObject() functions and served directly via GET or notification events using Contiki-NG's CoAP engine.

Actuator Parsing: The actuators employ a hybrid parsing strategy to optimize efficiency. During the registration phase, they utilize the cJSON library to parse the response received from the Cloud Application, specifically extracting the target sensor's IPv6 address from the key "e" (Endpoint). Conversely, during the monitoring phase (Machine-to-Machine communication), the data exchange is streamlined to minimize overhead: sensors transmit predictions and values as plain text strings (e.g., "3050" for temperature or "2" for danger levels). Consequently, the actuator processes these incoming payloads directly using standard string-to-integer conversion functions (e.g., atoi), bypassing the computational overhead of JSON parsing for the real-time control loop.

References

- [1] Tran, V. D., Park, D., Lee, Y., Lee, Y. C. (2020). Indoor air pollution, related human diseases, and recent trends in the control and improvement of indoor air quality. *International Journal of Environmental Research and Public Health*, 17(8), 2927.
- [2] Gawade, S. P., Bhagat, R. V., Karandikar, P. (2020). Indoor Air Quality Improvement. *arXiv preprint arXiv:2012.15387*.