CHAITANYA BHARATHI INSTITUTE OF TECHNOLOGY (A)
Kokapet (Village), Gandipet, Hyderabad, Telangana-500075. www.cbit.ac.in

Title: Edge-Driven CO2 and Occupancy Prediction with Certificate Generation for Optimized Classroom Ventilation Systems

CBIT/IT/2024-25/IT1/104

Presented By:

C. BHAVITHA-160121737005
G. SRI HARSHA -160121737033
V. S. RAGHU VARDHAN-160121737061

Our Guide:

Mrs. E. Ramalakshmi
Assistant Professor, Dept of IT.

Department of Information Technology
Chaitanya Bharathi Institute of Technology

Overview

Abstract

As indoor ventilation quality becomes increasingly critical in educational settings, optimizing classroom ventilation systems is essential for enhancing the well-being of students and teachers. This study presents an innovative approach leveraging edge computing to develop a real-time prediction system for carbon dioxide (CO2) levels and classroom occupancy, accompanied by a certificate generation mechanism to certify the building's ventilation performance. By deploying low-cost IoT devices equipped with CO2 sensors and our solution collects environmental data at the classroom level. A lightweight Long Short-Term Memory (LSTM) model is implemented on these edge devices to provide accurate forecasts of CO2 concentrations and occupancy rates, allowing for immediate ventilation adjustments. The system architecture features a hierarchical design that includes local edge devices for data acquisition and initial predictions, an intermediate fog layer utilizing Raspberry Pi for aggregation and building-wide decision-making for long-term data storage and analysis. Our methodology also incorporates dataset augmentation techniques to adapt existing air quality datasets to classroom environments, ensuring robust model training and generalization. Additionally, a certificate is dynamically generated based on the building's ventilation performance using a metric called KPIv, providing a transparent and accessible evaluation of ventilation quality over time.

Key words

Edge Computing, CO2 Prediction, Occupancy Estimation, Classroom Ventilation, IoT Devices, LSTM Model, Smart Educational Environments

INTRODUCTION

- Indoor air quality has become increasingly important in educational settings, as proper ventilation directly impacts the health and productivity of students and teachers.

- The project focuses on developing a smart classroom ventilation system using IoT devices and edge computing, with real-time predictions of CO_2 levels and occupancy through lightweight LSTM models.

- By collecting and analyzing environmental data at the classroom level, the system enables immediate ventilation adjustments and generates dynamic certificates based on the KPIv metric to evaluate long-term ventilation performance.

OBJECTIVES

- Importance of maintaining healthy indoor air quality in educational environments.
- Real-time prediction of CO_2 levels and classroom occupancy using edge-deployed LSTM models.
- Implementation of a dynamic certification system to evaluate ventilation performance using KPIv.
- Scalable architecture integrating IoT devices, edge computing, and fog nodes for efficient data processing.
- Adaptation of existing air quality datasets through augmentation for improved model training and generalization.

Year	Author Name	Title	Institute	Strengths
2021	Shaohua Yang, Zeqiong Huang, Cong Wang, Xu Ran, Changhao Feng, Bin Chen	A real-time occupancy detection system for unoccupied, normally and abnormally occupied situation discrimination via sensor array and cloud platform in indoor environment	Elseiver	• Integrated multiple sensors have been proposed to improve the detection reliability. *Sensor array fusion* is a technique that combines data collected from multiple sensors to provide enhanced performance over any single sensor • The voting based weighted extreme learning machine (WV-ELM) model achieved the highest prediction accuracy and the combination of light and CO2 sensors.

Research Gaps

The layout and actual environmental parameters of different indoor environments may be different. Whenever the system deployed to a new environment, it is necessary to train algorithm again to obtain for optimization.

Year	Author Name	Title	Institute	Strengths
2024	Nyoman Kusuma Wardana, Suhaib A. Fahmy, Julian W. Gardner	Low-cost SCADA/HMI with Tiny Machine Learning for Monitoring Indoor CO2 Concentration	IEEE	• This paper demonstrated integration of tiny machine learning for microcontrollers with low-cost SCADA/HMI to monitor and predict indoor CO2 concentrations. • The trained predictors produce RMSE ranging from approximately 0.5–7 ppm when predicting future CO2 concentrations 1, 15, and 30 minutes ahead

Research Gaps

- Due to resource constraints in the microcontrollers used, a simple linear regression model was chosen as the machine learning algorithm.

Year	Author Name	Title	Journal	Strengths
2024	Francisco Maciá-Pérez , Senior Member, IEEE, Iren Lorenzo-Fonseca , and José-Vicente Berná-Martínez	An AI-Based Ventilation KPI Using Embedded IoT Devices	IEEE	• The project implements two regressive neural network models to estimate the number of occupants in an enclosed space and the CO2 levels. • These models are embedded in IoT devices that monitor real-time ventilation conditions and ensure optimal indoor air quality (IAQ).It uses a ventilation Key Performance Indicator (KPIv) that calculates a ratio between the estimated number of occupants based on Wi-Fi connections and CO2 concentrations.

Research Gaps

- Use of ANN models which fail to capture the temporal data and time-based nature of the data.
- Server heavy computation which doesn’t make the system real-time

Year	Author Name	Title	Journal	Strengths
2022	Lorenzo Monti Rita Tse Su-Kit Tang Silvia Mirri Giovanni Delnevo Vittorio Maniezzo Paola Salomoni	Edge-Based Transfer Learning for Classroom Occupancy Detection in a Smart Campus Context	MDPI	• An edge-based system for classroom occupancy detection was developed using Intel RealSense D415 cameras. A simplified version of YOLOv3 was fine-tuned through transfer learning with images from small and large classrooms. This approach enabled a shift from a traditional client-server model to a fat client–thin server architecture, performing predictions directly at the edge.

Tools and Technologies

Hardware:

- **IoT Devices:** Arduino (connected via USB Serial to the Raspberry Pi, using ports like /dev/ttyS0, /dev/ttyS1).
- **Sensors:**
- DHT11 (Temperature/Humidity, connected to Arduino A1).
- Analog CO2 Sensor (connected to Arduino A0).
- IR Beam Sensors (Occupancy counting, connected to Arduino digital pins).
- Fog Layer: Raspberry Pi (Acts as the central hub running the main Python logic).

Tools and Technologies

Models:

- Type: LSTM models built and run on the Raspberry Pi using TensorFlow/Keras.
- Training: Involves two phases:

1. Pre-training on a general, merged dataset (**IndoorVentilationModel.py**).
2. Fine-tuning on specific classroom data collected live (lstm.py), potentially leveraging pre-trained weights.

- Preprocessing: Includes MinMaxScaler for normalization and a custom DataHarmonizer for outlier handling and adding context (like hour).

Software:

- **Cloud**: ThingSpeak for cloud data uploads and visualization.
- **ML Framework:** TensorFlow/Keras (standard version, running on Raspberry Pi).
- **Languages**: Python (for Raspberry Pi logic), C/C++ (for Arduino firmware).
- **Key Libraries**: pyserial (serial communication), numpy (numerical operations), tensorflow (model building/training), scikit-learn (MinMaxScaler), RPi.GPIO & RPLCD (LCD control).

WORKING

1. Data Source (Arduino):

- Read Temp/Humidity (DHT11), CO2 (Analog), Occupancy (IR beams).
- Calculate KPIv based on CO2/Occupancy on-device.
- Format data (temp, hum, co2, counts, kpiv, trend) with delimiters
- Transmit formatted string via Serial periodically.

2. Pre-training (Offline - IndoorVentilationModel.py):

- Load & preprocess merged historical dataset (select columns, handle missing values, add 'hour' feature).
- Normalize 5 features (co2, temp, hum, occ, hour) with MinMaxScaler; Save scaler.
- Create sequences (10 input steps -> 1 output step).
- Build & train LSTM (5 features) with EarlyStopping.
- Save best pre-trained model weights.

3. System Initialization (RPi - lstm.py):

- Initialize ClassroomMonitor, establish Serial connections.
- Create classroom-specific LSTMModel instances (8 input features).
- Load existing fine-tuned model or create new structure.
- If new, load & transfer pre-trained weights from Step 2.

WORKING

4.Live Data Processing (RPi - lstm.py):

- Continuously read & parse Serial data from Arduinos.
- Harmonize data: handle outliers, normalize occupancy, add 'hour' feature (yields 8 features).
- Store harmonized data for training & prediction buffers.

5.Model Fine-tuning (RPi - lstm.py): If yes: prepare sequences from recent classroom data (8 features input -> 5 features output).

- Fit/apply classroom-specific MinMaxScaler.
- Compile model (lower LR if fine-tuning) & train with callbacks.
- Save updated classroom-specific model.

6.Prediction & Recommendation (RPi - lstm.py): Generate multi-step future predictions using the fine-tuned model.

- Inverse transform predictions to real-world units.
- **Calculate KPIv** for each predicted step based on predicted CO2/Occupancy.
- Score predicted future states based on temp, CO2, occupancy, and calculated KPIv.
- Recommend classroom with the best predicted score.

7.Cloud Upload :

- Periodically send latest sensor data to ThingSpeak

DATASETS

GAMS Indoor Air Quality Dataset

www.measureofquality.com

This dataset contains indoor and outdoor air quality data recorded by GAMS. The indoor data is tracked via GAMS indoor air quality monitor, while the outdoor data is recorded from GAMS API.

indoor

Column	Desc	Unit	Sample Value
co2	Carbon Dioxide CO2	--	708.0
humidity	humidity	--	72.09
pm10	PM 10	--	10.2
pm25	PM 2.5	--	9.0
temperature	temperature	Celsius	20.83
voc	VOC	--	0.062

Room Occupancy Estimation

Donated on 8/15/2023

Data set for estimating the precise number of occupants in a room using multiple non-intrusive environmental sensors temperature, light, sound, CO2 and PIR.

Dataset Characteristics	Subject Area	Associated Tasks
Multivariate, Time-Series	Computer Science	Classification
Feature Type	**# Instances**	**# Features**
Real	10129	18

Dataset Information

Additional Information

The experimental testbed for occupancy estimation was deployed in a 6m x 4.6m room. The setup consisted of 7 sensc and one edge node in a star configuration with the sensor nodes transmitting data to the edge every 30s using wireles transceivers. No HVAC systems were in use while the dataset was being collected....

SHOW MORE

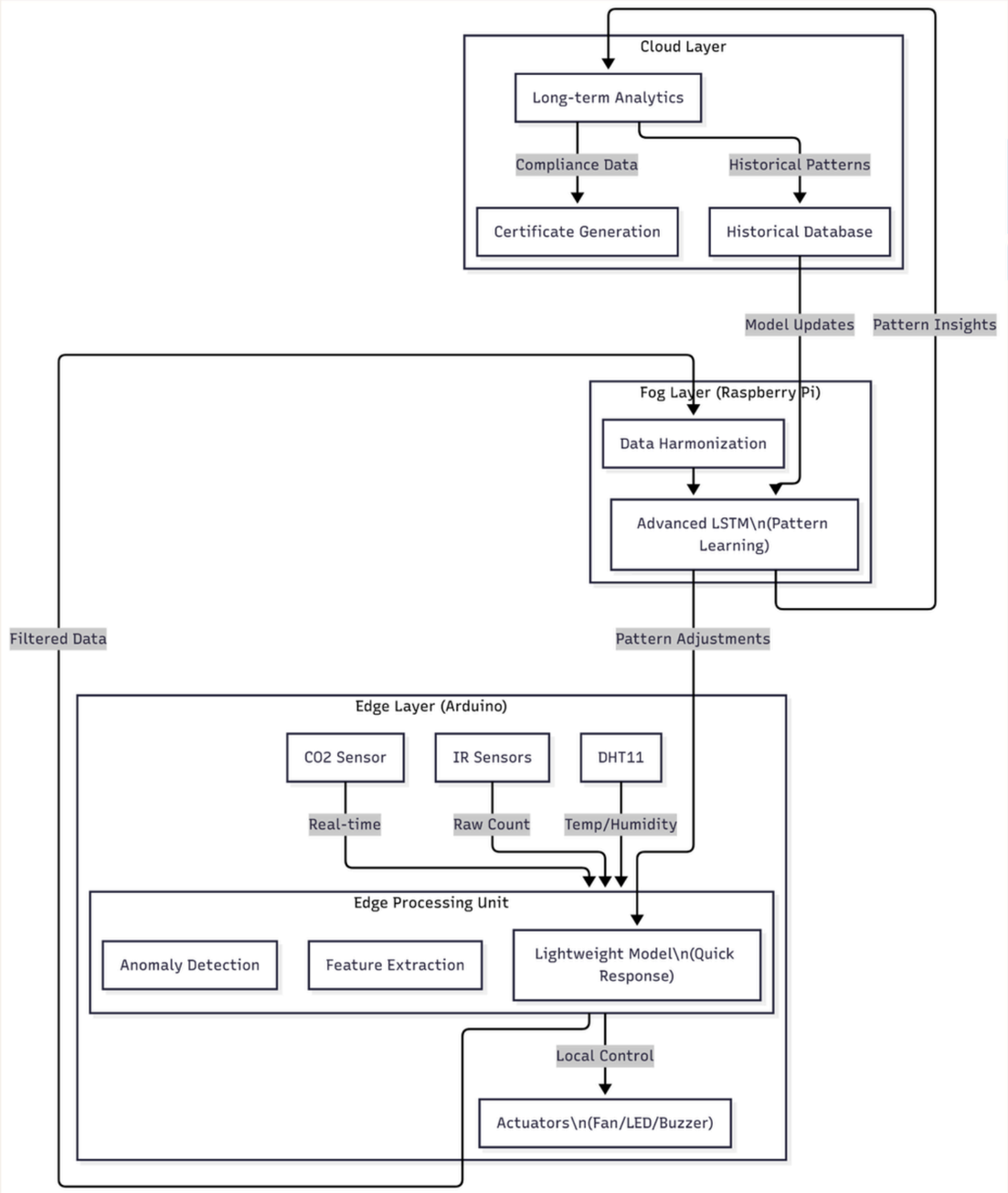
Has Missing Values?

No

Datasets (Q1)

Data set	Number of observations	Data Class Distribution (%)	
		0 (non-occupied)	1 (occupied)
Training	8143 of 7 variables	0.79	0.21
Testing 1	2665 of 7 variables	0.64	0.36
Testing 2	9752 of 7 variables	0.79	0.21

ARCHITECTURE



IMPLEMENTATION

```python
import pandas as pd
import numpy as np
from datetime import datetime
import warnings
warnings.filterwarnings('ignore')

def read_datasets(gams_path, occupancy_path, room_path):
    try:
        gams_df = pd.read_csv(gams_path)
        print(f"Successfully loaded GAMS dataset: {gams_df.shape[0]} rows")
    except Exception as e:
        print(f"Error loading GAMS dataset: {e}")
        gams_df = None

    try:
        occupancy_df = pd.read_csv(occupancy_path)
        print(f"Successfully loaded Occupancy dataset: {occupancy_df.shape[0]} rows")
    except Exception as e:
        print(f"Error loading Occupancy dataset: {e}")
        occupancy_df = None

    try:
        room_df = pd.read_csv(room_path)
        print(f"Successfully loaded Room dataset: {room_df.shape[0]} rows")
    except Exception as e:
        print(f"Error loading Room dataset: {e}")
        room_df = None

    return gams_df, occupancy_df, room_df

class KaggleDataHarmonizer:
    def __init__(self):
        self.unified_columns = [
            'timestamp',
            'co2',
            'temperature',
            'humidity',
            'occupancy',
            'light',
            'sound'
```

```python
def harmonize_datasets(self, gams_df, occupancy_df, room_df):
    harmonized_dfs = []

    if gams_df is not None:
        gams_processed = self._process_gams(gams_df)
        harmonized_dfs.append(gams_processed)

    if occupancy_df is not None:
        occupancy_processed = self._process_occupancy(occupancy_df)
        harmonized_dfs.append(occupancy_processed)

    if room_df is not None:
        room_processed = self._process_room(room_df)
        harmonized_dfs.append(room_processed)

    if harmonized_dfs:
        merged_df = pd.concat(harmonized_dfs, ignore_index=True)
        merged_df = merged_df.sort_values('timestamp')

        merged_df = merged_df.drop_duplicates(subset=['timestamp', 'data_source'])

        return merged_df
    else:
        raise ValueError("No datasets were successfully processed")

def _process_gams(self, df):
    processed = pd.DataFrame()

    processed['timestamp'] = pd.to_datetime(df['ts'])

    processed['co2'] = df['co2']
    processed['temperature'] = df['temperature']
    processed['humidity'] = df['humidity']
    processed['occupancy'] = (df['co2'] > 800).astype(int)

    processed['light'] = np.nan
    processed['sound'] = np.nan
    processed['pir'] = np.nan
    processed['data_source'] = 'gams'
```

IMPLEMENTATION

```python
    def _standardize_df(self, df):
        for col in self.unified_columns:
            if col not in df.columns:
                df[col] = np.nan

        df['timestamp'] = pd.to_datetime(df['timestamp'], errors='coerce')
        df['timestamp'] = df['timestamp'].dt.tz_localize(None)  # Remove timezone info

        df = df[self.unified_columns]
        df['co2'] = pd.to_numeric(df['co2'], errors='coerce')
        df['temperature'] = pd.to_numeric(df['temperature'], errors='coerce')
        df['humidity'] = pd.to_numeric(df['humidity'], errors='coerce')

        df.loc[df['co2'] > 5000, 'co2'] = np.nan
        df.loc[df['temperature'] > 50, 'temperature'] = np.nan
        df.loc[df['humidity'] > 100, 'humidity'] = np.nan

        return df

def analyze_harmonized_data(df):
    print("\nHarmonized Dataset Analysis:")
    print("-" * 50)
    print(f"Total rows: {df.shape[0]}")
    print(f"Date range: {df['timestamp'].min()} to {df['timestamp'].max()}")
    print("\nRows per data source:")
    print(df['data_source'].value_counts())
    print("\nMissing values per column:")
    print(df.isnull().sum())
    print("\nSummary statistics:")
    print(df.describe())
```

```python
def analyze_harmonized_data(df):
    print("\nHarmonized Dataset Analysis:")
    print("-" * 50)
    print(f"Total rows: {df.shape[0]}")
    print(f"Date range: {df['timestamp'].min()} to {df['timestamp'].max()}")
    print("\nRows per data source:")
    print(df['data_source'].value_counts())
    print("\nMissing values per column:")
    print(df.isnull().sum())
    print("\nSummary statistics:")
    print(df.describe())

if __name__ == "__main__":
    gams_df, occupancy_df, room_df = read_datasets(
        '../input/majorproj/gams_indoor.csv',
        '../input/majorproj/DataTraining.csv',
        '../input/majorproj/train (1).csv'
    )

    harmonizer = KaggleDataHarmonizer()
    try:
        harmonized_df = harmonizer.harmonize_datasets(gams_df, occupancy_df, room_df)

        harmonized_df.to_csv('harmonized_dataset.csv', index=False)
        print("\nSuccessfully saved harmonized dataset to 'harmonized_dataset.csv'")

        analyze_harmonized_data(harmonized_df)

    except Exception as e:
```

```
Successfully loaded GAMS dataset: 135099 rows
Successfully loaded Occupancy dataset: 8143 rows
Successfully loaded Room dataset: 8036 rows

Successfully saved harmonized dataset to 'harmonized_dataset.csv'

Harmonized Dataset Analysis:
--------------------------------------------------
Total rows: 149619
Date range: 2015-04-02 17:51:00 to 2023-11-22 23:59:58

Rows per data source:
data_source
gams         135069
room           8036
occupancy      6514
Name: count, dtype: int64

Missing values per column:
timestamp          0
co2                0
temperature        0
humidity        8036
occupancy          0
light         135069
sound         141583
pir           141583
data_source        0
dtype: int64

Summary statistics:
                           timestamp            co2    temperature  \
count                         149619  149619.000000  149619.000000
mean   2017-05-22 14:20:22.731297536     671.796952      22.971886
min              2015-04-02 17:51:00     345.000000      17.710000
25%              2017-01-02 00:12:30     426.000000      21.370000
50%              2017-02-09 22:22:00     476.000000      22.890000
75%              2017-03-07 23:15:30     805.000000      24.840000
max              2023-11-22 23:59:58    2626.000000      27.960000
std                              NaN     379.014946       2.101438

            humidity      occupancy          light         sound          pir
count  141583.000000  149619.000000  14550.000000  8036.000000  8036.000000
mean       37.320585       0.267921     66.826241     0.131513     0.110005
min        16.745000       0.000000      0.000000     0.052500     0.000000
25%        34.120000       0.000000      0.000000     0.062500     0.000000
50%        37.350000       0.000000      0.000000     0.070000     0.000000
75%        41.100000       1.000000     39.000000     0.075000     0.000000
max        72.090000       3.000000   1546.333333     3.182500     1.000000
std         5.875705       0.472365    141.952392     0.212157     0.312915
```

IMPLEMENTATION

IndoorVentilationModel.py 2, U

IndoorVentilationModel.py > load_and_preprocess_data

```python
import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
from sklearn.preprocessing import MinMaxScaler
import joblib # For saving the scaler
import os
import logging
from datetime import datetime

# Configure logging
logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s - %(levelname)s - %(message)s'
)

# --- Configuration ---
DATASET_PATH = './harmonized_dataset.csv'
MODEL_WEIGHTS_PATH = 'pretrained_lstm.weights.h5'
SCALER_PATH = 'pretrained_scaler.joblib'
SEQUENCE_LENGTH = 10 # Number of timesteps in each input sequence
NUM_FEATURES = 5 # co2, temp, hum, occ, hour
BATCH_SIZE = 64
EPOCHS = 5 # Adjust as needed


def load_and_preprocess_data(filepath):
    """
    Loads the merged dataset, selects relevant columns, handles missing values,
    and engineers the 'hour' feature.

    Args:
        filepath (str): Path to the CSV dataset.

    Returns:
        pd.DataFrame: Preprocessed data with columns ['co2', 'temperature', 'humidity', 'occupancy', 'hour'].
                      Returns None if loading fails.
    """
    logging.info(f"Loading data from: {filepath}")
    try:
        # Load dataset - Ensure 'timestamp' column is parsed as datetime
        df = pd.read_csv(filepath, parse_dates=['timestamp'])
        logging.info(f"Original dataset shape: {df.shape}")
```

```python
def build_pretrain_model(input_shape, num_output_features):
    """
    Builds the LSTM model architecture for pre-training.

    Args:
        input_shape (tuple): Shape of input sequences (sequence_length, num_features).
        num_output_features (int): Number of features to predict.

    Returns:
        tf.keras.models.Sequential: Compiled LSTM model.
    """
    logging.info(f"Building pre-training model with input shape {input_shape} and {num_output_features} outputs.")
    model = Sequential([
        LSTM(64, return_sequences=True, input_shape=input_shape, name='lstm_shared_1'),
        Dropout(0.2, name='dropout_shared_1'),
        LSTM(32, name='lstm_shared_2'),
        Dropout(0.2, name='dropout_shared_2'),
        Dense(16, activation='relu', name='dense_shared_1'),
        Dense(num_output_features, name='dense_pretrain_output') # Output layer
    ], name="PretrainIndoorLSTM")

    model.compile(optimizer='adam', loss='mse', metrics=['mae'])
    logging.info("Model compiled.")
    model.summary(print_fn=logging.info) # Log model summary
    return model
```

pretrained_lstm.weights.h5 U

pretrained_scaler.joblib U

IMPLEMENTATION

```python
# LCD Pin Configuration
LCD_RS = 25
LCD_EN = 24
LCD_D4 = 23
LCD_D5 = 17
LCD_D6 = 18
LCD_D7 = 22

# Initialize LCD
lcd = CharLCD(
    pin_rs=LCD_RS,
    pin_e=LCD_EN,
    pins_data=[LCD_D4, LCD_D5, LCD_D6, LCD_D7],
    numbering_mode=GPIO.BCM,
    cols=16,
    rows=2
)

# Classroom metadata - can be expanded
CLASSROOM_METADATA = {
    'classroom1': {
        'room_size': 50,  # square meters
        'capacity': 25,   # maximum people
        'ventilation_type': 'natural',
        'window_area': 8  # square meters
    },
    'classroom2': {
        'room_size': 60,
        'capacity': 30,
        'ventilation_type': 'hvac',
        'window_area': 6
    }
}
```

```python
PRETRAINED_WEIGHTS_PATH = 'pretrained_lstm.weights.h5'
PRETRAINED_INPUT_SHAPE = (10, 5) # Timesteps, Features (co2, temp, hum, occ
FINE_TUNE_LEARNING_RATE = 1e-4 # Lower learning rate for fine-tuning

# Configuration
CLOUD_UPLOAD_INTERVAL = 3600  # seconds (1 hour)
MODEL_UPDATE_INTERVAL = 86400  # seconds (24 hours)
DATA_HARMONIZATION_WINDOW = 5  # readings to use for harmonization

# Serial Port Configuration
usb0_port = "/dev/ttyS0"
usb1_port = "/dev/ttyS1"
baud_rate = 9600
```

```python
def create_model(self):
    """Create a new LSTM model with consistent layer names for potential weight transfer."""
    # Assuming input shape is (timesteps, features)
    timesteps, features = self.input_shape
    logging.info(f"Creating model architecture with input shape: ({timesteps}, {features})")
    model = Sequential([
        # Use consistent names matching the pre-training script
        LSTM(64, return_sequences=True, input_shape=(timesteps, features), name='lstm_shared_1'),
        Dropout(0.2, name='dropout_shared_1'),
        LSTM(32, name='lstm_shared_2'),
        Dropout(0.2, name='dropout_shared_2'),
        Dense(16, activation='relu', name='dense_shared_1'),
        # Final Dense layer specific to this model's output (5 features)
        Dense(5, name='dense_target_output') # Output: temp, humidity, co2, occupancy (normalized), KPIv
    ], name=f"ClassroomLSTM_{self.classroom_id}")

    # Compile step moved to train() method for fine-tuning flexibility
    return model
```

IMPLEMENTATION

```python
# LCD Pin Configuration
LCD_RS = 25
LCD_EN = 24
LCD_D4 = 23
LCD_D5 = 17
LCD_D6 = 18
LCD_D7 = 22

# Initialize LCD
lcd = CharLCD(
    pin_rs=LCD_RS,
    pin_e=LCD_EN,
    pins_data=[LCD_D4, LCD_D5, LCD_D6, LCD_D7],
    numbering_mode=GPIO.BCM,
    cols=16,
    rows=2
)

# Classroom metadata - can be expanded
CLASSROOM_METADATA = {
    'classroom1': {
        'room_size': 50,  # square meters
        'capacity': 25,   # maximum people
        'ventilation_type': 'natural',
        'window_area': 8  # square meters
    },
    'classroom2': {
        'room_size': 60,
        'capacity': 30,
        'ventilation_type': 'hvac',
        'window_area': 6
    }
}
```

```python
PRETRAINED_WEIGHTS_PATH = 'pretrained_lstm.weights.h5'
PRETRAINED_INPUT_SHAPE = (10, 5) # Timesteps, Features (co2, temp, hum, occ
FINE_TUNE_LEARNING_RATE = 1e-4 # Lower learning rate for fine-tuning

# Configuration
CLOUD_UPLOAD_INTERVAL = 3600  # seconds (1 hour)
MODEL_UPDATE_INTERVAL = 86400  # seconds (24 hours)
DATA_HARMONIZATION_WINDOW = 5  # readings to use for harmonization

# Serial Port Configuration
usb0_port = "/dev/ttyS0"
usb1_port = "/dev/ttyS1"
baud_rate = 9600
```

```python
def create_model(self):
    """Create a new LSTM model with consistent layer names for potential weight transfer."""
    # Assuming input shape is (timesteps, features)
    timesteps, features = self.input_shape
    logging.info(f"Creating model architecture with input shape: ({timesteps}, {features})")
    model = Sequential([
        # Use consistent names matching the pre-training script
        LSTM(64, return_sequences=True, input_shape=(timesteps, features), name='lstm_shared_1'),
        Dropout(0.2, name='dropout_shared_1'),
        LSTM(32, name='lstm_shared_2'),
        Dropout(0.2, name='dropout_shared_2'),
        Dense(16, activation='relu', name='dense_shared_1'),
        # Final Dense layer specific to this model's output (5 features)
        Dense(5, name='dense_target_output') # Output: temp, humidity, co2, occupancy (normalized), KPIv
    ], name=f"ClassroomLSTM_{self.classroom_id}")

    # Compile step moved to train() method for fine-tuning flexibility
    return model
```

IMPLEMENTATION

```python
def upload_data(self, classroom_id, data):
    """Upload classroom data to ThingSpeak"""
    # Check if enough time has passed since last upload
    current_time = time.time()
    if classroom_id in self.last_upload_time and \
       (current_time - self.last_upload_time[classroom_id]) < self.upload_interval:
        logging.info(f"Skipping upload - need to wait {self.upload_interval}s between updates")
        return False

    # Extract values from data
    temperature = data[0]
    humidity = data[1]
    co2 = data[2]
    occupancy = data[3]  # People count
    kpiv = data[5] if len(data) > 5 else 0  # KPIv value
    trend = data[6] if len(data) > 6 else 0  # Trend prediction

    # Calculate alert status
    alert_status = 0
    if kpiv >= 1.0 or co2 > 1000:
        alert_status = 2  # Critical
    elif kpiv >= 0.8 or co2 > 800:
        alert_status = 1  # Warning

    # Get API key for this classroom
    api_key = self.api_keys.get(classroom_id)
    if not api_key:
        logging.warning(f"No API key found for {classroom_id}")
        return False

    # Build payload
    payload = {
        'api_key': api_key,
        'field1': temperature,
        'field2': humidity,
        'field3': co2,
        'field4': occupancy,
        'field5': kpiv,
        'field6': trend,
        'field7': alert_status,
        'field8': 1.0  # Model version
    }

    try:
        # Send data to ThingSpeak
        response = requests.get(self.base_url, params=payload)
        if response.status_code == 200:
            self.last_upload_time[classroom_id] = current_time
            logging.info(f"Data uploaded to ThingSpeak for {classroom_id}")
            return True
        else:
            logging.error(f"Error uploading to ThingSpeak: {response.status_code}")
            return False
```

```python
def send_model_updates(self, classroom_id):
    """Send model parameter updates to Arduino"""
    serial_conn = self.ser_usb0 if classroom_id == 'classroom1' else self.ser_usb1

    # Generate parameters based on recent data
    params = self.calculate_model_parameters(classroom_id)

    command = "MODEL:"
    for key, value in params.items():
        command += f"{key}:{value},"
    command = command[:-1]  # Remove trailing comma

    try:
        serial_conn.write((command + "\n").encode())
        logging.info(f"Sent model update to {classroom_id}: {command}")
        return True
    except Exception as e:
        logging.error(f"Error sending model update: {e}")
        return False
```

IMPLEMENTATION

```
#include <LiquidCrystal.h>
#include <DHT.h>

#define DHTPIN A1
#define DHTTYPE DHT11
#define CO2_THRESHOLD 800     // Updated threshold for CO2
#define TEMP_THRESHOLD 34
#define ALARM_CO2 1000        // CO2 alarm threshold for KPIv
#define VN 2.0                // Predefined value when CO2 exceeds threshold

// CO2 to person estimation parameters
#define BASE_CO2 400          // Baseline outdoor CO2 (ppm)
#define CO2_PER_PERSON 20     // Approximate CO2 contributed per person (ppm)

String uno;
LiquidCrystal lcd(2, 3, 4, 5, 6, 7);
DHT dht(DHTPIN, DHTTYPE);

int in_sensor_pin = 8;
int out_sensor_pin = 9;
int co2_pin = A0;
int relay_pin = 11;
int buzzer_pin = 10;
```

```
void setup() {
  dht.begin();
  Serial.begin(9600);
  lcd.begin(16, 2);

  lcd.setCursor(0, 0);
  lcd.print("Edge-Driven CO2");
  lcd.setCursor(0, 1);
  lcd.print("Prediction V2"); // Indicate updated version
  delay(2000);
  lcd.clear();

  pinMode(in_sensor_pin, INPUT);
  pinMode(out_sensor_pin, INPUT);
  pinMode(co2_pin, INPUT);
  pinMode(relay_pin, OUTPUT);
  pinMode(buzzer_pin, OUTPUT);

  digitalWrite(relay_pin, HIGH); // Relay is likely HIGH to be off
  digitalWrite(buzzer_pin, LOW);

  // Initialize buffers
  for(int i = 0; i < BUFFER_SIZE; i++) {
    co2_buffer[i] = 0;
    temp_buffer[i] = 0;
  }
}
```

IMPLEMENTATION

```
void updateOccupancy() {
  // Read debounced sensor states (1 = beam broken, 0 = beam clear)
  int in_state = readDebounced(in_sensor_pin, lastInState, lastDebounceTimeIn);
  int out_state = readDebounced(out_sensor_pin, lastOutState, lastDebounceTimeOut);

  // If stable state has changed, update internal debounced state
  if(in_state != stableInState) stableInState = in_state;
  if(out_state != stableOutState) stableOutState = out_state;

  switch (currentState) {
    case IDLE:
      if (stableInState == HIGH && stableOutState == LOW) { // Inner beam broken firs
        currentState = IN_FIRST;
      } else if (stableInState == LOW && stableOutState == HIGH) { // Outer beam brok
        currentState = OUT_FIRST;
      }
      // If both break simultaneously, ignore for now or handle as error
      break;

    case IN_FIRST:
      if (stableInState == HIGH && stableOutState == HIGH) { // Outer beam also break
        currentState = PASSING_IN;
      } else if (stableInState == LOW && stableOutState == LOW) { // Went back out wi
        currentState = IDLE;
      }
      // Stay in IN_FIRST if only inner beam is broken
      break;
```

```
    case PASSING_IN: // Both beams broken, started with inner
      if (stableInState == LOW && stableOutState == HIGH) { // Inner beam
        people_count++;
        currentState = OUT_FIRST; // Now only outer is broken
        lcd.clear();
        lcd.print("Entered!");
        lcd.setCursor(0,1);
        lcd.print("Count: "); lcd.print(people_count);
        delay(500); // Short delay for message
      } else if (stableInState == LOW && stableOutState == LOW) { // Both
        currentState = IDLE; // Reset, missed exit trigger? Or fast pass
        people_count++;
        lcd.clear();
        lcd.print("Entered (Fast!)");
        lcd.setCursor(0,1);
        lcd.print("Count: "); lcd.print(people_count);
        delay(500);
      }
      // Stay in PASSING_IN if both are still broken
      break;

    case PASSING_OUT: // Both beams broken, started with outer
      if (stableInState == HIGH && stableOutState == LOW) { // Outer beam
        if (people_count > 0) people_count--;
        currentState = IN_FIRST; // Now only inner is broken
        lcd.clear();
        lcd.print("Exited!");
        lcd.setCursor(0,1);
        lcd.print("Count: "); lcd.print(people_count);
        delay(500); // Short delay for message
      } else if (stableInState == LOW && stableOutState == LOW) { // Both
        currentState = IDLE; // Reset, missed exit trigger? Or fast pass
        if (people_count > 0) people_count--;
        lcd.clear();
        lcd.print("Exited (Fast!)");
        lcd.setCursor(0,1);
        lcd.print("Count: "); lcd.print(people_count);
        delay(500);
      }
```

IMPLEMENTATION

```
// Lightweight prediction model
float predictTrend(int current_co2, int current_temp, int current_humidity) {
  // Get averages
  int avg_co2 = calculateMovingAverage(co2_buffer, BUFFER_SIZE);
  int avg_temp = calculateMovingAverage(temp_buffer, BUFFER_SIZE);

  // Calculate trends (positive = rising, negative = falling)
  float co2_trend = current_co2 - avg_co2;
  float temp_trend = current_temp - avg_temp;

  // Normalize trends
  co2_trend = co2_trend / 50.0;  // Normalize by expected variation
  temp_trend = temp_trend / 2.0;  // Normalize by expected variation

  // Apply weights
  float weighted_trend = (co2_weight * co2_trend) +
                         (temp_weight * temp_trend) +
                         (humidity_weight * (people_count > 0 ? 0.5 : -0.5)); // Humidi

  return weighted_trend;
}

// Calculate KPIv (Ventilation KPI)
float calculateKPIv(int carbon_value, int actual_people) {
  if (carbon_value > ALARM_CO2) {
    return VN;
  }
  int estimated_people = (carbon_value - BASE_CO2) / CO2_PER_PERSON;
  if (estimated_people < 0) estimated_people = 0;

  if (actual_people > 0) {
    return (float)estimated_people / actual_people;
  } else {
    // If no actual people, but CO2 is above baseline, ventilation might be poor
    return (estimated_people > 0) ? 0.5 : 0.0;
  }
}
```

```
// Check for model updates from Raspberry Pi
void checkForModelUpdates() {
  if (Serial.available() > 0) {
    String data = Serial.readStringUntil('\n');
    if (data.startsWith("MODEL:")) {
      data.trim(); // Remove potential newline characters
      data = data.substring(6); // Remove "MODEL:"
      int start = 0;
      int end = data.indexOf(',');
      while (end != -1) {
        updateParameter(data.substring(start, end));
        start = end + 1;
        end = data.indexOf(',', start);
      }
      updateParameter(data.substring(start)); // Process last pa
      lcd.clear();
      lcd.setCursor(0, 0);
      lcd.print("Model Updated!");
      delay(1000);
    }
  }
}

// Update model parameter
void updateParameter(String param) {
  int colon = param.indexOf(':');
  if (colon == -1) return; // Invalid format
  String name = param.substring(0, colon);
  float value = param.substring(colon + 1).toFloat();

  if (name == "co2_weight") co2_weight = value;
  else if (name == "temp_weight") temp_weight = value;
  else if (name == "humidity_weight") humidity_weight = value;
  else if (name == "trend_threshold") trend_threshold = value;
}
```

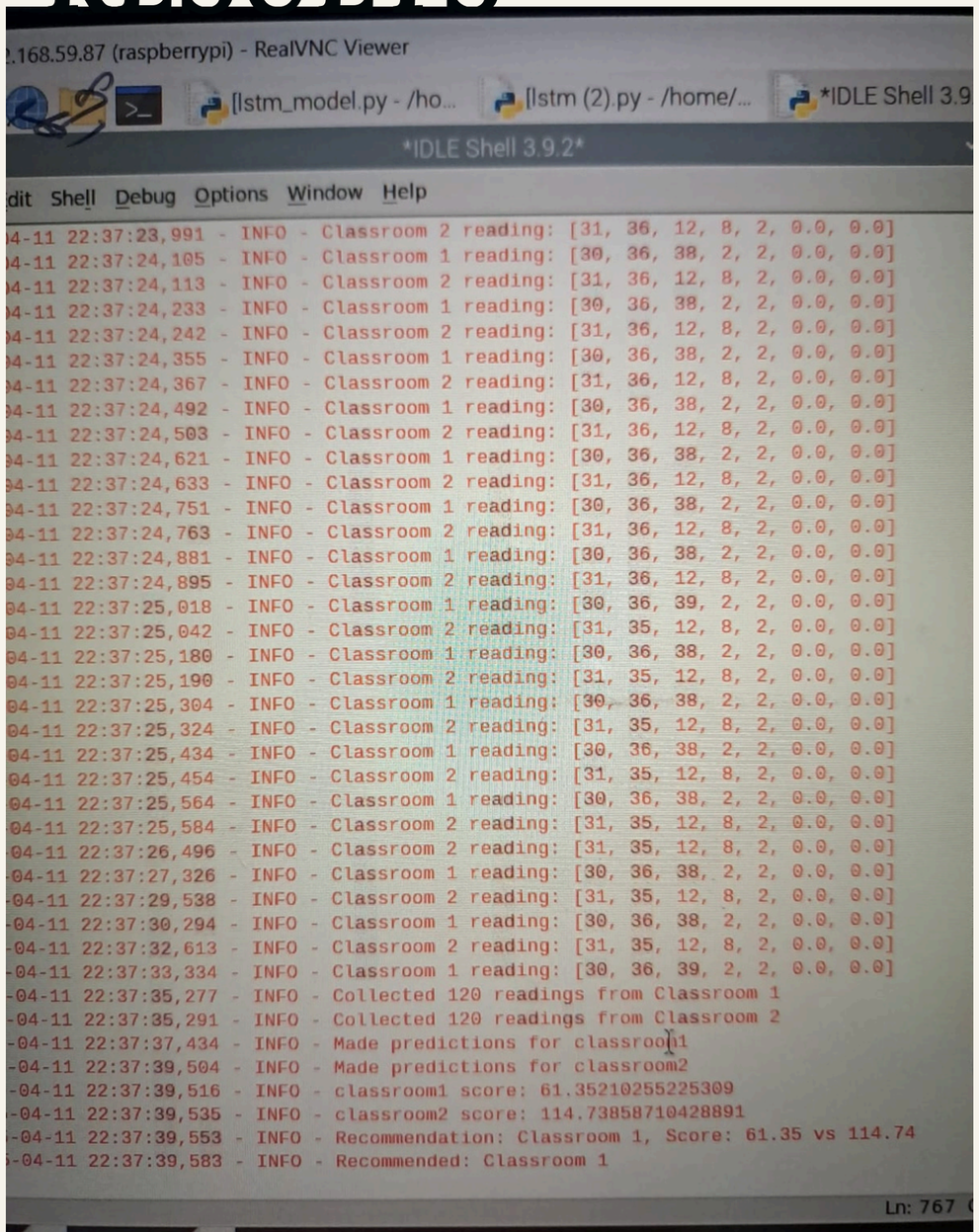
IMPLEMENTATION

```
// Send data to Raspberry Pi (maybe less frequently?)
static unsigned long lastSerialSend = 0;
if (currentTime - lastSerialSend > 5000) { // Send every 5 seconds
    uno = String("a") + String(temperature) +
          String("b") + String(humidity) +
          String("c") + String(carbon_value) +
          String("d") + String(people_count) + // Send corrected people_count
          String("e") + String(0) + // Sending 0 for out_count, as it's not used for occupancy
          String("f") + String(kpiv, 2) +
          String("g") + String(trend, 2) +
          String("h");
    Serial.println(uno);
    lastSerialSend = currentTime;
}
```

IMPLEMENTATION

```
// Send data to Raspberry Pi (maybe less frequently?)
static unsigned long lastSerialSend = 0;
if (currentTime - lastSerialSend > 5000) { // Send every 5 seconds
    uno = String("a") + String(temperature) +
          String("b") + String(humidity) +
          String("c") + String(carbon_value) +
          String("d") + String(people_count) + // Send corrected people_count
          String("e") + String(0) + // Sending 0 for out_count, as it's not used for occupancy
          String("f") + String(kpiv, 2) +
          String("g") + String(trend, 2) +
          String("h");
    Serial.println(uno);
    lastSerialSend = currentTime;
}
```

RESULTS


2.168.59.87 (raspberrypi) - RealVNC Viewer
[lstm_model.py - /ho...
[lstm (2).py - /home/...
*IDLE Shell 3.9
IDLE Shell 3.9.2
dit Shell Debug Options Window Help
04-11 22:37:23,991 - INFO - Classroom 2 reading: [31, 36, 12, 8, 2, 0.0, 0.0]
04-11 22:37:24,105 - INFO - Classroom 1 reading: [30, 36, 38, 2, 2, 0.0, 0.0]
04-11 22:37:24,113 - INFO - Classroom 2 reading: [31, 36, 12, 8, 2, 0.0, 0.0]
04-11 22:37:24,233 - INFO - Classroom 1 reading: [30, 36, 38, 2, 2, 0.0, 0.0]
04-11 22:37:24,242 - INFO - Classroom 2 reading: [31, 36, 12, 8, 2, 0.0, 0.0]
04-11 22:37:24,355 - INFO - Classroom 1 reading: [30, 36, 38, 2, 2, 0.0, 0.0]
04-11 22:37:24,367 - INFO - Classroom 2 reading: [31, 36, 12, 8, 2, 0.0, 0.0]
04-11 22:37:24,492 - INFO - Classroom 1 reading: [30, 36, 38, 2, 2, 0.0, 0.0]
04-11 22:37:24,503 - INFO - Classroom 2 reading: [31, 36, 12, 8, 2, 0.0, 0.0]
04-11 22:37:24,621 - INFO - Classroom 1 reading: [30, 36, 38, 2, 2, 0.0, 0.0]
04-11 22:37:24,633 - INFO - Classroom 2 reading: [31, 36, 12, 8, 2, 0.0, 0.0]
04-11 22:37:24,751 - INFO - Classroom 1 reading: [30, 36, 38, 2, 2, 0.0, 0.0]
04-11 22:37:24,763 - INFO - Classroom 2 reading: [31, 36, 12, 8, 2, 0.0, 0.0]
04-11 22:37:24,881 - INFO - Classroom 1 reading: [30, 36, 38, 2, 2, 0.0, 0.0]
04-11 22:37:24,895 - INFO - Classroom 2 reading: [31, 36, 12, 8, 2, 0.0, 0.0]
04-11 22:37:25,018 - INFO - Classroom 1 reading: [30, 36, 39, 2, 2, 0.0, 0.0]
04-11 22:37:25,042 - INFO - Classroom 2 reading: [31, 35, 12, 8, 2, 0.0, 0.0]
04-11 22:37:25,180 - INFO - Classroom 1 reading: [30, 36, 38, 2, 2, 0.0, 0.0]
04-11 22:37:25,190 - INFO - Classroom 2 reading: [31, 35, 12, 8, 2, 0.0, 0.0]
04-11 22:37:25,304 - INFO - Classroom 1 reading: [30, 36, 38, 2, 2, 0.0, 0.0]
04-11 22:37:25,324 - INFO - Classroom 2 reading: [31, 35, 12, 8, 2, 0.0, 0.0]
04-11 22:37:25,434 - INFO - Classroom 1 reading: [30, 36, 38, 2, 2, 0.0, 0.0]
04-11 22:37:25,454 - INFO - Classroom 2 reading: [31, 35, 12, 8, 2, 0.0, 0.0]
04-11 22:37:25,564 - INFO - Classroom 1 reading: [30, 36, 38, 2, 2, 0.0, 0.0]
04-11 22:37:25,584 - INFO - Classroom 2 reading: [31, 35, 12, 8, 2, 0.0, 0.0]
04-11 22:37:26,496 - INFO - Classroom 2 reading: [31, 35, 12, 8, 2, 0.0, 0.0]
04-11 22:37:27,326 - INFO - Classroom 1 reading: [30, 36, 38, 2, 2, 0.0, 0.0]
04-11 22:37:29,538 - INFO - Classroom 2 reading: [31, 35, 12, 8, 2, 0.0, 0.0]
04-11 22:37:30,294 - INFO - Classroom 1 reading: [30, 36, 38, 2, 2, 0.0, 0.0]
04-11 22:37:32,613 - INFO - Classroom 2 reading: [31, 35, 12, 8, 2, 0.0, 0.0]
04-11 22:37:33,334 - INFO - Classroom 1 reading: [30, 36, 39, 2, 2, 0.0, 0.0]
04-11 22:37:35,277 - INFO - Collected 120 readings from Classroom 1
04-11 22:37:35,291 - INFO - Collected 120 readings from Classroom 2
04-11 22:37:37,434 - INFO - Made predictions for classroom1
04-11 22:37:39,504 - INFO - Made predictions for classroom2
04-11 22:37:39,516 - INFO - classroom1 score: 61.35210255225309
04-11 22:37:39,535 - INFO - classroom2 score: 114.73858710428891
04-11 22:37:39,553 - INFO - Recommendation: Classroom 1, Score: 61.35 vs 114.74
04-11 22:37:39,583 - INFO - Recommended: Classroom 1
Ln: 767


C1 Readings: 74
C2 Readings: 74

RESULTS

RESULTS

Classroom Environment Certificate

classroom2 - March 12, 2025 to April 11, 2025

A+

95/100

Environmental Quality Rating

Based on 6 readings over the specified period

Average Metrics

Temperature	Humidity
31.7°C	38.0%

CO_2 Level	Occupancy
12 ppm	0.6

KPIv (Ventilation Quality Index)

0.00

Lower values indicate better ventilation

Min/Max Values

Metric	Minimum	Maximum	Range
Temperature (°C)	31.0	32.0	1.0
Humidity (%)	36.0	40.0	4.0
CO_2 (ppm)	11	13	2
KPIv	0.00	0.00	0.00

Recommendations

- Adjust temperature to optimal range (20-26°C). Current: 31.7°C.
- Maintain humidity between 40-60%. Current: 38.0%.

Conclusions

- **Seamless Integration of Fog and Edge Computing**
  - The architecture presented demonstrates a robust solution that leverages both fog and edge computing to optimize IoT data processing. By distributing intelligence between edge nodes and a central fog node, we ensure real-time decision-making and reduce latency, which is crucial for time-sensitive applications.

- **Efficient Resource Management**
  - By deploying LSTM models at both the fog and edge levels, the system efficiently processes sensor data locally, reducing the need for constant data transmission. This not only saves bandwidth but also improves the overall performance and response time of the system.

- **Scalability and Flexibility**
  - The architecture allows for scalable deployment across multiple edge nodes without compromising performance. Its modular design ensures the seamless integration of additional sensors, actuators, and nodes, making it adaptable for various IoT use cases

BIBILOGRAPHY

- An AI-Based Ventilation KPI Using Embedded IoT Devices. https://ieeexplore.ieee.org/document/1002157
- kaggle- Air quality dataset https://www.kaggle.com/datasets/saurabhshahane/adl-classification/data
- Low-Cost SCADA/HMI with Tiny Machine Learning for Monitoring Indoor CO2 Concentration .https://ieeexplore.ieee.org/document/10561089
- SMOTE and Gaussian Noise Based Sensor DataAugmentation https://ieeexplore.ieee.org/document/8907003
- Occupancy Estimation using WiFi: A Case Study for Counting Passengers on Busses https://ieeexplore.ieee.org/document/10561089/keywords#keywords

THANK YOU