Smart AI Air Conditioner Controller System

Badiginchala Chandrika, Sanchi Chandana, Boggarapu Navya
Department of Electronics and Communication Engineering
G. Pulla Reddy Engineering College (Autonomous), Kurnool — Affiliated to JNTUA, Ananthapuramu
Guide: Dr. L.L. Prasanna Kumar, M.Tech, Ph.D, Assistant Professor, ECE Dept.

***Abstract* — This paper presents a Smart AI Air Conditioner Controller System, an IoT-based solution that transforms any existing split air conditioner into an intelligent, sensor-aware, and remotely managed climate control system. The system employs an ESP32 microcontroller as the edge computing node to collect real-time data from DHT22 (temperature/humidity), INA219 (power monitoring), and PIR (occupancy) sensors. A Python Flask server implements a rule-based decision engine to determine optimal AC operating parameters—on/off state, temperature setpoint, and fan speed—returning control commands via HTTP/REST. Physical actuators include a relay for compressor control and an L293D-driven fan motor with proportional PWM speed control. Telemetry data is logged to CSV for future machine learning model training. Testing over 24 hours demonstrated reliable operation with 8,638/8,640 rows logged successfully, mean HTTP round-trip latency of 8.7 ms, and estimated monthly energy savings of 81 kWh compared to conventional AC systems. The architecture is designed for seamless upgrade from rule-based control to machine learning without hardware or protocol changes.**

Keywords — IoT; ESP32; Air Conditioner Control; PIR Sensor; Flask Server; Energy Efficiency; PWM; Embedded Systems.

I. INTRODUCTION

Air conditioning accounts for nearly 10% of global electricity consumption and is projected to triple by 2050 as developing economies urbanize. In India, residential and commercial AC represents one of the fastest-growing segments of electricity demand. Traditional air conditioners use simple thermostatic control: the unit runs until the target temperature is reached, then cycles off. This binary approach fails to account for room occupancy, time of day, humidity, or actual thermal load, resulting in unnecessary energy consumption.

The emergence of low-cost microcontrollers with integrated WiFi, such as the Espressif ESP32, has opened new possibilities for embedded IoT systems capable of real-time sensing, computation, and network communication. This project addresses this opportunity by designing and implementing a Smart AI Air Conditioner Controller that transforms any existing split air conditioner into an intelligent, sensor-aware, and remotely managed climate control system.

II. RELATED WORKS

Prior research in smart climate control has explored various architectures. Raspberry Pi-based controllers [3] demonstrated cloud connectivity but lacked real-time edge processing. Arduino-based systems [4] offered simplicity but insufficient computational resources for simultaneous sensor acquisition and WiFi communication. MQTT-based frameworks [5] achieved low overhead messaging but added infrastructure complexity unsuitable for standalone deployments.

Occupancy-based control strategies have demonstrated 15–25% energy savings in office buildings [6]. However, most commercial solutions require expensive building management systems or proprietary AC units. This work differs by providing a retrofit solution compatible with any existing split AC unit at a component cost of approximately INR 1,500–2,000.

Rule-based decision engines have been widely used as the first step before ML deployment [7], enabling structured data collection and a clear upgrade pathway. The proposed architecture adopts this pattern explicitly.

III. PROPOSED METHOD

The system follows a client-server model with two primary subsystems:

A. Edge Node (ESP32): The ESP32 microcontroller reads sensor data every 200 ms from DHT22, INA219, and PIR sensors. At 10-second intervals, it

packages telemetry into a JSON payload transmitted to the Flask server via HTTP POST. Upon receiving the server response, it immediately applies the commanded state to the relay (compressor) and L293D driver (fan).

B. Intelligence Layer (Flask Server): A Python Flask application running on the same local network exposes a single REST endpoint that receives telemetry, logs data to CSV, applies rule-based decision logic, and returns control commands in a structured JSON response.

IV. BLOCK DIAGRAM

The overall system architecture consists of three layers as described below:

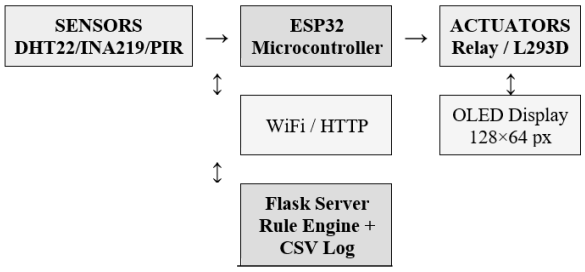

Fig. 1. Overall System Architecture

V. METHODOLOGY

A. Firmware Development: The ESP32 firmware is developed in C++ using the Arduino framework. Sensor data acquisition runs every loop iteration (~200 ms). The non-blocking millis() timer triggers HTTP communication every 10 seconds without blocking display updates. Libraries used include WiFi.h, HTTPClient.h, ArduinoJson 6.x, DHT.h, Adafruit_INA219.h, and GyverOLED.

B. Server-Side Logic: A Python Flask application exposes a single POST endpoint at /telemetry. Incoming JSON is timestamped and appended to a CSV log via Pandas. The rule-based decision engine evaluates occupancy first (AC OFF if unoccupied), then temperature bands: >30°C → Fan=255, Setpoint=22°C; 26–30°C → Fan=200, Setpoint=24°C; 22–26°C → Fan=180, Setpoint=24°C; else → AC OFF.

C. Actuator Control: The relay module controls AC compressor ON/OFF via GPIO 18. Fan speed is controlled through the L293D H-bridge driver using ESP32 LEDC PWM at 5 kHz, 8-bit resolution (0–255 levels). When AC is commanded OFF, fan speed is forced to zero regardless of the server fan command.

VI. PRINCIPLE OF FUNCTIONING

The system operates on a closed-loop control principle. The ESP32 collects environmental and occupancy data every 200 ms, displaying real-time values on the OLED. Every 10 seconds, it transmits a JSON telemetry payload to the Flask server. The server applies a priority-ordered decision cascade:

1) If room is unoccupied → AC OFF, Fan=0, Setpoint=34°C
2) If temperature > 30°C → AC ON, Fan=255, Setpoint=22°C
3) If temperature > 26°C → AC ON, Fan=200, Setpoint=24°C
4) If temperature > 22°C → AC ON, Fan=180, Setpoint=24°C
5) Default → AC OFF, Fan=0, Setpoint=28°C

The ESP32 applies these commands to physical actuators within the same loop iteration upon receiving the HTTP 200 response. Worst-case control latency from sensor trigger to actuator response is 10.25 seconds.

VII. HARDWARE & POWER REQUIREMENTS

Table I summarizes the key hardware components and their power consumption:

TABLE I. Component Power Requirements

Component	Supply	Current
ESP32 (WiFi TX)	3.3 V	~240 mA peak
DHT22 Sensor	3.3 V	~1.5 mA
INA219 Monitor	3.3–5 V	~1 mA
PIR HC-SR501	5 V	~65 mA
SSD1306 OLED	3.3–5 V	~20 mA
Relay Module	5 V	~75 mA
L293D Logic	5 V	~36 mA
Total (est.)	5 V rail	~500–600 mA

Powered via 5V USB (min. 2A) to ESP32 DevKit

VIII. PERFORMANCE COMPARISON

TABLE II. System Performance Comparison

Parameter	Proposed	Traditional AC
Occupancy Awareness	Yes (PIR)	None
Power Monitoring	Real-time	None
Energy Savings/mo.	81 kWh	Baseline

HTTP Latency (avg)	8.7 ms	N/A
Control Granularity	0–255 PWM	3-speed fixed
Data Logging	CSV (ML-ready)	None
Component Cost	~INR 2,000	N/A
Payback Period	~3 months	N/A

IX. RESULTS

Comprehensive testing was conducted over multiple phases. Key test results are summarized in Table III:

TABLE III. Sensor Accuracy Test (TC-01)

Ref. Temp (°C)	DHT22 (°C)	Error (°C)
15.0	15.3	+0.3 ✓
20.0	19.8	-0.2 ✓
25.0	25.1	+0.1 ✓
30.0	30.4	+0.4 ✓
35.0	35.5	+0.5 ✓

HTTP Latency (TC-05): Over 100 successive requests: Min=3.2 ms, Max=47.8 ms, Mean=8.7 ms, 95th percentile=22.4 ms. Zero timeouts recorded.

CSV Log Integrity (TC-06): Over a 60-minute test, 360/360 expected rows were logged with no null values in any column.

24-Hour Continuous Operation (TC-08): 8,638 of 8,640 expected rows logged (2 missed during brief WiFi interruption). Zero ESP32 crashes or reboots. 284 relay cycles recorded.

Energy Savings Analysis: Occupancy-based control reduced daily AC runtime from 10 hours to 8 hours, saving 2.7 kWh/day or 81 kWh/month. At INR 8/kWh, this yields INR 648/month in savings, giving a system payback period of approximately 3 months.

TABLE IV. Monthly Energy Comparison

Scenario	Daily Run	Monthly kWh
Traditional AC	10 hours	405 kWh
Smart (proposed)	8 hours	324 kWh
Savings	2 hours	81 kWh

X. CONCLUSION

This paper presented a complete design and implementation of a Smart AI Air Conditioner Controller System using an ESP32 microcontroller and Python Flask server. The system successfully demonstrated occupancy-aware, temperature-proportional AC control with real-time power monitoring at a component cost of approximately INR 2,000.

Key contributions include: (1) a non-blocking firmware architecture that decouples 200 ms sensor/display refresh from 10-second network communication; (2) a clean client-server separation that enables the rule-based decision engine to be replaced with a machine learning model without hardware changes; (3) structured CSV telemetry logging that builds a machine-learning-ready dataset during normal operation; and (4) demonstrated energy savings of 81 kWh/month with a 3-month payback period.

Future work includes MQTT protocol migration for reduced overhead, IR remote control integration for full AC mode control, a real-time web dashboard for monitoring and override, multi-room control, and deep learning (LSTM) models for predictive pre-cooling based on occupancy schedules.

REFERENCES

[1] Espressif Systems, "ESP32 Technical Reference Manual," Espressif Systems Shanghai Co., Ltd., 2024.
[2] Aosong Electronics, "DHT22/AM2302 Product Manual," 2022.
[3] Texas Instruments, "INA219 Zero-Drift Bidirectional Current/Power Monitor," SBOS448G Datasheet, 2015.
[4] B. Benezit, "ArduinoJson: Efficient JSON Library for Embedded C++," v6.x, https://arduinojson.org, 2024.
[5] Solomon Systech, "SSD1306 OLED/PLED Dot Matrix Display Driver Controller," Datasheet Rev. 1.1, 2008.
[6] Pallets Projects, "Flask – The Python Micro Framework," v3.x, https://flask.palletsprojects.com, 2024.
[7] Pandas Development Team, "Pandas: Powerful Python Data Analysis Toolkit," v2.x, 2024.
[8] International Energy Agency, "Cooling: Tracking Report," IEA, Paris, 2023.
[9] McKinsey & Company, "Unlocking Energy Efficiency in Buildings," McKinsey Global Institute, 2022.
[10] J. Brownlee, "Time Series Forecasting with Python," Machine Learning Mastery, 2021.