PROJECT ABSTRACT

Local-LLM Home Controller using ESP32, MQTT and Ollama

A hybrid home-automation prototype in which an ESP32 senses room conditions and drives a relay board over MQTT, while a local Python agent backed by an Ollama LLM on the student's laptop makes supervisory control decisions and explains each one in plain language — with hard safety rules running on the ESP32 itself so the LLM can never override them.

Project type: Hybrid (hardware prototype + software dashboard)

Availability: Built to order

Suitable branches: IoT & Embedded, Electronics / E&TC, Computer / IT, AI & Machine Learning

Technologies: ESP32, MQTT (Mosquitto), Python, Ollama, DHT22, 4-channel relay module, HTML/CSS/JS dashboard

Price: Request a quotation

Prepared by Projectech · projectech.in — for academic project reference.

1. Introduction / Background

Cloud voice assistants can switch home appliances on and off, but they send household sensor data to remote servers, need accounts and subscriptions, and give no reason for what they do. A growing alternative is to run a small language model locally: Ollama lets a laptop host open models such as Llama 3.1:8b with no internet dependency and no API key, and MQTT (via the Mosquitto broker) is the standard lightweight messaging protocol that ties embedded devices to software. This project combines the two into a home controller with an honest engineering split: the ESP32 does the sensing and actuating, a Python agent on the laptop talks to a local LLM, and the LLM acts only as a supervisory advisor whose decisions are explained in the dashboard's decision log.

2. Problem Statement

Students encounter two extremes in home automation: cloud-dependent voice assistants that are black boxes, and threshold-based IoT demos that switch relays but cannot reason or explain. Neither teaches how modern edge-AI systems are actually architected — a reliable real-time layer plus a slower, smarter supervisory layer. The project addresses this gap by building a controller where the fast, safe decisions live in ESP32 firmware (rules with max-ON timers, a power ceiling and a temperature cut-off) and the slow, smart decisions come from a local LLM that proposes actions and justifies each one in plain language, demonstrating the layered architecture used in real agentic IoT systems.

3. Objectives

- Build an ESP32 sensor-actuator node (DHT22 temperature/humidity, 4-channel relay board) that publishes telemetry over MQTT.
- Run a local Mosquitto MQTT broker and a Python agent on the student's laptop, connected to an Ollama-hosted LLM.
- Implement the honest split: hard safety rules on the ESP32 (max-ON timers, 500 W power ceiling, 60 °C fire cut-off) that the LLM cannot override.
- Have the LLM make supervisory decisions (fan/lamp/pump scheduling, energy-saving suggestions) with a plain-language “why” for each decision.
- Provide a working web dashboard showing live telemetry, relay states and the agent's decision log.
- Deliver the complete student kit: prototype, firmware, agent code, wiring docs, report, PPT and viva Q&A.;

4. Proposed Methodology

The system is developed in three layers. (1) **Edge layer:** the ESP32 reads the DHT22 every 5 seconds, publishes to *home/<room>/state* topics over Wi-Fi, listens on *home/relay/<n>/set* for actuation commands, and enforces its safety rules independently of the laptop. (2) **Agent layer:** a Python program on the laptop subscribes to the state topics, builds a prompt with current readings, relay states and house rules, and calls the local Ollama model (e.g. Llama 3.1:8b) every decision cycle. The agent parses the model's structured reply (action + justification) and only issues MQTT commands that pass the ESP32's rule check. (3) **Presentation layer:** a single-file HTML dashboard subscribes to the same topics and renders live telemetry, relay states and the timestamped decision log with per-decision latency.

5. System Architecture / Working

The DHT22 measures room temperature and humidity; the ESP32 publishes these over MQTT (QoS 1) on a 5-second cadence. The Python agent on the laptop receives the readings, adds context (time of day, tank level, motion events, house rules) and asks the local LLM what to do. The LLM replies with an action and a justification — for example, turning the ceiling fan on at 29.4 °C and 68% humidity, or holding the water pump off because the tank is at 42%, above the 20% refill trigger. The agent forwards allowed actions as MQTT set-commands; the ESP32 applies them only if its own safety rules permit, then acknowledges on *.../ack*. Every decision, its justification and its round-trip latency (design target ~2–6 s) appear in the dashboard log. If the laptop or Wi-Fi drops, the ESP32 keeps running its rule-based control on its own.

6. Hardware / Software Requirements

- ESP32 development board (Wi-Fi) with USB power
- DHT22 temperature/humidity sensor module
- 4-channel relay module (5 V logic, 10 A/250 VAC contacts) with a small test load rig
- Breadboard, jumper wires, 5 V adapter
- Laptop running Mosquitto MQTT broker, Python 3 with paho-mqtt, and Ollama with a pulled model (e.g. Llama 3.1:8b)
- Arduino IDE (C/C++ firmware for the ESP32); a modern browser for the dashboard

7. Expected Outcomes

- A working prototype: ESP32 telemetry flowing over local MQTT with relay actuation acknowledged end-to-end.
- A Python agent that consults a local LLM and issues supervisory control decisions with plain-language justifications.
- A dashboard with live sensor graphs, relay states and a timestamped agent decision log.
- Demonstrated safety: rule-layer overrides (e.g. pump auto-off, fire cut-off) that the LLM cannot bypass.
- A complete report, PPT and viva Q&A; covering MQTT, edge vs supervisory control, and local-LLM deployment.

8. Applications

- Teaching the layered architecture of modern agentic IoT systems (real-time edge + supervisory AI).
- Demonstrating local-LLM deployment with Ollama — no cloud account, no API key, no data leaving the home.
- Energy-awareness demos: the agent explains why it switches loads, not just that it did.
- A base platform for final-year extensions: more sensors, voice input, or multi-room nodes.

9. Future Scope

- Additional sensor nodes (PIR motion, LDR light level, ultrasonic tank level) on a second ESP32.
- Local voice commands via an on-device wake-word + speech-to-text pipeline.
- Smaller fine-tuned models (e.g. Llama 3.2:1b/3b) compared for latency vs decision quality.
- Energy analytics: per-relay consumption estimates and weekly usage reports.
- Optional cloud mirror of the dashboard for remote viewing (data stays local by default).