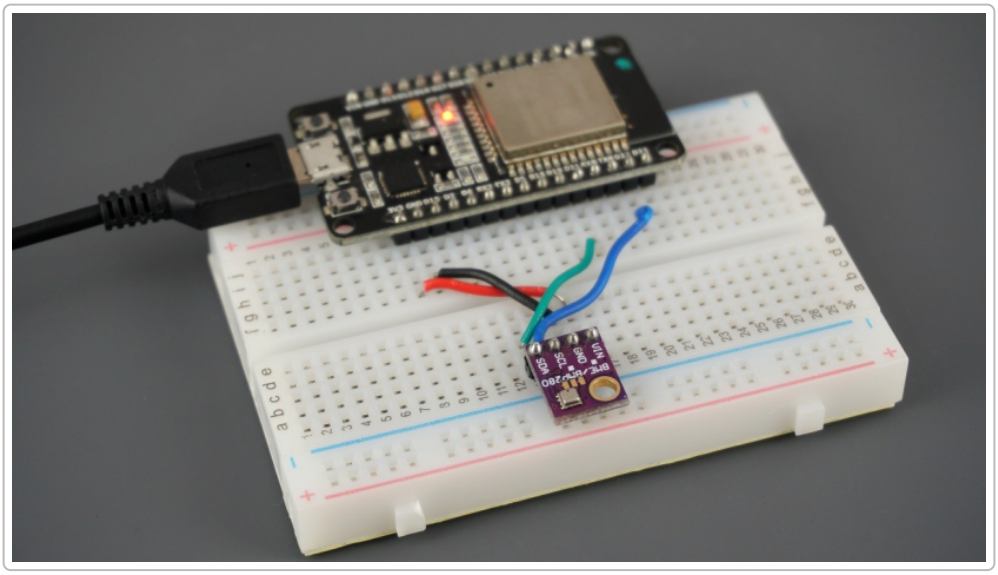
Prototype Design for HVAC Control

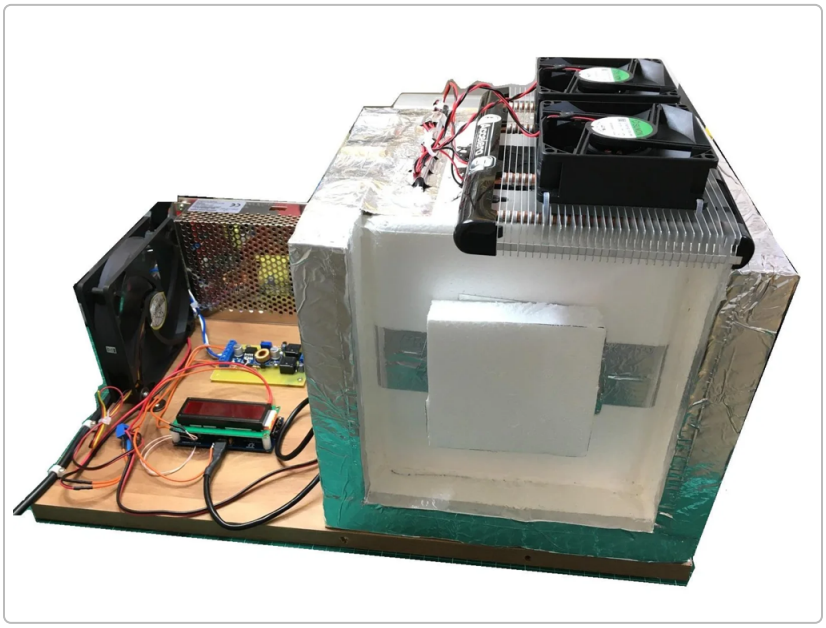
The goal is to build a small enclosed chamber (≈1 ft²) where only **temperature** and **humidity** are actively controlled. A well-insulated box (e.g. foam-lined plastic or wood) should house the sensors and actuators. The chamber should have tight seals and a fan or two inside to mix the air. A hole (≈6×6 cm) can be cut in one wall for the Peltier module; the Peltier is sandwiched between heat sinks with its "cold" side facing inside and "hot" side outside, and the gap sealed (e.g. silicone) [1]. All wiring (power and signals) passes through the enclosure to the control board. Other HVAC variables (air flow, external temperature) are kept constant by insulation and minimal leaks.

Sensors (Temperature & Humidity)



We recommend digital T/H sensors that provide ready-to-use readings. For example, the Bosch **BME280** module (I²C interface) measures humidity and temperature (and pressure) with high accuracy. It is "high linearity" and "high accuracy" [2] and the humidity accuracy is about ±3% RH [3]. The BME280 can be wired to the ESP32 over I²C: Vin→3.3 V, GND→GND, SCL→GPIO22, SDA→GPIO21 (default I²C pins) [4]. In a simpler budget setup, a DHT22/AM2302 sensor can be used: it is low-cost, 3–5 V powered, with ~2–5% RH accuracy [5] (and ~0.5 °C temperature accuracy). Both types output a digital signal (I²C or 1-wire) that the ESP32 can read. The sensor(s) should be placed inside the chamber (e.g. away from walls) to accurately track the air conditions.

Actuators (Heating/Cooling and Humidity)



For temperature control, a **Peltier (TEC) module** is convenient in a small chamber. By driving current one way it cools the chamber, and by reversing polarity it heats. Mount the Peltier in the wall hole: attach one heat sink (with fan) outside and one inside, sealed against the box [1]. The ESP32 (via a MOSFET driver or H-bridge) can supply PWM power to the Peltier; relays or an H-bridge are used to reverse polarity for heating [6]. A PWM-controlled heat pump module and fan arrangement (as in DIY designs [6]) allows fine control of temperature.

For humidity control, an ultrasonic **mist-maker/humidifier module** (5–12 V, ~2 W) can add moisture on demand [7]. For example, inexpensive ultrasonic discs produce fog when driven with ~5 V. To lower humidity, one can simply stop misting and/or use a small **exhaust fan** to vent humid air, or let a cold surface condense moisture. In a fuzzy-control scheme, one controller adjusts the humidifier power and an exhaust/vent fan to maintain target RH [8]. (A classic design uses one fuzzy loop for temperature (controlling heaters/fans) and another for humidity (controlling humidifier/fan) [8].)

Control & wiring: The Peltier and fans (and humidifier) are driven by the ESP32's GPIOs through driver modules (MOSFETs or relays). For example, one MOSFET can switch the Peltier current (with PWM), and two relays can swap its polarity [6]. The humidifier module is simply switched on/off by a GPIO with a MOSFET. Ensure the ESP32's ground is common with all modules, and use flyback diodes for inductive loads. In operation, the ESP32 reads the T/H sensors and applies the fuzzy logic (or heuristic) to set the actuator outputs each cycle.

Microcontroller & Control Logic

Use an **ESP32** development board as the central controller. The ESP32 is a low-cost MCU with integrated Wi-Fi/Bluetooth [9], plenty of GPIO/PWM channels, and 3.3 V I/O, making it ideal for IoT projects. It can run Arduino or MicroPython code to read the sensors and control the actuators. For example, Adafruit's BME280 or DHT libraries run on the ESP32. The control program implements the fuzzy logic (or PID) loops: e.g.

fuzzify the temperature error to set the Peltier drive, and fuzzify the humidity error to set the humidifier/fan. In software, these can use membership functions and rule sets (e.g. with the Python `skfuzzy` library on a connected PC, or coded in C++ on the ESP32).

The ESP32 also handles data communication. It can stream sensor readings (and power usage data) to a PC for analysis. One option is to send data over the USB-serial link to a Python script; another is to use Wi-Fi (e.g. POST data to a local server or MQTT broker). The Python side will run the LLM/fuzzy analysis. For LLM integration, the Python script can package recent T/H readings (and perhaps historical trends) into prompts for a large language model (via an API). The model's output could be interpreted as new setpoints or control adjustments. (For instance, Zhu *et al.* demonstrate converting LLM "chain-of-thought" reasoning into HVAC control commands [10].) In practice, one might use the LLM to periodically re-tune the fuzzy rules or predict occupancy effects, then update the ESP32's control parameters.

Power Consumption Measurement

To track efficiency, include a **power sensor** on the actuator supply. If the system runs on DC (e.g. a 12 V supply for Peltier and fans), an INA219 high-side current/power sensor can be used. The INA219 breakout monitors bus voltage and current (up to ~26 V, 3.2 A) [11]. Wire the INA219 in series with the 12 V line feeding the Peltier or humidifier. The ESP32 reads INA219 over I^2C (same bus as the T/H sensor). It can compute power = V*I and log it over time. If any component runs on AC mains (e.g. a 230 V heater), an AC wattmeter or clamp sensor would be needed instead. These measurements allow the Python analysis to quantify energy use and optimize efficiency (e.g. minimizing power for the same comfort).

Data Acquisition and Software Integration

In operation, the ESP32 continuously reads T/H data and actuator outputs. It can **log or transmit** this data to a PC or cloud. For example, the ESP32 can run a simple web server (serving JSON) or push MQTT messages; a Python script can subscribe and record the stream. Alternatively, the ESP32 can print values to serial, and a Python program (using pySerial) captures them. Once in Python, the data feeds the fuzzy controller and any LLM analysis. The fuzzy controller computes new duty cycles or on/off signals each time step. The LLM (if used) might run on a slower loop: e.g. every few minutes it reviews recent data, then suggests tweaks (these can be sent back to the ESP32 via a network command or by updating parameters). The result is a closed-loop adaptive system: sensor → ESP32 → actuators, with Python/LLM in the loop for optimization.

Recommended Hardware Components

- **Temperature/Humidity sensor** – Bosch BME280 (I^2C, ±3% RH accuracy [3]) *or* DHT22/AM2302 (digital, ±2–5% RH [5]). These plug directly into the ESP32.
- **Microcontroller** – ESP32 dev board (Wi-Fi/Bluetooth MCU) [9].
- **Thermoelectric (Peltier) module** – e.g. 40×40 mm TEC with heat sinks/fans. Drive via PWM and polarity reversal [6].
- **Humidifier** – Ultrasonic mist-maker (5 V, ≈2 W) [7]. Optionally a small fan for dehumidifying.
- **Air circulation fans** – Small 5–12 V fans to mix chamber air and cool heat sinks.
- **Power sensor** – INA219 I^2C breakout (up to 26 V/3.2 A) [11] for DC power; or an AC power meter if needed.

- **Driver modules** – MOSFET drivers or relay board for switching Peltier and humidifier on/off.
- **Enclosure** – Insulated box (~1 ft^2) with openings for sensors, Peltier, humidifier. (Example: Styrofoam box or acrylic chamber.)

Each component is readily available within a ~Rs.50,000 budget. Total hardware cost is modest (~Rs.5–10k) well under the budget, leaving room for sensors and polish.

References: Standard designs and datasheets were used to select components. For example, Bosch's BME280 datasheet highlights its "high accuracy" humidity sensor [2] [3]. Instructable guides demonstrate controlling a Peltier with PWM and reversing polarity for heating [6], and placing the Peltier in a chamber wall with sealed heat sinks [1]. Similarly, Arduino projects use ultrasonic mist modules for humidity [7]. Fuzzy HVAC control strategies (with a humidifier+fan loop) are described in the literature [8], and recent research has explored using LLMs for high-level HVAC decision-making [10]. These guides inform the prototype design above.

---

[1] [6] DIY Temperature Controlled Chamber Box With Peltier TEC Module : 4 Steps (with Pictures) - Instructables
https://www.instructables.com/DIY-Temperature-Controlled-Chamber-Box-With-Peltie/

[2] [3] Humidity Sensor BME280 | Bosch Sensortec
https://www.bosch-sensortec.com/products/environmental-sensors/humidity-sensors-bme280/

[4] ESP32 with BME280 using Arduino IDE (Pressure, Temperature, Humidity) | Random Nerd Tutorials
https://randomnerdtutorials.com/esp32-bme280-arduino-ide-pressure-temperature-humidity/

[5] Overview | DHT11, DHT22 and AM2302 Sensors | Adafruit Learning System
https://learn.adafruit.com/dht/overview

[7] Smart Humidifier (make Your Room Comfortable) : 4 Steps (with Pictures) - Instructables
https://www.instructables.com/Smart-Humidifier-make-Your-Room-Comfortable/

[8] (PDF) Design of a room temperature and humidity controller using fuzzy logic
https://www.researchgate.net/publication/342120112_Design_of_a_room_temperature_and_humidity_controller_using_fuzzy_logic

[9] ESP32 Wi-Fi & Bluetooth SoC | Espressif Systems
https://www.espressif.com/en/products/socs/esp32

[10] Heating, Ventilation, and Air Conditioning (HVAC) Temperature and Humidity Control Optimization Based on Large Language Models (LLMs)
https://www.mdpi.com/1996-1073/18/7/1813

[11] INA219 High Side DC Current Sensor Breakout - 26V ±3.2A Max [STEMMA QT] : ID 904 : Adafruit Industries, Unique & fun DIY electronics and kits
https://www.adafruit.com/product/904