Lebanese University

Faculty of Engineering III

Mini Project

# AI-Controlled Energy Management and Overcurrent Protection System

Presented by:

| | |
|---|---|
| Fatima Al Fakih | 6251 |
| Roaa Darwish | 6147 |

Dr. Mohammad Aoude

## Contents

## 1. Introduction

In modern households and facilities, managing energy consumption is crucial not only for cost savings but also to prevent overloads that may trip the main switch or damage equipment. Our project presents an AI-controlled system that monitors power usage and intelligently disconnects low-priority circuits in the event of high consumption. The system is designed not for fault protection, but as a preemptive load management assistant.

## 2. Problem Statement

Traditional circuit breakers act only after the threshold is exceeded. There is a lack of dynamic systems that intelligently reduce load based on circuit importance and usage. Our system addresses this by:

- Monitoring individual circuits in real time.
- Classifying circuits by criticality and usage behavior.
- Automatically shutting off lower-priority circuits when total current exceeds a safe threshold.

## 3. System Overview

**Hardware Used:**

- Arduino Uno
- INA219 current and voltage sensors (4x)
- Relay modules (4-channel)
- NPN transistors and resistors
- Household load simulators (lamps, pumps, etc.)
- Serial communication (virtual COM link for Proteus simulation)

**Software Used:**

- Python 3
- SQLite database
- Scikit-learn (for circuit classification)
- Rule-based AI python agent (for real-time decision-making)

**Data Flow:**

1. INA219 sensors collect voltage and current data for each circuit.
2. Arduino sends readings to Python via serial link.
3. Python logs the data into the SQLite `sensor_readings` table.
4. Based on real-time current sum, the AI agent triggers OFF signals to Arduino for lower-priority circuits if needed.

**AI Agent**
The system uses both traditional AI techniques (rule-based and decision tree classification) and can be extended with generative AI for natural language interfaces or automatic reporting.

---

## 4. Key Functionalities

**1. Real-Time Data Logging:**

- Each circuit is turned ON individually for measurement.
- Data (voltage, current, power) is recorded and sent to Python.
- Python logs this into a database.

**2. Circuit Classification:**

- When a new circuit is added, it is monitored for 24 hours.
- Features such as average consumption, criticality, running time ratio, and temperature are used.
- A rule-based classifier(rule_based_classifier.py) assigns a priority and category.

**3. Intelligent Load Management:**

- A Python AI agent monitors total current.
- If it exceeds a predefined threshold (e.g., 10A), the system shuts down low-priority circuits.
- This helps prevent main breaker tripping while maintaining critical appliances.

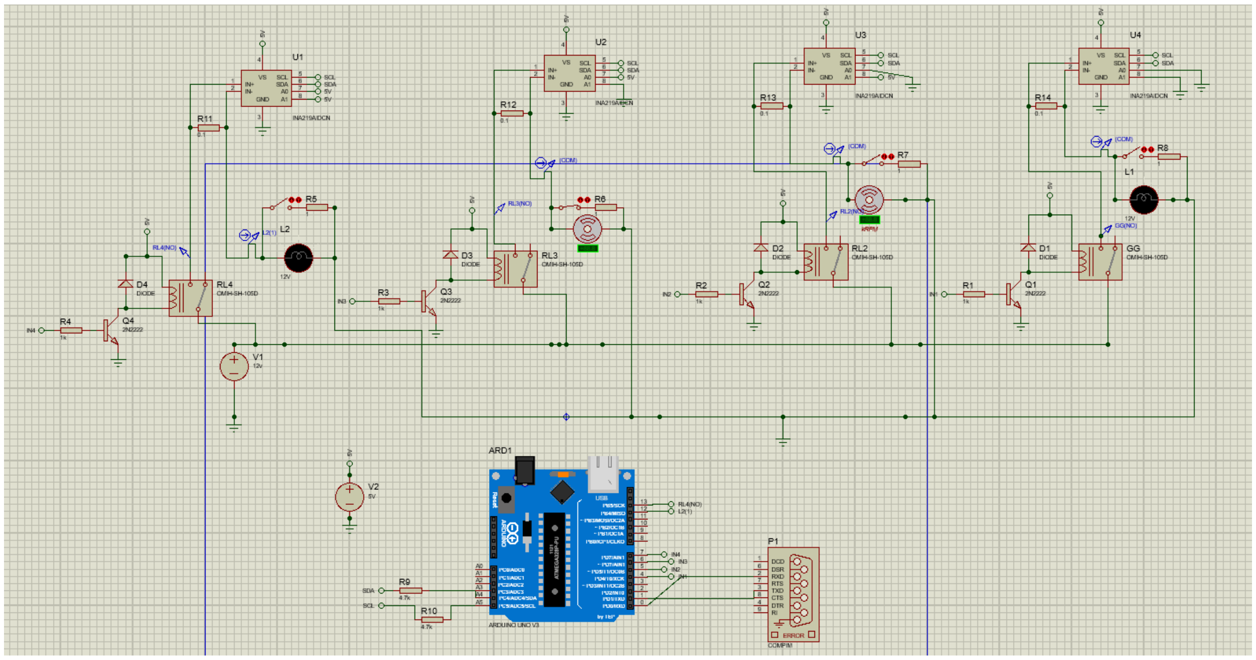
---

## 5. Circuit Diagram

The circuit diagram has 4 separate circuits connected to a voltage source.

DC has been used for simulation.

Shunt resistors of low values have been added to simulate high-current consumption controlled by manual switches.



## 6. Machine Learning Integration

A decision tree model was trained on generated circuit data to identify appliance types based on features such as:

- Priority (0 to 5)
- Criticality
- Running time ratio
- Temperature
- Average and Max consumption
- Peak operation time

This model adds an intelligent layer to recognize circuits and assist in classification automatically.

To train the model, we used database that has been collected by a UKJ research centre, from the article UK Domestic Appliance-Level Electricity (UK-DALE) dataset.

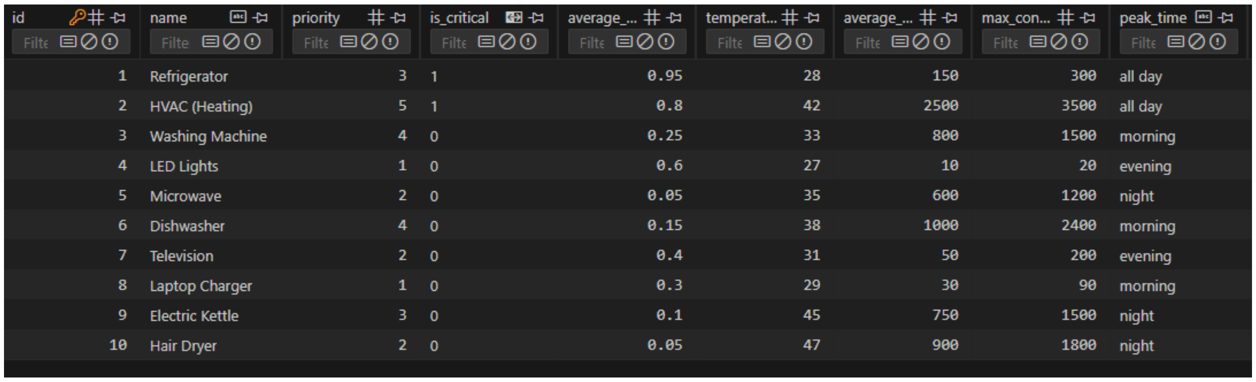
The dataset was not ready for us to use 100%, so we did some data cleaning and feature engineering to get the dataset suitable for our specific application.

# 7. Database

Using sqlite, we created 3 separate database tables.

**1. Circuits:** this table includes all the circuits connected in the system, and their corresponding features and ID's.

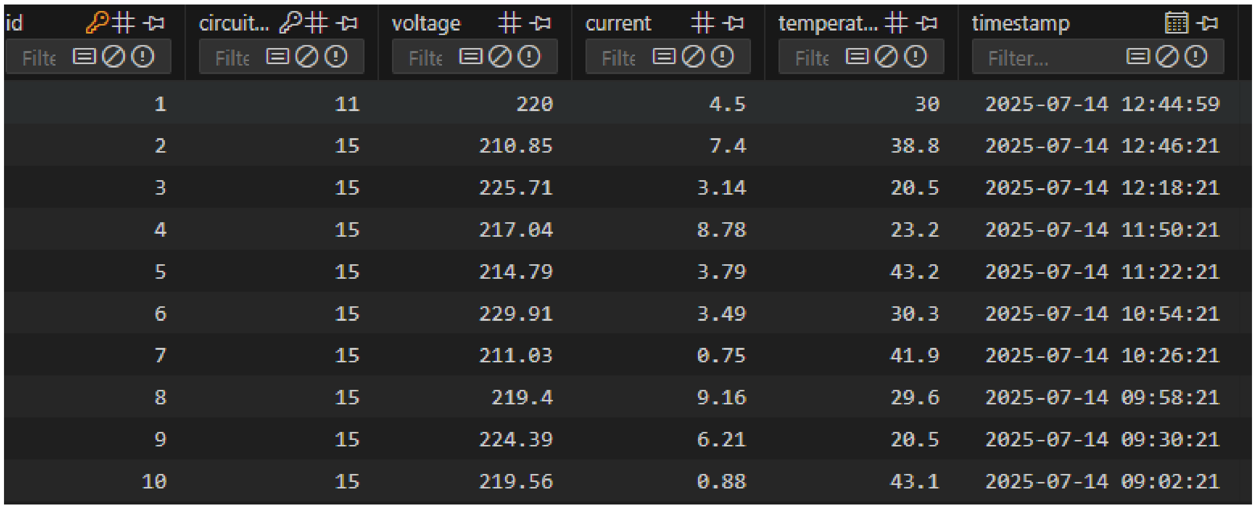
Below is a sample of this table:



| id | name | priority | is_critical | average_... | temperat... | average_... | max_con... | peak_time |
|---|---|---|---|---|---|---|---|---|
| 1 | Refrigerator | 3 | 1 | 0.95 | 28 | 150 | 300 | all day |
| 2 | HVAC (Heating) | 5 | 1 | 0.8 | 42 | 2500 | 3500 | all day |
| 3 | Washing Machine | 4 | 0 | 0.25 | 33 | 800 | 1500 | morning |
| 4 | LED Lights | 1 | 0 | 0.6 | 27 | 10 | 20 | evening |
| 5 | Microwave | 2 | 0 | 0.05 | 35 | 600 | 1200 | night |
| 6 | Dishwasher | 4 | 0 | 0.15 | 38 | 1000 | 2400 | morning |
| 7 | Television | 2 | 0 | 0.4 | 31 | 50 | 200 | evening |
| 8 | Laptop Charger | 1 | 0 | 0.3 | 29 | 30 | 90 | morning |
| 9 | Electric Kettle | 3 | 0 | 0.1 | 45 | 750 | 1500 | night |
| 10 | Hair Dryer | 2 | 0 | 0.05 | 47 | 900 | 1800 | night |

**2. Sensor Readings:** this table includes real-time logged data of sensors received from Arduino, each to its circuit ID and timestamp of reading

Below is a sample of this table:



| id | circuit... | voltage | current | temperat... | timestamp |
|---|---|---|---|---|---|
| 1 | 11 | 220 | 4.5 | 30 | 2025-07-14 12:44:59 |
| 2 | 15 | 210.85 | 7.4 | 38.8 | 2025-07-14 12:46:21 |
| 3 | 15 | 225.71 | 3.14 | 20.5 | 2025-07-14 12:18:21 |
| 4 | 15 | 217.04 | 8.78 | 23.2 | 2025-07-14 11:50:21 |
| 5 | 15 | 214.79 | 3.79 | 43.2 | 2025-07-14 11:22:21 |
| 6 | 15 | 229.91 | 3.49 | 30.3 | 2025-07-14 10:54:21 |
| 7 | 15 | 211.03 | 0.75 | 41.9 | 2025-07-14 10:26:21 |
| 8 | 15 | 219.4 | 9.16 | 29.6 | 2025-07-14 09:58:21 |
| 9 | 15 | 224.39 | 6.21 | 20.5 | 2025-07-14 09:30:21 |
| 10 | 15 | 219.56 | 0.88 | 43.1 | 2025-07-14 09:02:21 |

**3. Observation:** When new circuits are added, they are first put in this table for 24 hours with critical status and high priority, after that they are classified and sent to the circuits table.

## 8. Deployment with Uvicorn

To deploy the backend that communicates between the Arduino and the AI decision engine, we used FastAPI, a high-performance web framework for Python. FastAPI allows us to expose endpoints for sensor data logging, circuit classification, and control actions.

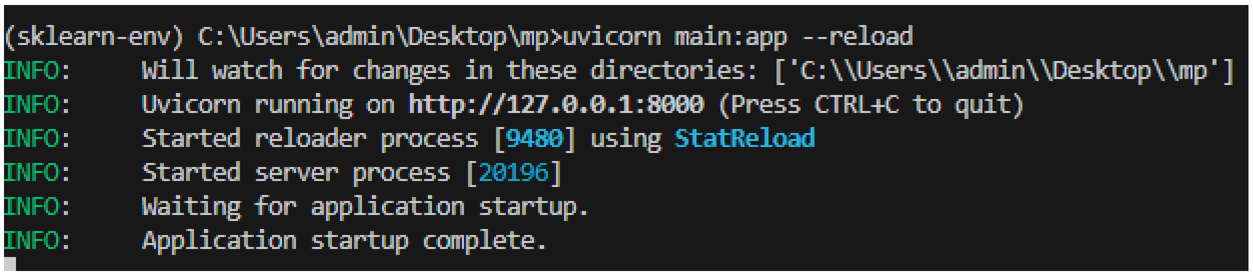
We used Uvicorn, a lightning-fast ASGI server, to run the FastAPI app. Uvicorn is responsible for serving the application, handling HTTP requests from external interfaces (like a future GUI), and running the backend asynchronously.

**Key reasons for using Uvicorn:**

**1. Speed:** It is built on uvloop and httptools, making it much faster than traditional WSGI servers.

**2. Async support:** Uvicorn allows asynchronous request handling, which is ideal for real-time operations like sensor reading and actuator control.

**3. Simplicity:** Running the backend is as simple as: uvicorn main:app –reload



```
(sklearn-env) C:\Users\admin\Desktop\mp>uvicorn main:app --reload
INFO:     Will watch for changes in these directories: ['C:\\Users\\admin\\Desktop\\mp']
INFO:     Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO:     Started reloader process [9480] using StatReload
INFO:     Started server process [20196]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
```

## 9. Python Virtual Environment

To ensure clean and consistent software development, we used a Python virtual environment for managing the required libraries and dependencies.

A virtual environment allows us to isolate our project’s dependencies from the system-wide Python packages. This is especially useful when different projects require different versions of the same packages.

So, a virtual environment was created, activated, and dependencies installed.

**Dependencies:** fastapi**,** Uvicorn, pyserial, scikit-learn, pandas, sqlite3

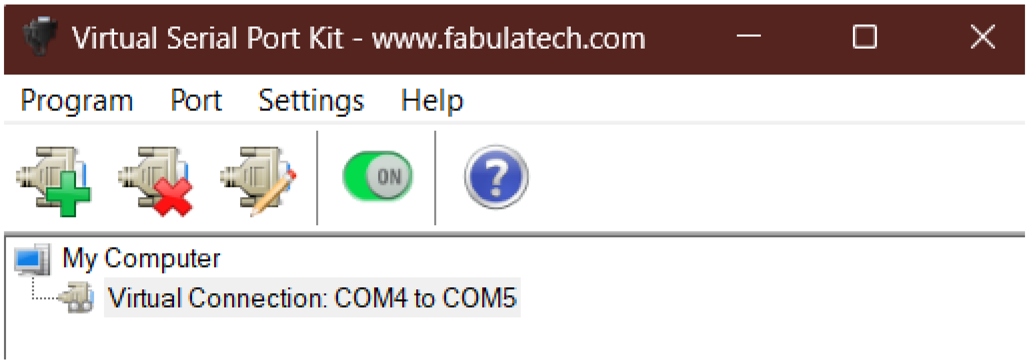
## 10. Connection of Proteus and Python

A virtual connection was established using:

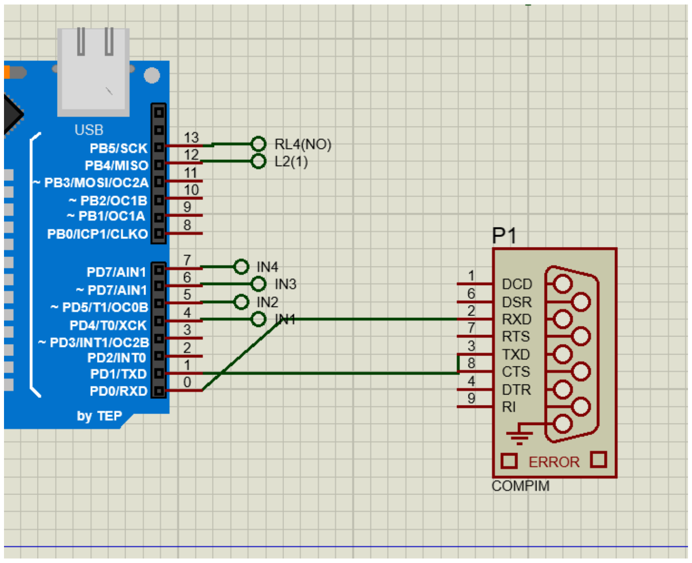
- COMPIM module in proteus
- Arduino
- Virtual Serial Port software
- Serial Library in python

The compim module was connected to rx and tx of the Arduino, and then it is linked to virtual COM4. The software creates a connection virtually between COM4 and virtual COM5, which was used by python.

Then all serial write and read commands were able to communicate between the two.



Virtual Serial Port Kit - www.fabulatech.com
Program
Port
Settings
Help
ON
My Computer
Virtual Connection: COM4 to COM5



# 11. Generative AI Assistant Integration

To extend the intelligence and usability of our AI-Controlled Energy Management and Overcurrent Protection System, we integrated a lightweight local Large Language Model (LLM) using Ollama. This enabled us to add natural language interaction to the system through a simple command-line assistant powered by the Gemma 2B model.

**Objective:**

- The assistant allows users to:
- Ask questions about the system's behavior (e.g., "Why was Circuit 4 turned off?")
- Get recommendations (e.g., "How can I reduce energy consumption today?")
- Understand logic behind the AI decisions
- Interact in plain English without needing technical knowledge

**Setup:**

We installed the Gemma:2b model via ollama run gemma:2b

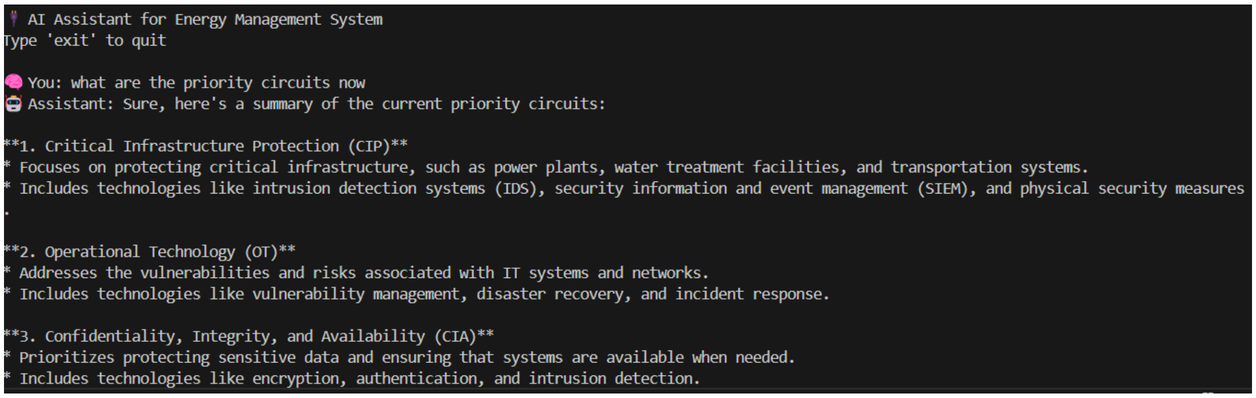
Used the ollama Python API to send and receive chat messages from the model

Created a CLI tool (ai_assistant.py) to let users interact via terminal

**Sample Interaction:**

You: What triggered the circuit shutdown?

Assistant: The system detected that total current exceeded the 10A threshold. Based on the rule-based logic, it deactivated low-priority non-critical circuits to prevent overload.



```
 AI Assistant for Energy Management System
Type 'exit' to quit

 You: what are the priority circuits now
 Assistant: Sure, here's a summary of the current priority circuits:

**1. Critical Infrastructure Protection (CIP)**
* Focuses on protecting critical infrastructure, such as power plants, water treatment facilities, and transportation systems.
* Includes technologies like intrusion detection systems (IDS), security information and event management (SIEM), and physical security measures
.

**2. Operational Technology (OT)**
* Addresses the vulnerabilities and risks associated with IT systems and networks.
* Includes technologies like vulnerability management, disaster recovery, and incident response.

**3. Confidentiality, Integrity, and Availability (CIA)**
* Prioritizes protecting sensitive data and ensuring that systems are available when needed.
* Includes technologies like encryption, authentication, and intrusion detection.
```

**Benefits**:

- Makes the system more user-friendly
- Demonstrates how generative AI can assist in explainability
- Opens doors for future expansion (e.g., voice interface, mobile app)

**Future Enhancement:**

In future versions, this assistant can be connected to the system's SQLite database, allowing it to:

- Answer queries about real-time sensor data
- Report status of individual circuits
- Log events or explanations on command

---

## 12. Challenges Faced

- Limited number of INA219 sensors; we simulated with 4.
- Arduino-Python communication required a virtual COM link due to Proteus.
- High current not passing through INA219 in simulation (we switched to probes and analog read methods).
- Lack of national data
- Absence of detailed information on domestic circuits

---

## 13. Future Work

- Build a GUI app that shows the live status of all devices and their priorities.
- Add mobile notifications and remote control.
- Implement energy-saving recommendations based on historical usage.
- Explore other sensors that can relate to more numbers
- Build a **natural language interface** for smart homes.
- Use **OpenAI's API** to generate reports

---

## 14. Conclusion

This project showcases a smart, AI-driven energy management system that ensures safer and more efficient energy usage. By predicting and reacting to consumption in real time, and classifying circuit importance, the system reduces unnecessary usage and avoids reaching overload conditions.

It is a stepping stone toward smarter homes that are not only automated, but also energy-aware and AI-assisted.