**AI Integration Layer – High-Level Overview**

This layer brings *intelligence* to electricity management system — moving beyond simple rule-based control.

**1. 🎯 Core AI Goals**

| **Goal** | **Description** |
| --- | --- |
| 🔍 **Monitor** | Continuously observe voltage, current, power |
| 📊 **Analyze Patterns** | Detect usage behavior (daily cycles, overuse, peak hours) |
| 🤖 **Predict** | Forecast demand or detect potential overload/failure |
| 🧠 **Recommend or Act** | Suggest shutdowns, optimize load timing, or auto-control relays |

**2. 📥 Data Required for AI**

| **Type of Data** | **Source** | **Purpose** |
| --- | --- | --- |
| Voltage/Current/Power | power\_log.csv | Core features for ML models |
| Relay States | From control layer | Know when devices are ON/OFF |
| Time of Day | System time | Learn daily/weekly usage patterns |
| User Overrides | Future UI logs | To learn what user prefers |

**3. 🔁 How Will AI Be Integrated?**

You have two main options:

**🔄 Real-Time Mode (on-device):**

* AI model runs on Raspberry Pi
* Uses live sensor data
* Makes instant relay decisions
* Good for remote/off-grid sites

**🧠 Cloud AI Mode (optional later): (Future Phase)**

* Upload logs to cloud
* Train heavier models (e.g., deep learning)
* Receive optimized policy updates

**4. 🧪 AI Functions to Build**

| **Task** | **Method** |
| --- | --- |
| Load & clean data | Python (pandas, NumPy) |
| Train simple model (e.g., SVM) | Scikit-learn |
| Predict “shutdown” or “OK” | Binary classifier |
| Recommend next action (JSON) | AI → recommendations.json → Control Layer |

**5. 🛠️ Plan for AI Script**

We will create: ai\_predictor.py

* Trains on your historical log
* Generates recommendations
* Logs to file for control system

**🔍 Start with On-Device (Raspberry Pi) AI, then scale later**

| **Strategy** | **Why We Recommend It Now** |
| --- | --- |
| ✅ **On-Device AI** | - Works offline (rural/remote homes) - No need for internet - Fast and simple to start |
| 🌐 **Cloud AI (Optional)** | - Can come later for advanced forecasting - Needs stable connectivity & more cost |
| 🧠 **Hybrid in Future** | - Train heavy models in cloud, push results to Pi |

**🎯 Decision: Start with simple AI logic on Raspberry Pi**

We’ll:

* Run Python code on Pi to **analyze the last 10–30 minutes**
* Decide if a “shutdown”, “delay”, or “normal” tag is needed
* Write this into recommendations.json
* Control layer will **read this and apply decisions**

This gives us:

* Real AI power ✅
* Works without cloud ❌
* Easy to grow later 🌱

**✅**

🚀 Next Step:

1. Automate this to run every 60 seconds (cron or loop)
2. Expand the AI model (use ML libraries)
3. Add feedback from user override for learning
4. Build visualizations/dashboard from AI logs

**✅ Step 1: Automate ai\_predictor.py to run every 60 seconds**

we have two main options for automation on Raspberry Pi:

**Option A: 🌀 Python Loop (Good for testing)**

Add this to a new script called ai\_loop\_runner.py:

python

Copy

import time

import os

while True:

os.system("python3 ai\_predictor.py")

time.sleep(60) # Wait 60 seconds before next prediction

Just run:

bash

Copy

python3 ai\_loop\_runner.py

✅ Easy to stop with Ctrl + C  
✅ No extra setup  
🔁 Keeps looping

**Option B: 🕓 Cron Job (Best for long-term)**

1. Open crontab:

bash

Copy

crontab -e

1. Add this line to run ai\_predictor.py every minute:

bash

Copy

\* \* \* \* \* /usr/bin/python3 /home/pi/PowerMind/ai\_predictor.py

📌 Adjust the path based on where your script is stored.

✅ Runs even after reboot  
✅ No need to keep terminal open  
✅ Fully automatic in background

**🧠 Step 2: Expand the AI Model for Smarter Decisions**

We’ll gradually go from **basic logic → real machine learning**.

**✅ Current Logic (in ai\_predictor.py)**

* Uses **simple thresholds** (e.g., if power > 600 W → shutdown)
* No learning or pattern memory

**🎯 Goal of This Step:**

Move toward **true AI** that can:

* Learn **patterns** (daily usage, appliance habits)
* Detect **anomalies** (unexpected surges/drops)
* Make **predictions** (will we overload in next 15 mins?)
* Personalize recommendations (for each home/business)

**🧪 Let’s Begin with a Smarter Model:**

**✅ Plan for This Step:**

1. Use **historical data** from power\_log.csv
2. Add **feature engineering** (like hour of day, voltage changes)
3. Train a **simple machine learning model** (like Random Forest)
4. Output prediction: "ok", "standby", or "shutdown"
5. Update ai\_predictor.py to use this model

**📦 Required Files:**

* ai\_predictor\_advanced.py → new smarter script
* train\_ai\_model.py → to train your first model
* model.pkl → saved trained model (can grow later)

Next step:

* use train\_ai\_model.py with sample data simulation?
* Or wait until you have real logs from your hardware?

we can still test with dummy data — what do you prefer?

Your AI training pack is ready:

📂 use train\_ai\_model.py  
📦 use model.pkl (trained model)

**🧠 What This Does:**

* Simulates 500 training rows (voltage, current, power, hour)
* Labels behavior: ok, standby, or shutdown
* Trains a **Random Forest model**
* Saves it as model.pkl (you can load and use it in prediction script)

🎯

**Step 3: Collect Override Feedback**

This helps the AI **learn from user behavior** — especially when users **override AI suggestions** (e.g., AI says shutdown, but user turns ON).

**🎯 Why This Matters:**

| **Without Feedback** | **With Feedback** |
| --- | --- |
| AI can't tell when it's wrong | AI learns from real user decisions |
| Hard-coded decisions stay static | AI adapts to preferences or conditions |
| User may not trust AI recommendations | User input improves accuracy over time |

**🛠️ Implementation Plan**

**📁 Create a new file: override\_log.json**

* Every time user **manually switches relay ON/OFF**, log:
  + Timestamp
  + Which relay
  + Action (ON/OFF)
  + What AI recommended at that time
  + User reason (optional via UI or log)

json

Copy

[

{

"timestamp": "2025-07-10T14:05:22",

"relay": "relay\_1",

"action": "ON",

"ai\_decision": "shutdown",

"user\_reason": "needed lights"

}

]

**🔄 Control Layer (smart\_control.py) Changes:**

1. **When user triggers manual override**:
   * Read recommendations.json
   * Write override entry to override\_log.json
2. Later, AI can read override\_log.json for *personalized training*.

**🧠 Option 1: log\_override() Function Inside smart\_control.py**

**🏆 Best choice for full integration**

| **✅ Pros** | **❌ Cons** |
| --- | --- |
| Centralized – everything in 1 script | Slightly harder to debug alone |
| Can log automatically on every manual action | More risk if smart\_control fails |
| Keeps code tight and efficient | Needs proper testing |

**🧰 Option 2: override\_logger.py as Separate Script**

**✅ Best for modularity and testing**

| **✅ Pros** | **❌ Cons** |
| --- | --- |
| Easier to test/debug independently | You must call it separately |
| Great for future mobile or web UI integration | Not automatic unless linked in code |
| Safer to isolate override logic | Slight delay in syncing overrides |

**✅ Recommendation:**

If plan to:

* Keep things **simple and unified**: use log\_override() inside smart\_control.py
* Add a **mobile app, dashboard, or multiple user sources** later: use override\_logger.py

**🛠 How to Use Option 1:**

1. When the user overrides a relay (e.g., in your interface or CLI), call:

python

Copy

from override\_logger import log\_override

log\_override("relay\_1", "ON", "user wants fridge always ON")

1. This writes an entry like:

json

Copy

{

"timestamp": "2025-07-10T14:05:22",

"relay": "relay\_1",

"action": "ON",

"ai\_decision": "shutdown",

"user\_reason": "needed lights"

}

1. AI can use this later to train better behavior for each relay/user.

* Add a test script to simulate a few override logs?

**What It Does:**

* Simulates 3 different manual relay overrides
* Calls log\_override() with relay name, action (ON/OFF), and reason
* Adds entries to override\_log.json (auto-created if not found)

📊 **Step 4: AI log visualization dashboard** — to see decisions, overrides, and power history graphically?

**What It Does:**

1. **Plots power data** from power\_log.csv
   * Saves chart as power\_log\_plot.png
2. **Displays latest AI recommendation**
   * Reads from recommendations.json
3. **Shows last 5 manual overrides**
   * Reads from override\_log.json

Make sure you have:

* power\_log.csv with timestamp and power column
* recommendations.json generated
* override\_log.json from override logs

Bottom of Form