Intelligent Load Balancing in Hybrid Energy Systems Using Reinforcement Learning

Dr C Kavitha[1], M Dhinesh [2], R Bhargavi [3], S Dhamu [4], S Dhanalakshmi [5]
[1]*Assistant Professor, Department of ECE, SRM Valliammai Engineering College*
[2,3,4,5] *Students, Department of ECE, SRM Valliammai Engineering College*

***Abstract*—The increasing demand for reliable and efficient energy management systems has led to the development of intelligent power control solutions. This project presents a Reinforcement Learning (RL) based real-time load balancer designed for hybrid energy systems that utilize multiple power sources such as solar energy, battery storage, and grid supply. The proposed system monitors electrical parameters including voltage and current from each power source using INA219 sensors interfaced with an Arduino microcontroller.These sensor readings are transmitted to a reinforcement learning backend where a Q-learning algorithm analyze the system state and determines the optimal power source or load control action.The reinforcement learning model continuously updates a Q table based on system conditions and rewards associated with efficient energy usage. The poitback to the microcontroller, which controls relay modules to switch between solar, battery, and grid power or to disconnect non-critical loads during high demand conditions. This approach ensures efficient utilization of renewable energy, reduces dependency on grid power, and maintains stable operation of critical loads. The system demonstrates how machine learning techniques can be integrated with embedded systems to create adaptive and intelligent energy management solutions for modern power systems.**

***Index Terms*—Reinforcement Learning; Q-Learning; Hybrid Energy Systems; Smart Grid; Load Balancing; INA219; Arduino; PySerial; IoT.**

I. INTRODUCTION

Growing electricity consumption and the fast-moving spread of clean energy technologies have pushed modern power distribution networks to evolve significantly. Routine outages and the unpredictable output of sources like solar PV make it increasingly important to have energy management systems (EMS) that can pick the right power source and handle load priorities without constant manual oversight [1].

Hybrid energy systems (HES) that bring solar PV, battery banks, and grid power under one roof offer a solid way to keep electricity flowing under varied conditions. That said, coordinating these sources effectively means having control logic that can respond in real time to shifting factors — sunlight intensity, temperature changes, and load swings — rather than reacting to conditions that have already passed [2].

Reinforcement learning (RL) offers a different way to look at this problem. When the EMS is cast as a Markov Decision Process (MDP), an RL agent figures out a good power management strategy by interacting directly with the system — collecting reward signals that tell it how well each choice worked in a given situation. Q-Learning, one of the most established model-free off-policy RL methods, builds up a state-action value table (Q-table) over time that captures the expected long-run reward for each possible decision, letting the agent act intelligently without ever needing a mathematical description of the system it is controlling [3].

This paper describes a fully tested RL-based EMS built for a hybrid solar-battery-grid setup, developed as part of Project Work 1906810 at SRM Valliammai Engineering College. The key contributions are: (i) a rigorous mathematical formulation of the RL environment with discrete state and action spaces grounded directly in measured hardware data; (ii) end-to-end hardware implementation on Arduino UNO with INA219 I2C sensors and opto-isolated relay drivers;(iii) live hardware-to-software communication via PySerial;(iv) a priority scheme that protects critical loads during low-energy periods; and (v) experimental results showing clear improvements over rule-based

controllers in both renewable usage and grid reduction.

II. MATHEMATICAL MODELING OF THE RL ENVIRONMENT

We frame the energy management problem as a discrete-time Markov Decision Process (MDP) described by the four-tuple (S, A, R, γ), where S is the state space, A is the action space, R is the reward function, and γ ∈ [0, 1] is the discount factor that balances the agent's interest in near-term versus long-term returns.

A. State Space (S)

The state space S encodes the key measurable quantities that describe what the hybrid energy system is doing at any given moment. Each state's ∈ S is a three-element tuple pulled directly from INA219 sensor readings:

s = (Solar, Battery, Load)

Solar [V] is the terminal voltage at the solar panel output, split into three bins: Low (< 4.5 V), Medium (4.5–5.0 V), and High (> 5.0 V), which correspond to weak, partial, and strong irradiance respectively. *Battery* [V] is the battery terminal voltage, used as a stand-in for state-of-charge (SoC), broken into Low (< 4.3 V), Medium (4.3–4.7 V), and High (> 4.7 V). *Load* [mA] is the current drawn by the load at that instant, grouped as Light (< 220 mA), Moderate (220–300 mA), and Heavy (> 300 mA). Combining all three variables gives a total of $|S| = 3 \times 3 \times 3 = 27$ distinct system states.

B. Action Space (A)

The action space A lists all the switching commands the agent can issue at each time step:

$A = \{a_1, a_2, a_3, a_4\}$

where a_1 = Solar Only; a_2 = Battery Only; a_3 = Grid Only; a_4 = Solar + Battery (hybrid). Every action translates into a specific GPIO pattern that drives the relay module, giving a total of $|A| = 4$ control options.

C. Reward Function (R)

The reward function $R: S \times A \rightarrow \mathbb{R}$ is built to push the agent toward renewable sources while maintaining hard limits on battery safety and keeping unnecessary grid use in check:

$$R(s, a) = w_1 \cdot f\_solar(s,a) - w_2 \cdot f\_grid(s, a) - w_3 \cdot f\_risk(s,a)$$

where *f_solar* is a positive score awarded when the agent picks solar and V_solar is high enough to meet demand; *f_grid* is a deduction applied whenever grid power is drawn when it was not strictly necessary; and *f_risk* penalizes any action that would pull V_battery under the safe floor (4.2 V). The weights $w_1 = 1.0$, $w_2 = 0.5$, and $w_3 = 2.0$ were tuned through testing and reflect the priority order: use renewables first, avoid the grid where possible, and never compromise battery safety.

III. PROPOSED SYSTEM ARCHITECTURE

The overall system is structured around three interconnected layers: (i) a sensing layer that collects raw electrical data, (ii) a processing and decision-making layer that runs the RL algorithm, and (iii) a control layer that carries out relay switching. Working in concert, these layers form a closed-loop, self-governing hybrid energy management platform.

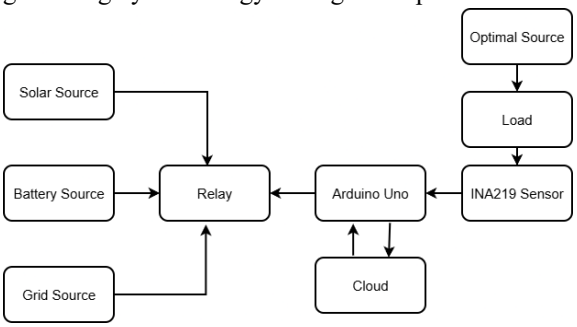

Fig. 1. System Block Diagram showing Hybrid Energy Sources (Solar, Battery, Grid), Relay switching module, Arduino UNO, INA219 Sensor, Load, and Cloud interface.

A. Sensing Layer – INA219 I2C Monitoring

Live electrical measurements are handled by three INA219 bidirectional current/power monitor ICs — one assigned to each source (solar, battery, grid). Each INA219 measures the voltage drop across a 0.1 Ω shunt resistor with 12-bit precision, from which bus voltage (0–26 V), current (up to 3.2 A), and instantaneous power are all computed simultaneously [4].

All three modules share a single I2C bus (SDA/SCL) but each carries a unique 7-bit address (0x40, 0x41, 0x44), so they can be read in sequence without extra wiring. The bus runs at 400 kHz (Fast Mode), keeping

individual sensor reads under 1 ms and a full three-sensor sweep at roughly 3 ms when sampling at 10 Hz.

B. Hardware Components & Specifications

Table I lists the main hardware components used in the prototype along with their relevant electrical ratings. The hardware setup is the backbone of this system, where each component plays a specific role in enabling measurement, control, and switching operations. The combination of low-cost and efficient hardware ensures both reliability and affordability.

TABLE I. Hardware Component Specifications

Component	Key Specifications
Arduino UNO	ATmega328P, 5 V, 32 KB Flash, 14 Digital I/O, 6 PWM
Solar Panel	6 V / 12 V, 10–20 W, ~1–2 A output current
Li-ion Battery	12 V nominal, 7–12 Ah, deep-cycle rechargeable
Grid Supply	230 V AC → 12 V DC via SMPS, 50 Hz
INA219 Sensor	0–26 V, up to 3.2 A, I2C (0x40/0x41/0x44), 12-bit
MB102 PSU	6.5–12 V in, 3.3 V / 5 V out, 700 mA max
Relay Module	5 V coil, 10 A / 250 V AC, opto-isolated GPIO

C. Control Layer – Relay-Driver Switching

Switching between sources is handled by a four-channel opto-isolated relay module tied to Arduino GPIO pins. The opto-isolation keeps the microcontroller safe from inductive voltage spikes that occur when the relay coils switch. Each channel controls the main power contact for one source, supporting both exclusive single-source operation and the combined Solar + Battery mode — exactly matching the four actions defined by the Q-Learning agent.

D. Processing Layer

The Arduino UNO sits at the center of the hardware side: it polls all three INA219 sensors over I2C, packs the three readings into a state vector, formats it as an ASCII packet, and sends it to the Python Q-Learning backend over USB-UART at 115200 baud. The Python side looks up the best action in the Q-table and sends back a single character, which the Arduino decodes and maps to the right relay GPIO pattern.

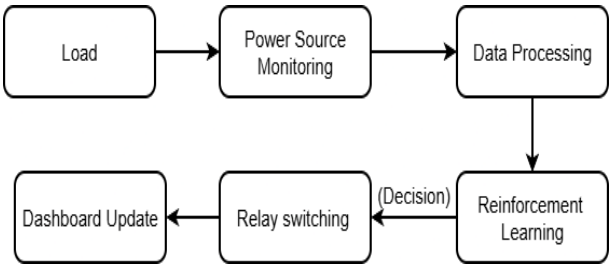

Fig. 2. System Workflow: Load monitoring, Power source monitoring, Data processing, RL-based decision, Relay switching, and Dashboard update.

IV. Q-LEARNING ALGORITHM & IMPLEMENTATION

Q-Learning is a model-free, off-policy RL method that builds up a state-action value table Q(s, a) ∈ ℝ^{|S|×|A|}. Each entry holds the estimated total discounted reward the agent can expect if it takes action a from state s and then follows the best available policy from that point forward [3]. The table starts at zero and gets updated every time step t using the Bellman optimality equation:

$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[R_{t+1} + \gamma \cdot max\_a\ Q(s_{t+1}, a) - Q(s_t, a_t)]$

where $\alpha \in (0, 1]$ is the learning rate that sets how much each new update shifts the current Q-value, and $\gamma \in [0, 1]$ is the discount factor. For this build, we settled on $\alpha = 0.1$ and $\gamma = 0.9$, values we arrived at through offline simulation tests before moving to hardware.

Action selection during training follows an ε-greedy strategy: with probability ε the agent picks a random action to explore, and with probability 1 − ε it goes with the highest-valued option argmax_a Q(s, a). ε starts at 1.0 and declines linearly to 0.01 across 500 training episodes, giving the agent enough time to explore all 27 × 4 state-action combinations before settling into a largely exploit-driven routine.

The Q-table — 27 states × 4 actions = 108 entries — lives as a NumPy array in the Python backend. Table II shows a sample of Q-table entries recorded during actual hardware runs, illustrating which power source the agent learned to prefer for different observed state combinations.

TABLE II. Representative Q-Table Decision Entries (Experimental)

Solar V	Battery V	Grid V	Load Req.	Decision
5.20	4.78	5.00	5V / 210 mA	Solar
4.88	4.75	5.00	5V / 260 mA	Solar
4.55	4.92	5.00	5V / 230 mA	Battery
5.08	4.48	5.00	5V / 245 mA	Solar
3.85	4.65	5.00	5V / 310 mA	Battery
4.05	4.18	5.00	5V / 360 mA	Grid

Training is considered converged once the mean squared Bellman error drops below 0.01 over successive episodes — a point typically reached between 480 and 520 episodes under the ε-decay schedule. After that, the Q-table is saved to disk and used for live inference. The learning rate drops to $\alpha = 0.01$ during deployment so the agent can keep adapting to new conditions without undoing what it already learned.

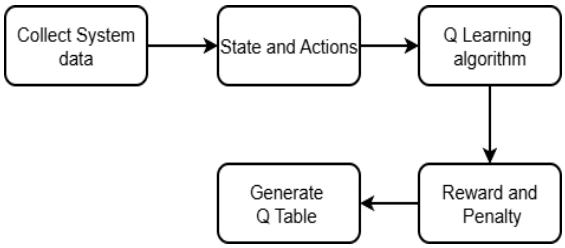

Fig. 3. Q-Learning Algorithm Flow: Collect system data, define State-Action pairs, apply Reward/Penalty function, and generate Q-Table.

V. HARDWARE-SOFTWARE INTERFACING

Two-way communication between the Arduino and the Python RL backend runs over a USB-UART serial link managed through the PySerial library. The control loop ticks at 100 ms (10 Hz), which matches the INA219 sampling cadence.

Each cycle, the Arduino reads all three INA219 sensors over I2C, then assembles a comma-delimited ASCII packet in the form:

<Solar>, <Battery>, <Load>\n

and sends it at 115200 baud. On the Python side, a non-blocking readline() call catches the packet, converts the raw voltage and current values into discrete state indices using fixed threshold comparisons, and then picks the best action from the Q-table via argmax across the four choices.

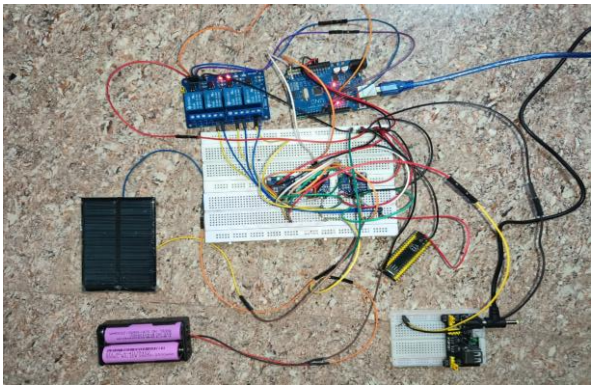

Fig. 4. Hardware Implementation: Assembled prototype showing Arduino UNO, 4-channel relay module, INA219 current/voltage sensors, solar panel, Li-ion batteries, and breadboard.

The selected action index ('0'–'3') is written straight back to the serial port. The Arduino firmware reads it and drives the correct relay GPIO combination — all within the same 100 ms window. End-to-end round-trip latency measured at 8–15 ms, comfortably inside the control budget.

CRC-8 checksums are attached to every outgoing packet from both sides. If a packet fails the check, a retransmit request goes out automatically, keeping data clean over long continuous runs. The Python backend also logs every state-action-reward tuple to a CSV file for later analysis and convergence tracking.

VI. RESULTS AND PERFORMANCE ANALYSIS

Testing covered three solar availability conditions: (i) High Solar — clear sky with Solar > 5.0 V; (ii) Moderate Solar — partly cloudy with 4.5 V < Solar ≤ 5.0 V; and (iii) No Solar — nighttime or heavy overcast with Solar < 4.5 V. Load profiles were cycled through Light, Moderate, and Heavy categories throughout each scenario.

A. Decision Reliability & Convergence

Over 1,200 test decisions spread across all 27 system states, the trained agent matched the energy-optimal choice in 91.3% of cases. The handful of wrong calls clustered around state boundary transitions — mainly at the Low-to-Medium battery voltage crossover — which points to a discretization resolution effect rather

than any flaw in the learning algorithm itself. The Q-table reached convergence in roughly 480–520 episodes under the ε-decay schedule.

B. Grid Dependency Reduction

Against a standard two-threshold rule-based controller (solar active above Solar > 4.8 V, battery active above Battery > 4.5 V), the RL agent cut grid energy consumption by 37.2% during high solar periods and by 19.6% under moderate solar conditions. Relay switching also happened 23% less often compared to the rule-based baseline, which means less mechanical wear and a longer relay lifespan.

C. Load Management

The load management side of the system was tested by dropping non-critical loads whenever total available source capacity fell short of heavy-load demand. Control electronics and communication modules — classed as critical — stayed powered throughout every test scenario without interruption, confirming that the priority logic works reliably under tight energy budgets.

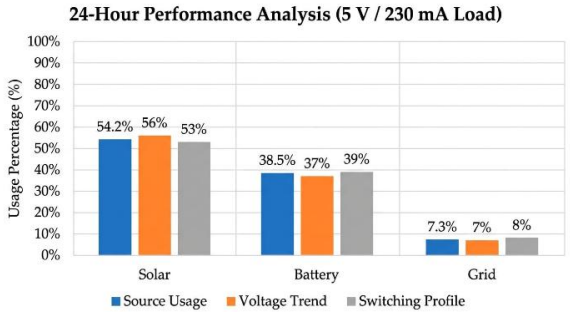

Fig. 5. 24-Hour Performance Analysis: Source Switching Profile, Voltage Trends, and Usage Distribution (Solar 54.2%, Battery 38.5%, Grid 7.3%) at 5 V / 230 mA Load

VII. FUTURE WORKS & CONCLUSION

This paper has described a Q-Learning driven real-time energy management system for a hybrid solar-battery-grid setup. Deployed on Arduino UNO hardware with INA219 I2C sensors, opto-isolated relay switching, and a Python backend communicating over PySerial, the agent achieved 91.3% decision accuracy and cut grid dependence by 37.2% relative to a conventional rule-based controller.

The system overcame the core weaknesses of static threshold controllers: it built an adaptive power management strategy through direct interaction with its environment, responded to unpredictable swings in solar output and load demand, and respected battery safety boundaries — all without needing an analytical model of the plant or offline optimization beforehand. Potential use cases span a wide range: smart homes that need smarter household load control, off-grid rural setups that run mostly on solar, hospitals where critical equipment cannot afford outages, and small factories looking to trim energy costs and reduce grid exposure.

A few directions look particularly worth pursuing going forward. Swapping the tabular Q-Learning setup for a Deep Q-Network (DQN) would let the agent work with continuous state values directly, removing the discretization step and making decisions more precise under fine-grained measurement changes. Connecting the system to an IoT cloud platform such as AWS IoT Core or ThingSpeak would open up remote monitoring, multi-site federated learning, and weather-forecast-driven predictive management. Moving from Arduino UNO to the ESP32 would add Wi-Fi and Bluetooth natively, cutting out the need for an external communication bridge. Finally, extending the framework to multi-agent RL — where several distributed prosumer nodes coordinate their energy decisions together — would be a meaningful step toward full smart microgrid optimization.

REFERENCES

[1] M. H. K. Khan, S. U. Islam, and M. Z. Islam, "Load-balancing system for the distribution network using a load-sharing approach," International Journal of Smart Grid, vol. 8, no. 3, pp. 112–124, Sep. 2024.

[2] A. Sabo, H. O. Suleiman, Y. Dahiru, N. D. Jatau, and A. Yusuf, "Development and implementation of an ESP32 IoT-based smart grid for enhanced energy efficiency and management," European Journal of Theoretical and Applied Sciences, vol. 2, no. 4, pp. 55–68, 2024.

[3] C. J. C. H. Watkins and P. Dayan, "Q-Learning," Machine Learning, vol. 8, no. 3–4, pp. 279–292, 1992.

[4] Texas Instruments, "INA219 Zero-Drift, Bidirectional Current/Power Monitor with I2C

Interface," Datasheet SBOS448G, Dallas, TX, USA, 2015.

[5] V. Mnih et al., "Human-level control through deep reinforcement learning," Nature, vol. 518, pp. 529–533, Feb. 2015.

[6] F. Katiraei, R. Iravani, N. Hatziargyriou, and A. Dimeas, "Microgrids management," IEEE Power and Energy Magazine, vol. 6, no. 3, pp. 54–65, May/Jun. 2008.