International Journal of Research in Engineering and Science (IJRES)
ISSN (Online): 2320-9364, ISSN (Print): 2320-9356
www.ijres.org Volume 14 Issue 11 ǁ November 2026 ǁ PP. 36-45

Memory Efficient Reinforcement Learning for Microcontroller Class Edge Devices: Near DQN Control at a Fraction of the Memory in Solar-Powered Street Lighting

Dan Dan-Abia[*1], Kingsley Udofia[2], Dr. Kufre Udofia[3]

[*1]*Department of Electrical/Electronics Engineering, University of Uyo, Uyo, Nigeria*
[2]*Department of Electrical/Electronics Engineering, University of Uyo, Uyo, Nigeria*
[3]*Department of Electrical/Electronics Engineering, University of Uyo, Uyo, Nigeria*
Corresponding Author: Dan Dan-Abia(dandan-abia.phd@uniuyo.edu.ng)

Abstract
Solar-powered street lights and similar smart city nodes must decide, minute by minute, how much to sense, which on-device model to run, and how brightly to light, all from a small battery and a modest panel, on a microcontroller with very little memory. Reinforcement learning (RL) can learn this trade-off, but most reported controllers say little about what the policy itself costs to store. This paper compares three RL controllers, tabular Q-learning, linear function approximation (Linear FA), and a deep Q-network (DQN), against five baseline policies in a Python simulation of a solar street-lighting node with a 100,000 J battery, a 100 mW peak solar panel, and a 0.2 to 1.6 W LED. The control problem is posed as a Markov decision process with a four-dimensional state and a 27-action joint control space covering TinyML model selection, sensing frequency, and LED brightness. Over 20 greedy evaluation episodes, the RL policies earned cumulative rewards of 298.7 to 349.6, against −355.6 to −822.7 for the baselines. DQN reached 94.9% accuracy and kept the node running for the full 1,440 minutes, whereas the Greedy baseline exhausted its battery at minute 741. Linear FA reached 94.7% accuracy with a policy of only 1.3 KB, compared with 14.9 KB for DQN and 63.3 KB for Q-learning, and earned a composite deployment score of 0.926 against 0.958 for DQN and 0.634 for Q-learning. The differences were significant at the 95% confidence level. For nodes of this class, a well-featured linear policy captures most of the benefit of deep RL at a small fraction of the footprint.

Keywords: *Adaptive control, deep Q-network, edge computing, energy harvesting, linear function approximation, reinforcement learning, smart street lighting, TinyML*

Date of Submission: 28-10-2026 Date of acceptance: 07-11-2026

I. INTRODUCTION

Street lights, air-quality monitors, and traffic counters are becoming standard parts of smart city infrastructure, and a growing share of them sit where mains power is unavailable or where replacing batteries on a schedule is not realistic [1], [2]. The usual answer is to pair the device with a battery and a small photovoltaic (PV) panel [3]. That arrangement solves one problem and creates another. Harvested energy arrives irregularly, demand for service changes through the day, and the battery is finite, so the node has to keep deciding how much it can afford to spend. The design target in this setting is energy-neutral operation, in which consumption over the long run does not exceed what is harvested [4].

Rule-based controllers, adaptive duty cycling, and energy-aware scheduling are the traditional tools for this job [5], [6]. They work well when the environment is understood, but thresholds tuned for one site and season are often poor choices elsewhere. Reinforcement learning (RL) offers another route. Instead of relying on hand-set thresholds, the controller learns a policy from its own interaction with the environment [7], and deep RL has shown how far that idea can go [8]. Several studies have already applied RL to harvesting-powered sensor nodes, mostly to set duty cycles or sampling rates [9]–[12].

What these studies seldom report is the cost of the controller itself. A node built around a microcontroller (MCU) in the Arm Cortex-M class typically has tens to a few hundred kilobytes of RAM, and the firmware, the sensing stack, and any on-device inference model all compete for that space [13], [14]. A policy that wins by a few percent in simulation but cannot be stored on the target device has little practical value. Published comparisons are also hard to read across papers, because the baselines, harvest traces, and metrics differ from one study to the next [15], [16]. For solar street lighting in particular, we found little published work that tests RL on a battery-

constrained node. The closest recent example is a deep-learning controller on a Jetson Nano that dims lights from video detections, which is a supervised perception pipeline and not a harvest-aware RL policy [17].

This paper addresses that gap with a controlled simulation study. A solar-powered street-lighting node is modeled as a Markov decision process (MDP) in which the controller jointly chooses a TinyML inference model, a sensing frequency, and an LED brightness level. Three RL controllers of increasing complexity, namely tabular Q-learning, linear function approximation (Linear FA), and a deep Q-network (DQN), are trained on the same environment and compared with five baseline policies. Memory footprint is treated as a result in its own right, next to control performance. The contributions are as follows: (i) a joint formulation of model selection, sensing, and lighting control with a 27-action space; (ii) a like-for-like comparison of three RL families against five baselines under identical conditions; (iii) a composite deployment score that puts control quality and memory cost on a single scale; and (iv) evidence that a linear policy over engineered features delivers close to DQN-level performance at a small fraction of the memory.

The rest of the paper is organized as follows. Section II reviews related work, Section III presents the system model, and Section IV describes the policies and baselines. Section V sets out the experimental protocol, Sections VI and VII report and discuss the results, and Sections VIII and IX cover the limitations of the study and the conclusions.

II. BACKGROUND AND RELATED WORK

2.1 Energy-Neutral Operation on Harvesting Nodes

A harvesting node is usually described by four parts: a harvester, an energy store, a load, and a controller. The load consists of sensing, computation, and communication and, in lighting applications, the lamp itself. Early work on harvesting sensor networks established the principle of energy-neutral operation and showed that a node can sustain a higher duty cycle when it adapts to predicted harvest than when it runs at a fixed rate [4], [5]. Prediction-based schemes, such as exponentially weighted averages of past harvest, remain common [4], [18], and real-time scheduling approaches can offer deadline guarantees under energy limits [6]. Solar street lighting makes the problem harder in one particular way. The load peaks at night, when harvest is zero, so the battery has to carry the day's energy forward, and a run of overcast days can drain it completely [3].

2.2 RL for Energy-Harvesting Power Management

RL has been used to manage harvesting nodes for more than a decade. Hsu et al. applied it to time-of-day power provisioning for sustainable sensor-node operation [9], and Shresthamali et al. used it to manage a solar-powered node within an energy budget [10]. Aoudia et al. proposed an actor-critic energy manager, RLMan, that aims at energy-neutral operation [11]. Fraternali et al. configured sensing parameters automatically with RL [12] and later showed that one policy can be shared among nodes with similar conditions, which reduces training effort in large deployments [19]. Most of these controllers act on one or two variables, typically the duty cycle or the sampling rate. The node studied here selects the inference model, the sensing rate, and the lamp level together. That widens the action space and couples decisions that are usually handled separately.

2.3 Edge-Deployable RL and TinyML

Running learned policies on microcontrollers has a literature of its own. Inference runtimes such as TensorFlow Lite Micro and optimized kernel libraries for Cortex-M processors have made small neural networks practical on such devices [13], [20]. Quantization, pruning, and distillation shrink models further [21]–[23], and memory movement, more than arithmetic, tends to dominate the energy cost of neural workloads [24], [25]. A lightweight RL energy manager for harvesting-powered wearables has been demonstrated on low-end microcontrollers [26]. These results show that compact deep policies are feasible. They do not tell a designer when a deep policy is worth its footprint, which is the question this study sets out to answer.

2.4 Smart Street Lighting and the Research Gap

Adaptive street lighting is a natural application for RL. The dimming level is a small discrete action, demand depends on ambient light and traffic, and solar harvesting ties daytime conditions to nighttime service [1]. Much of the recent work on adaptive lighting that we are aware of centers on perception. One example adjusts streetlight intensity from video-based detection on an embedded GPU to cut late-night energy use [17]. What is missing is a study that treats the lamp, the sensing stack, and the on-device model as one energy-constrained decision problem, compares several policy classes on equal terms, and reports what each policy costs to store. The experiments in Sections V and VI are designed to supply that comparison.

Table 1 places the present study among the closest published work. The comparison is deliberately qualitative: the studies use different harvest traces, battery models, and baselines, so their reported numbers cannot be set side by side.

Table 1: Closest Related Studies and the Present Work

Ref.	Application	Method	Objective
[9]	Harvesting sensor node	RL time-of-day power provisioning	Sustainable node operation
[10]	Solar harvesting sensor node	RL power management	Utility within the energy budget
[11]	Harvesting sensor network	Actor-critic energy manager (RLMan)	Energy-neutral operation
[12]	Harvesting sensor configuration	RL automatic configuration (ACES)	Sensing quality and uptime
[19]	Indoor solar sensors	RL sampling-rate configuration with shared policies	Maximize sensed data without depleting storage
[26]	Harvesting wearable IoT device	Lightweight RL energy manager (tinyMAN)	Device utilization under harvest and battery limits
[17]	Street lighting (non-RL)	Deep-learning intensity control on an embedded GPU	Reduce late-night energy use
This work	Solar street-lighting node	Q-learning, Linear FA, and DQN against five baselines	Joint model, sensing, and lamp control under a memory budget

III. SYSTEM MODEL AND PROBLEM FORMULATION

The controller follows the layered structure shown in Fig. 1. Layer 1 estimates the energy state of the node, Layer 2 turns raw readings into a compact feature vector, Layer 3 holds the learned or rule-based policy, and Layer 4 acts as a safety supervisor that protects the battery floor whatever action the policy proposes [27]. Layer 5, the fleet-level learning loop, is part of the wider architecture but is not exercised in this study, which concentrates on Layers 1 to 4 for a single node.

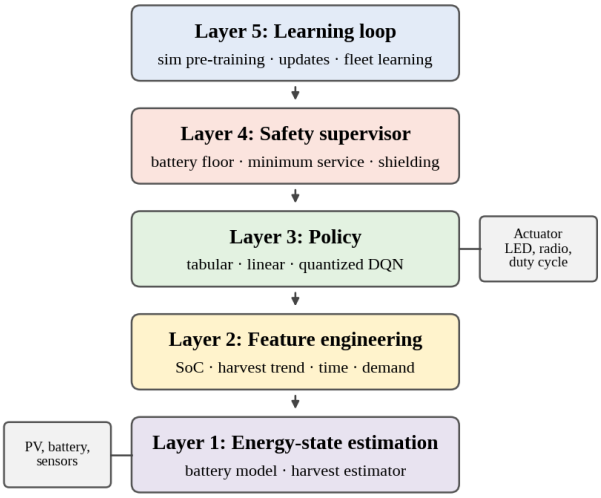


Figure 1: Layered control architecture for the solar street-lighting node.

3.1 Node and Environment Model

The simulated node has a PV panel, a battery, an LED luminaire with three brightness levels, a sensing front end, and an MCU that runs one of several TinyML models. Time advances in one-minute steps, and an episode covers 1,440 minutes, which is one full day. The panel has a peak output of 100 mW, and the harvested power follows the standard panel model

$$P_{\mathrm{h,t}} = \eta_{\mathrm{pv}} \cdot A \cdot G_t \cdot (1 - \xi_t) \tag{1}$$

where G_t is the solar irradiance, A is the panel area, η_{pv} is the conversion efficiency, and ξ_t is a loss factor that covers soiling, shading, and temperature effects. The battery stores at most $E_{\max} = 100{,}000$ J and evolves according to

$$E_{\mathrm{t+1}} = \min\{\, E_{\max}, \max\{\, 0, E_t + \eta_c P_{\mathrm{h,t}} \Delta t - P_{\mathrm{l,t}} \Delta t / \eta_d \,\} \,\} \tag{2}$$

where $P_{\mathrm{l,t}}$ is the total load power and η_c and η_d are the charge and discharge efficiencies. The load is the sum of the LED power, the sensing power, and the inference power of the selected model. The LED draws 0.2 W, 0.8 W, or 1.6 W, depending on the brightness level. The minute at which the battery is first exhausted is recorded as a measure of service continuity.

3.2 Markov Decision Process

The control problem is formulated as an MDP defined by a state space S, an action space A, transition dynamics P, a reward r, and a discount factor γ [7]. The agent looks for a policy that maximizes the expected discounted return. The state is four-dimensional and comprises the battery state of charge, the time of day, the solar harvest level, and an illumination-demand indicator. At each step the agent chooses one of 27 joint actions, and each action fixes three settings at once: the TinyML model used for inference, the sensing frequency, and the LED brightness. The reward balances the service delivered against the energy spent and penalizes battery depletion:

$$r_t = w_s s_t - w_p p_t - w_d \cdot \mathbb{1}[\, E_t \le E_{\min} \,] \tag{3}$$

Here s_t is the service delivered at step t (illumination adequacy and sensing accuracy), p_t is the energy spent, and the last term penalizes depletion below the battery floor $E_{\min}$. The weights have a large effect on what the agent learns. If the depletion penalty is too heavy, the agent holds the lamp at its lowest setting and delivers almost no service. If it is too light, the agent spends the battery early in the day and has nothing left for the night [15], [28].

3.3 Feature Engineering

Raw state variables are mapped to a vector of 12 features before they reach the policy. Time of day is strongly periodic, so it enters as a sine and cosine pair instead of a raw clock value, which avoids a false jump at midnight. Linear FA and DQN consume the 12-feature vector directly, whereas tabular Q-learning works on a discretized version of the state. Keeping the feature set small matters on a microcontroller, because its width fixes the number of weights, and therefore the memory, of every policy that uses it.

3.4 Safety Supervisor and Battery Floor

A learned policy can propose an action that looks attractive for the reward but drains the battery below a level the node cannot recover from. For that reason Layer 4 sits between the policy and the actuator. It checks every proposed action against the battery floor $E_{\min}$ and a minimum service requirement, and it overrides the action when either would be violated [27]. The supervisor keeps the comparison between policies fair, because the learned and rule-based controllers are bounded by the same safety envelope, and it matches how such controllers would be deployed in practice, where an unconstrained policy would not be accepted by an operator [28].

IV. CONTROL POLICIES AND BASELINES

4.1 Tabular Q-Learning

Tabular Q-learning stores one action value for every discretized state–action pair and updates it after each transition [29]:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[\, r + \gamma \max_{a'} Q(s',a') - Q(s,a) \,] \tag{4}$$

Inference is a table lookup, which is about as cheap as computation gets on a microcontroller. The cost is memory. The table holds one row per discretized state and one column for each of the 27 actions, so every additional state variable or finer grid multiplies its size, while a coarse grid saves space at the price of lost information [7].

4.2 Linear Function Approximation

Linear FA replaces the table with a weight vector over the 12 features. Each action has its own weight vector, so the action values are

$$\hat{Q}(s,a;\mathbf{w}) = \mathbf{w}_a^{\mathrm{T}}\, \boldsymbol{\varphi}(s) \tag{5}$$

and the weights are updated by semi-gradient Q-learning:

$$\mathbf{w}_a \leftarrow \mathbf{w}_a + \alpha[\, r + \gamma \max_{a'} \hat{Q}(s',a';\mathbf{w}) - \hat{Q}(s,a;\mathbf{w}) \,]\, \boldsymbol{\varphi}(s) \tag{6}$$

The memory needed is $27 \times 12 = 324$ weights, which at four bytes each is about 1.3 KB, and it does not depend on how finely the underlying state is resolved. A decision costs 27 dot products of length 12. Because the weights are shared across neighboring states, what the agent learns in one situation carries over to similar ones [7].

4.3 Deep Q-Network

The DQN approximates the action values with a small neural network that maps the 12-feature vector to 27 outputs. Training uses experience replay and a slowly updated target network for stability [8], and it minimizes the temporal-difference loss

$$L(\theta) = E[\ (\ r + \gamma\ \max_{a'}\ Q(s',a';\ \theta^{-}) - Q(s,a;\ \theta)\ )^2\ ] \quad (7)$$

where θ^{-} denotes the fixed target-network parameters. The network size, optimizer, and replay settings are listed in Table 3. All three RL policies are implemented in pure NumPy, and their footprints are compatible with an Arm Cortex-M4-class device.

4.4 Baseline Policies

Five baseline policies provide the reference point. They range from myopic rules to simple state-aware controllers of the kind used in practice. Rule-based dimming is still the industry norm, so beating it is the relevant test for any learned controller. Table 2 summarizes all the policies compared in this study.

Table 2: Control Policies Compared in This Study

Policy	Class	Stored representation / rule
Tabular Q-learning	RL, value-based	Q-table over the discretized state and 27 actions
Linear FA	RL, value-based	27 × 12 weight matrix over engineered features
DQN	RL, deep value-based	Neural-network weights (12 inputs, 27 outputs)
Greedy	Baseline	Myopic rule that favors immediate service and ignores future energy needs
Baselines 2 to 5	Baseline	for example a fixed-brightness rule, a state-of-charge threshold rule, a time-schedule rule, and a random policy]

V. EXPERIMENTAL SETUP AND EVALUATION METRICS

5.1 Simulation Environment

All experiments run in a Python simulator built on NumPy, with Matplotlib used for plotting. The environment implements the equations of Section III and applies identical dynamics to every policy, learned or rule-based. Table 3 lists the parameters. The battery holds 100,000 J, the panel peaks at 100 mW, and the LED draws 0.2, 0.8, or 1.6 W. An episode lasts 1,440 minutes. The RL agents were trained in this environment and then frozen before evaluation.

Table 3: Simulation and Training Parameters

Parameter	Symbol	Value
Battery capacity	E_{max}	100,000 J (about 27.8 Wh)
Solar panel peak output	—	100 mW
LED power levels	—	0.2 W, 0.8 W, 1.6 W
Episode length	—	1,440 min (24 h)
State dimension / engineered features	—	4 / 12
Action space	A	27 joint actions (TinyML model, sensing frequency, LED level)
Greedy evaluation episodes	—	20
Confidence level	—	95%

5.2 Evaluation Protocol

After training, each RL policy was run greedily, with exploration switched off, for 20 evaluation episodes. The baselines were evaluated under the same protocol. Repeated runs matter because deep RL results are known to be sensitive to seeds and hyperparameters, and a single run can mislead [16]. Results are reported over the 20 episodes, and differences are judged at the 95% confidence level.

5.3 Performance Metrics

Five quantities are reported. *Cumulative reward* is the sum of the step rewards over an episode. *Accuracy* is the average inference delivered by the selected TinyML models over the episode. *Service continuity* is the number of minutes the node operates before the battery is first exhausted, so 1,440 minutes means the full day was covered. *Memory footprint* is the storage taken by the policy parameters, in kilobytes: the Q-table for tabular Q-learning, the weight matrix for Linear FA, and the network weights for DQN. Finally, to place control quality and memory cost on one scale, we use a *composite deployment score*

$$S = \Sigma_k\ w_k\ \tilde{m}_k, \quad \Sigma_k\ w_k = 1 \quad (8)$$

where each $\tilde{m}_k$ is a normalized sub-metric in [0, 1] and w_k is its weight. A higher score is better, and a policy that is accurate but too large to store is penalized accordingly.

5.4 Implementation Notes

The simulator, the three learners, and the baselines share a single code base, so that every policy sees the same state construction, the same action set, and the same reward function. Evaluation episodes are generated from the same environment definition for every policy, and the learned policies are frozen during evaluation, so differences in outcome come from the policies and not from the conditions they faced. Policy footprints were measured on the stored parameter arrays after training, which is the quantity that has to fit on the device.

VI. RESULTS

6.1 Control Performance

All three RL controllers outperformed all five baselines on cumulative reward. Over the 20 evaluation episodes, the RL policies earned between 298.7 and 349.6, while the baselines finished between −355.6 and −822.7 (Table 4). Even the weakest RL result is 654.3 reward units above the best baseline, so the separation between the two groups is clear and does not depend on which RL algorithm is chosen. Within the RL group, DQN achieved the strongest performance.

Table 4: Cumulative Reward over 20 Greedy Evaluation Episodes

Policy group	Policies	Cumulative reward
RL controllers	Q-learning, Linear FA, DQN	298.7 to 349.6
Baseline policies	Five baselines	−355.6 to −822.7

6.2 Service Continuity and Energy Autonomy

The reward gap shows up directly in service. DQN kept the node running for the full 1,440-minute day. The Greedy baseline exhausted its battery at minute 741, roughly half way through the day, with 699 minutes still to go. This is the failure the learned controllers avoid. A policy that spends freely early in the day has nothing left when demand continues, and with a small panel and no grid backup the battery cannot recover.

6.3 Memory Footprint

The three policies differ in memory by a factor of almost 50 (Fig. 2a). Linear FA needs 1.3 KB, DQN needs 14.9 KB, and tabular Q-learning needs 63.3 KB. The Q-table is therefore roughly 50 times larger than the linear policy and more than four times larger than the DQN network, even though it is the simplest algorithm to implement. The DQN, in turn, is more than ten times larger than the linear policy.

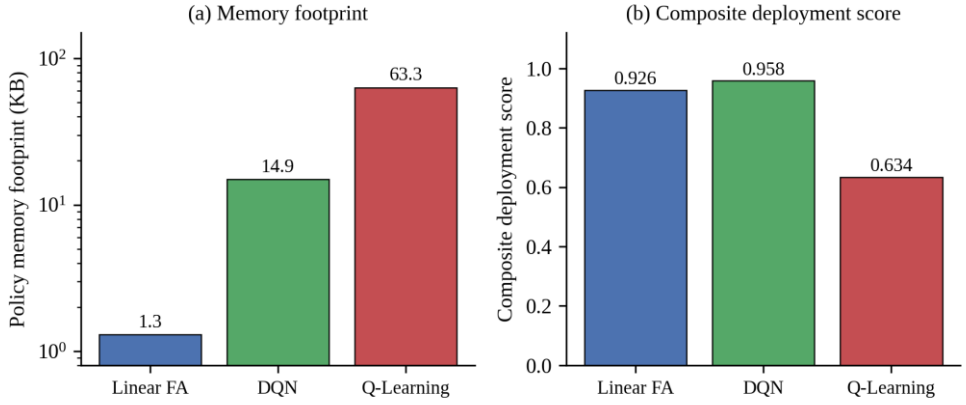


Figure 2: (a) Policy memory footprint (log scale) and (b) composite deployment score for the three RL policies.

6.4 Composite Deployment Score and the Accuracy–Memory Trade-off

The composite deployment score ranks DQN first at 0.958, Linear FA second at 0.926, and Q-learning last at 0.634 (Table 5 and Fig. 2b). Linear FA therefore keeps about 97% of the DQN score (0.926 against 0.958) and 99.8% of its accuracy (94.7% against 94.9%), while using about 8.5% of its memory. Plotting score against footprint makes the trade-off visible (Fig. 3). Linear FA sits at the left edge with almost the same height as DQN, while Q-learning is both the largest and the lowest.

Table 5: Deployment Comparison of the RL Policies

Policy	Memory (KB)	Accuracy (%)	Composite score
Linear FA	1.3	94.7	0.926
DQN	14.9	94.9	0.958
Tabular Q-learning	63.3	[value]	0.634

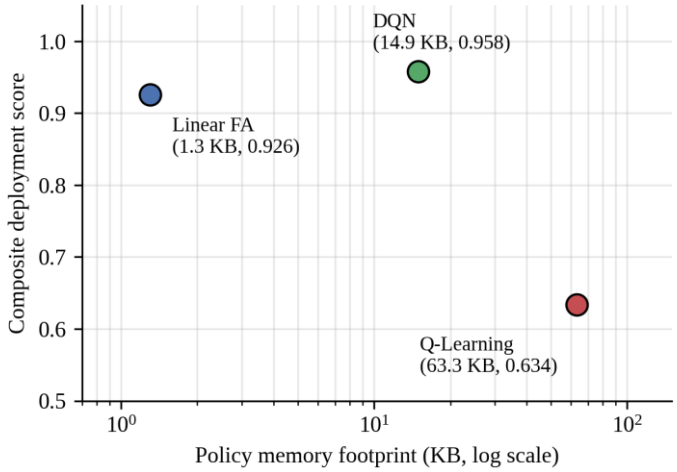


Figure 3: Composite deployment score against policy memory footprint.

6.5 Statistical Significance

The differences reported above were statistically significant at the 95% confidence level over the 20 evaluation episodes.

6.6 Summary of Findings

Table 6 collects the principal findings. Three of them bear directly on design. First, learning pays off: every RL controller separated cleanly from every baseline. Second, the most expensive policy is not the best choice by default, since the linear policy came within a few percent of DQN. Third, the simplest algorithm is not necessarily the cheapest to deploy, because the Q-table was the largest of the three.

Table 6: Summary of Principal Findings

Finding	Evidence
RL outperforms all baselines	Cumulative reward 298.7 to 349.6 for RL against −355.6 to −822.7 for the baselines
DQN gives the best service	94.9% accuracy and full 1,440-minute continuity
Greedy control fails within the day	Battery exhausted at minute 741 of 1,440
Linear FA is the most compact policy	1.3 KB, against 14.9 KB for DQN and 63.3 KB for Q-learning
Linear FA retains most of the DQN benefit	94.7% accuracy and a composite score of 0.926 against 0.958
Q-learning is the costliest to store	63.3 KB and a composite score of 0.634

VII. DISCUSSION

7.1 Why Linear FA Comes So Close to DQN

The closeness of the linear and deep policies deserves an explanation, even though our experiments do not isolate one. The state is only four-dimensional, and the 12 engineered features already encode the structure that matters, notably the daily cycle through the sine and cosine pair and the harvest trend. When the features carry that much information, the extra capacity of a neural network has little left to add. A related point is that neighboring states call for similar actions: a battery at 62% and one at 64% should be treated alike, and a linear model shares what it learns across such neighbors automatically [7]. The small gap that remains, 0.2 percentage points of accuracy and 0.032 of composite score, is the price of that simplicity.

7.2 The Cost of the Q-Table

Tabular Q-learning scored lowest on the composite measure. It still beat every baseline on reward, so the result should not be read as a failure to learn, and its footprint, the largest of the three, is the most obvious contributor. A table has to be fine enough to resolve battery level and time of day, and every extra resolution step multiplies the number of entries by the 27 actions. Coarser grids shrink the table but throw away information [7]. The method keeps its place on nodes with a small state space and a comfortable memory budget, where a plain lookup is attractive.

7.3 Choosing a Policy for a Given Node

The results suggest a simple rule of thumb, summarized in Table 7. The right policy depends less on peak performance, which is close for DQN and Linear FA, than on the resources the node has to spare.

Table 7: Policy Selection Guidance Based on the Results

Node situation	Suggested policy	Reason
Very small memory budget, or firmware already fills most of the RAM	Linear FA	1.3 KB footprint, about 97% of the DQN composite score
Moderate memory budget, service quality has priority	DQN	Highest accuracy (94.9%) and full-day continuity at 14.9 KB
Small, coarse state space and no feature engineering	Tabular Q-learning	Simple lookup, but the largest footprint here (63.3 KB)
Learned control must come with a safe fallback	Any RL policy plus a supervisor	All RL policies beat the baselines on reward; a supervisor protects the battery floor [27]

7.4 Decision Cost and Deployment Considerations

For a lamp that draws between 0.2 and 1.6 W, the energy used to evaluate a policy at each decision step is negligible next to the load, so the choice among policies is governed by memory and not by energy [24], [25]. For a microwatt-class sensing node that sleeps most of the time, the balance would be different, and the cost of inference and memory access would have to enter the reward. Smaller policies also leave more room for the rest of the firmware, which is often the binding constraint on low-end devices [13].

7.5 Implications for Reporting

On accuracy alone, Linear FA and DQN look almost interchangeable. It is the footprint that tells them apart, by more than a factor of ten. This argues for reporting memory next to reward in studies of RL for edge control, and for comparing learned policies with simple baselines under the same conditions [15]. A comparison that leaves out the cost of storing the policy risks favoring the largest model in every case. General-purpose TinyML benchmarks already report model size alongside accuracy [30], and energy-harvesting control would benefit from the same habit.

7.6 A Practical Path from Simulation to Street

The results suggest a staged route to deployment. The policy is first selected in simulation, then exported as a compact array or weight file and compiled into firmware. A Linear FA policy is a 27×12 matrix of single-precision numbers, which can be held in flash and evaluated with a few hundred multiply–accumulate operations per decision. The next step is hardware-in-the-loop testing, in which the policy runs on the target MCU while the environment stays simulated. After that comes a shadow period, in which the RL agent's recommended action is logged while the existing controller keeps control, followed by a gradual rollout to a subset of luminaires with a fallback to the incumbent rule. Updates to the stored policy should be signed and reversible, since a faulty update could darken many luminaires at once. Because the Linear FA policy is only 1.3 KB, the update itself is small enough to send over a constrained link.

7.7 Positioning Against Conventional Control

RL is not the only way to manage a harvesting node, and a fair assessment has to place it next to the alternatives (Table 8). Threshold rules are simple, transparent, and cheap to run, but their settings are tied to the site and season for which they were tuned. Prediction-based heuristics adapt the duty cycle to a forecast of harvest [4], and their quality depends on how good that forecast is. Model predictive control handles constraints explicitly and works well when the model and forecast are accurate, but it requires an optimization problem to be solved online, which is a heavy burden on an MCU [31]. Lazy scheduling offers real-time guarantees for task-based workloads [6].

Table 8: RL Controllers Compared with Conventional Control for Harvesting Nodes

Approach	Needs a model?	Adaptivity	Compute	Main weakness
Rule / threshold	No	Low	Very low	Fixed; needs per-site tuning
Prediction-based heuristic [4]	Harvest predictor	Medium	Low	Sensitive to forecast error
Model predictive control [31]	Yes	Medium (re-plans)	High	Model error; heavy online solve
Lazy scheduling [6]	Yes	Low to medium	Low	Task-model assumptions
RL, tabular and linear	Simulator for training	High	Very low	Training effort; table size (Q-learning)
RL, deep (DQN)	Simulator for training	High	Medium	Footprint; data hunger

The present results sharpen the last two rows of the table. Tabular and linear RL are both cheap to run, but they are not equally cheap to store: the Linear FA policy was the smallest of the three RL controllers by a wide margin, while the Q-table was the largest. Where a simulator is available and the memory budget is tight, a linear policy therefore combines the adaptivity of RL with a footprint that is hard for other approaches to beat. Where no trustworthy simulator exists, or where certifiable deterministic behavior is required, a rule-based controller with a safety margin may remain the better engineering choice.

VIII. LIMITATIONS AND THREATS TO VALIDITY

The study is a simulation, and its conclusions carry the usual caveats. No hardware-in-the-loop test has been run, so the memory figures describe policy storage and not the full RAM and flash usage, latency, or energy of a deployed firmware image.

Each episode lasts one day. Because the panel is small relative to the higher LED levels, a one-day episode mainly tests how well a controller budgets the stored energy. Multi-day low-irradiance spells and seasonal change, which decide whether a policy is truly energy-neutral [4], are not covered. The study also uses a single node at a single site, and 20 evaluation episodes, although adequate for the significance tests reported, cannot reflect every weather pattern.

The composite deployment score depends on the sub-metrics and weights chosen, and a different weighting could change the ranking of close competitors. The size of the Q-table depends on the discretization used and could be reduced with a coarser grid. Finally, the five baselines do not include optimization-based controllers such as model predictive control [31], which are heavier to run on an MCU but may be competitive where compute allows.

IX. CONCLUSION AND FUTURE WORK

This paper compared tabular Q-learning, Linear FA, and DQN with five baseline policies for the joint control of model selection, sensing, and lighting on a solar-powered street-lighting node. All three RL controllers beat every baseline on cumulative reward, with scores of 298.7 to 349.6 against −355.6 to −822.7. DQN performed best, with 94.9% accuracy and full 1,440-minute continuity, while the Greedy baseline ran out of energy at minute 741. The more practical finding concerns memory. Linear FA reached 94.7% accuracy with 1.3 KB, about 8.5% of the DQN footprint and a small fraction of the 63.3 KB needed by the Q-table, and kept roughly 97% of the DQN composite deployment score. For microcontroller-class nodes where memory is tight, a linear policy over well-chosen features is a strong default, and DQN is worth its extra footprint mainly where service quality has priority and memory is available.

Several steps would strengthen these results. The first is hardware-in-the-loop validation on a Cortex-M4 board to measure real RAM, flash, latency, and energy per decision. The second is multi-day and seasonal evaluation with measured irradiance data. A quantized DQN should also be tested, to see how much of the footprint gap closes with compression [21], [22], together with a sensitivity analysis of the reward weights. Beyond these, sharing policies across nodes with similar conditions [19] is a natural route to scaling the approach to a city-wide fleet.

REFERENCES

[1]. A. Zanella, N. Bui, A. Castellani, L. Vangelista, and M. Zorzi, “Internet of Things for smart cities,” *IEEE Internet Things J.*, vol. 1, no. 1, pp. 22–32, Feb. 2014.

[2]. W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, “Edge computing: Vision and challenges,” *IEEE Internet Things J.*, vol. 3, no. 5, pp. 637–646, Oct. 2016.

[3]. S. Sudevalayam and P. Kulkarni, “Energy harvesting sensor nodes: Survey and implications,” *IEEE Commun. Surveys Tuts.*, vol. 13, no. 3, pp. 443–461, 3rd Quart. 2011.

[4]. A. Kansal, J. Hsu, S. Zahedi, and M. B. Srivastava, "Power management in energy harvesting sensor networks," *ACM Trans. Embedded Comput. Syst.*, vol. 6, no. 4, Art. 32, 2007.

[5]. J. Hsu, S. Zahedi, A. Kansal, M. Srivastava, and V. Raghunathan, "Adaptive duty cycling for energy harvesting systems," in *Proc. Int. Symp. Low Power Electron. Design (ISLPED)*, 2006, pp. 180–185.

[6]. C. Moser, D. Brunelli, L. Thiele, and L. Benini, "Real-time scheduling for energy harvesting sensor nodes," *Real-Time Syst.*, vol. 37, no. 3, pp. 233–260, 2007.

[7]. R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. Cambridge, MA, USA: MIT Press, 2018.

[8]. V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[9]. R. C. Hsu, C.-T. Liu, and H.-L. Wang, "A reinforcement learning-based ToD provisioning dynamic power management for sustainable operation of energy harvesting wireless sensor node," *IEEE Trans. Emerg. Topics Comput.*, vol. 2, no. 2, pp. 181–191, Jun. 2014.

[10]. S. Shresthamali, M. Kondo, and H. Nakamura, "Adaptive power management in solar energy harvesting sensor node using reinforcement learning," *ACM Trans. Embedded Comput. Syst.*, vol. 16, no. 5s, Art. 181, 2017.

[11]. F. A. Aoudia, M. Gautier, and O. Berder, "RLMan: An energy manager based on reinforcement learning for energy harvesting wireless sensor networks," *IEEE Trans. Green Commun. Netw.*, vol. 2, no. 2, pp. 408–417, Jun. 2018.

[12]. F. Fraternali, B. Balaji, D. Sengupta, D. Hong, and R. K. Gupta, "ACES: Automatic configuration of energy harvesting sensors with reinforcement learning," *ACM Trans. Sens. Netw.*, vol. 16, no. 4, Art. 36, 2020.

[13]. R. David *et al.*, "TensorFlow Lite Micro: Embedded machine learning for TinyML systems," in *Proc. Mach. Learn. Syst. (MLSys)*, vol. 3, 2021, pp. 800–811.

[14]. P. Warden and D. Situnayake, *TinyML: Machine Learning with TensorFlow Lite on Arduino and Ultra-Low-Power Microcontrollers*. Sebastopol, CA, USA: O'Reilly Media, 2019.

[15]. G. Dulac-Arnold *et al.*, "Challenges of real-world reinforcement learning: Definitions, benchmarks and analysis," *Mach. Learn.*, vol. 110, no. 9, pp. 2419–2468, 2021.

[16]. P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep reinforcement learning that matters," in *Proc. AAAI Conf. Artif. Intell.*, 2018, pp. 3207–3214.

[17]. M. Asif *et al.*, "Adaptive control of streetlights using deep learning for the optimization of energy consumption during late hours," *Energies*, vol. 15, no. 17, Art. 6337, 2022.

[18]. J. R. Piorno, C. Bergonzini, D. Atienza, and T. S. Rosing, "Prediction and management in energy harvested wireless sensor nodes," in *Proc. Int. Conf. Wireless Commun., Veh. Technol., Inf. Theory Aerosp. Electron. Syst. Technol. (Wireless VITAE)*, 2009, pp. 6–10.

[19]. F. Fraternali *et al.*, "Scaling configuration of energy harvesting sensors with reinforcement learning," in *Proc. Int. Workshop Energy Harvesting Energy-Neutral Sens. Syst. (ENSsys)*, Shenzhen, China, Nov. 2018 (arXiv:1811.11259).

[20]. L. Lai, N. Suda, and V. Chandra, "CMSIS-NN: Efficient neural network kernels for Arm Cortex-M CPUs," arXiv:1801.06601, 2018.

[21]. S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," in *Proc. ICLR*, 2016.

[22]. B. Jacob *et al.*, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2018, pp. 2704–2713.

[23]. A. A. Rusu *et al.*, "Policy distillation," in *Proc. ICLR*, 2016.

[24]. V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proc. IEEE*, vol. 105, no. 12, pp. 2295–2329, Dec. 2017.

[25]. M. Horowitz, "Computing's energy problem (and what we can do about it)," in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, 2014, pp. 10–14.

[26]. U. Y. Ogras *et al.*, "tinyMAN: Lightweight energy manager using reinforcement learning for energy harvesting wearable IoT devices," in *Proc. tinyML Res. Symp.*, 2022.

[27]. M. Alshiekh, R. Bloem, R. Ehlers, B. Könighofer, S. Niekum, and U. Topcu, "Safe reinforcement learning via shielding," in *Proc. AAAI Conf. Artif. Intell.*, 2018, pp. 2669–2678.

[28]. J. García and F. Fernández, "A comprehensive survey on safe reinforcement learning," *J. Mach. Learn. Res.*, vol. 16, pp. 1437–1480, 2015.

[29]. C. J. C. H. Watkins and P. Dayan, "Q-learning," *Mach. Learn.*, vol. 8, no. 3–4, pp. 279–292, 1992.

[30]. C. Banbury *et al.*, "MLPerf Tiny benchmark," in *Proc. NeurIPS Datasets Benchmarks Track*, 2021.

[31]. E. F. Camacho and C. Bordons, *Model Predictive Control*, 2nd ed. London, U.K.: Springer, 2007.