Ember: Energy Management of Batteryless Event Detection Sensors with Deep Reinforcement Learning

Francesco Fraternali
UC San Diego
frfrater@ucsd.edu

Bharathan Balaji
Amazon
bhabalaj@amazon.com

Dhiman Sengupta
UC San Diego
dhimnsen@ucsd.edu

Dezhi Hong
UC San Diego
dehong@ucsd.edu

Rajesh K. Gupta
UC San Diego
gupta@ucsd.edu

ABSTRACT

Energy management can extend the lifetime of batteryless, energy-harvesting systems by judiciously utilizing the energy available. Duty cycling of such systems is especially challenging for event detection, as events arrive sporadically and energy availability is uncertain. If the node sleeps too much, it may miss important events; if it depletes energy too quickly, it will stop operating in low energy conditions and miss events. Ideally, the sensor should only turn on just before an event happens. We propose Ember, an energy management system based on deep reinforcement learning to duty cycle event-driven sensors in low energy conditions. We train a policy using historical real-world data traces of motion, temperature, humidity, pressure, and light events. The policy can learn to capture up to 95% of the events without depleting the node. As it may be difficult to obtain historical data for training a policy when deploying a node at a new location, we propose a self-supervised mechanism to collect ground-truth data while learning from the data at the same time. Ember learns to capture the majority of events within a week without any historical data and matches the performance of the policies trained with historical data in a few weeks. We deployed 40 nodes running Ember for indoor sensing and demonstrate that the learned policies generalize to real-world settings as well as outperform state-of-the-art techniques.

CCS CONCEPTS

• **Computer systems organization → Sensor networks**; • **Computing methodologies → Reinforcement learning**.

KEYWORDS

Batteryless, Event-Driven Sensing, Deep Reinforcement Learning

ACM Reference Format:
Francesco Fraternali, Bharathan Balaji, Dhiman Sengupta, Dezhi Hong, and Rajesh K. Gupta. 2020. Ember: Energy Management of Batteryless Event Detection Sensors with Deep Reinforcement Learning. In *The 18th ACM Conference on Embedded Networked Sensor Systems (SenSys '20), November 16–19, 2020, Virtual Event, Japan.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3384419.3430734

SenSys '20, November 16–19, 2020, Virtual Event, Japan

ACM ISBN 978-1-4503-7590-0/20/11…$15.00
https://doi.org/10.1145/3384419.3430734

1 INTRODUCTION

The Internet of Things market expects to reach over 75 billion connected devices by 2025 [1]. Many of these devices are battery-powered and require manual intervention to maintain working order[2], making it neither economical nor scalable. On the other hand, energy harvesting techniques avoid the use of batteries by powering the devices with energy gathered from the environment [7, 8, 30, 45], thus significantly mitigating the maintenance overhead. However, these solutions often come at the cost of intermittent operations due to sporadic energy availability impacting both application performance and system reliability as they stop working in low energy harvesting conditions.

Typical energy-harvesting applications [9, 27, 29, 41] fall into three categories based on their wake-up mechanism: (i) periodic, (ii) opportunistic (complete a task only when enough energy is gathered), and (iii) event-based (e.g., PIR motion, processing polling, etc). While some applications have been realized (e.g., periodical sensing of the environment at a reasonable rate [21]), the *event-driven* sensing scheme remains difficult to sustain using the batteryless energy-harvesting system. As an example, Campbell et al. can only detect 66% of door events due to low energy availability [9].

For a system to accomplish event-driven applications on batteryless energy-harvesting nodes, it needs to judiciously spare energy available by turning on the sensors *just before* the events happen. Therefore, the key challenge lies in learning an effective policy to decide when to power the sensors. Real-world events are not perfectly predictable, and thus the policy needs to trade-off between capturing events and depleting energy available. As the available energy and event patterns change across environments, we need to learn a data-driven policy that adapts to the environment. Recent works have proposed a reinforcement learning (RL)-based approach to learn such policies for energy-harvesting sensors [22, 46]. ACES [22] uses a Q-learning algorithm to duty cycle sensors, but they only learn the energy availability patterns in the environment and treat event sensors similar to periodic sensors. Therefore, their learned policy cannot adapt to event patterns in the environment. SpotON [46] also uses Q-learning to predict both events and energy availability. However, their tabular approach limits the number of inputs they can use for prediction and leads to a low detection accuracy of 19%. An additional challenge is that historical events data is required for learning a data-driven policy.

To address these issues, we design a system that maximizes events captured possible while avoiding the depletion of the sensor node's energy faced with unpredictable, sporadic energy availability. To this end, we present *Ember*, an energy management technique that uses deep RL to learn energy availability and event patterns in the environment, and adapt to the changing trends. RL-based methods adopt a *trial-and-error* approach: an agent learns to make decisions while interacting with the environment. At each step, the agent receives a reward informing how well it is performing on the task. Several properties make RL a good choice for solving our problem: (i) *learn directly from experience* – RL offers an alternative to heuristic-based approaches as it learns 'rules' that are specific to each environment. Studies show that RL agent is comparable and sometimes better than ad-hoc heuristics [48]; (ii) *delayed reward* – RL learns by maximizing cumulative reward in the long term: it can learn to sacrifice detection of a few events to spare energy for capturing more events later; (iii) *pattern recognition* – whenever present, deep RL can find repetitive patterns even under noisy environmental conditions, making it suitable for real-world tasks. When we use historical data to train the RL policy, our batteryless nodes can capture up to 95% of the events in an indoor setting, powered by harvesting energy from ambient light. However, the collection of reliable ground-truth event data itself requires at least temporary use of battery-powered sensor nodes, which increases the difficulty of deployment of energy-harvesting nodes. We propose a *self-supervised* (SS) data collection mechanism that facilitates deployment by opportunistically collecting ground-truth event data. Over time, Ember gathers enough ground-truth data to train performant RL policies. We show that using self-supervision, Ember learns to capture the majority of the events within a week without any historical data and matches the performance of policies trained with historical data in a few weeks. We evaluate Ember by deploying 40 batteryless sensing nodes in an office building. We compare our solution against state-of-the-art techniques that also exploit RL and show that Ember can improve event detection for batteryless energy-harvesting systems by 27% on average while not using historical data. In summary, we make the following contributions:

- We present a deep RL based sensor duty cycling system to detect a wide range of events—motion, pressure, temperature, humidity, light—using batteryless energy-harvesting sensors. Our solution addresses the challenge of predicting both energy availability and event occurrence probability. We formulate the Markov Decision Process for the problem of event detection using batteryless energy-harvesting sensors. We create a simulator that helps to train the policy quickly and deploy on real nodes.
- We develop a self-supervised data collection mechanism that discovers events over time and relieves the need for historical data. Our method helps discover new events and reconstruct the ground-truth data patterns. Therefore, it eases deployment by not using historical data as it works in conjunction with RL to incrementally improve performance over time.
- We report on real-world results from 40 batteryless sensor nodes deployed in a building and compare the performance against state-of-the-art techniques targeting batteryless sensing.

Ember is open-source with a permissive license[1].

[1]Code available at - https://github.com/francescofraternali/Ember.git

2 BACKGROUND

2.1 Related Work

Energy-harvesting Systems. In the last decade, numerous innovative energy-harvesting systems [66] have been introduced that enable applications ranging from energy metering [16] and opportunistic indoor event detection [9] to battery-free camera [52, 53] and wearable sensing [65]. Backscatter sensors eliminate communication related energy expenditure by leveraging existing RF signals for both energy harvesting and communications [19, 39, 42, 44, 75]. By gathering energy from the environment, these systems can power specific applications while avoiding battery replacement for long-term operation. They employ small energy storage (e.g. supercapacitor) to store just enough energy to fulfill a task (e.g. measure temperature). While they reduce space, cost, and design complexity, the system performance hinges on the sporadic and unpredictable energy availability. Therefore, these systems cannot detect events reliably. A few works have proposed to use larger energy storage that can power sensors for a day [21, 36, 46]. However, when the ambient energy is insufficient to fill the storage capacity, events will be missed. We focus on these low energy scenarios where active duty-cycling of the sensor is necessary to maximize event detection. With operability in low energy conditions, harvesting based sensors are deployable in a wider set of environments. Efforts also exist on energy storage management [28] and programming support [29, 73]. As energy-harvesting systems mature, intermittent computing emerges as a new computing paradigm [45], where works are focused on data consistency [15], timely operation [31], inference [27], event-driven execution [73], and service availability [47]. These works are complementary to our work.

Energy Management. To increase the lifetime and to reduce manual intervention, adaptive duty cycling of energy-harvesting sensors has been used to achieve energy-neutral operations [10, 24, 32, 38, 50, 62, 69]. Using estimated energy availability, nodes adjust their duty-cycle parameters to maximize application performance. However, these works use static rules that are not as effective data-driven approaches in learning event and energy patterns.

Several works have proposed machine learning for managing energy in energy-harvesting systems [4, 11, 12, 17, 22, 33–35, 68, 72]. Some of these are focused on sensor measurements such as temperature and maximize the number of measurements given uncertain energy availability [4, 11, 12, 22, 33, 34, 64], whereas our goal in this paper is to maximize event detection. In addition, a majority of these works focus on *one-time* learning assuming static environment [34, 35] (e.g., energy availability pattern does not change over time). In contrast, we seek a solution that can adapt to environmental changes over time.

On another front, there are studies that derive policies to maximize the information acquired [17] (e.g., avoiding sampling data when no changes are exhibited) or events detected [46], where they explore the use of Q-learning [71], which is a tabular RL algorithm. The tabular approach limits the number of inputs for prediction and leads to low detection accuracy. Murad et al. [51] demonstrate the use of deep RL to achieve maximal duty cycle while minimizing the variance of On time during each decision period, i.e., steady operation. Similar to a few others [4, 33, 34, 64], they only evaluate

Table 1: Comparison of Ember and state-of-the-art techniques for battery less event detection sensors.

Technique	General Apps Event-driven	No Intermittent Operations	Continuous Learning	Real-World Deployment	Hist Data Not Needed	Event Det [%] in Low Energy
Intermittent Comp [9, 19, 28, 53]			✓	✓		NA
ACES [22]		✓	✓	✓		59.6
SpotOn [46]	✓	✓				38.7
Ember	✓	✓	✓	✓	✓	76.2 - 95.3

the solution via simulations over synthetic data, and it is unclear how the solution would perform with real-world data.

We use a deep RL policy to judiciously turn sensors on for maximal event detection while avoiding energy storage depletion even in low energy conditions. Unlike prior works that use well-established simulators [14, 55], we create our own simulator that captures the physics of an energy-harvesting system. We demonstrate results in a real-world building and solve issues of deploying RL in practice. Additionally, a key innovation in our solution distinguishes itself from prior work—it bears a *self-supervised* data collection strategy that decides when to stay on for a short period of time to collect ground-truth data to enable deployment *without historical data*, and to *adapt to* changing energy and event patterns over time.

Self-Supervision. For Ember, we design a self-supervised data collection mechanism. Any method to automatically gather ground-truth data can be labelled as self-supervision. The technique initially emerged in the image domain [18, 25, 58], where images are translated, rotated or scaled to generate artificial labels, and learn a representation of the image using supervised learning techniques like convolutional neural networks [40]. The methods have been extended to video [70] and audio [54] domains. Self-supervision has also been used for RL to generate artificial rewards that incentivize exploration through curiosity [57] or diversity [20], and augment reward functions [63]. Self-supervised reward functions have been used for tasks such as grasping [74] and navigation [37]. In our problem setting, we need ground-truth events data to train an RL policy in simulation. We devise a self-supervised policy to switch between using the RL policy and collecting ground-truth data when there is sufficient energy to expend. Over time, the data collected is sufficient to capture event patterns, and the RL policy matches the performance of a sensor trained with historical data.

Table 1 summarizes the main differences between our work and current state-of-the-art techniques.

2.2 Proximal Policy Optimization

We use Proximal Policy Optimization (PPO) [61], a state-of-the-art policy gradient algorithm that has been widely used for applications such as manipulation [3], navigation [5, 49] and games [6]. We briefly describe how the algorithm works. In RL, an agent interacts with an environment to maximize its long term cumulative rewards. The agent chooses an action a according to its policy π given the current state s of the environment. Based on the action taken, the environment returns with the next state s' and reward r. We use the episodic setting, where the environment is initialized with state s_0, and terminates after T steps. PPO builds on the policy gradient algorithm [67], where the policy π is represented by a neural network with parameters θ. A randomly initialized policy interacts with the environment, and collects data in the form of (s, a, r, s'). The resulting data is used to train the neural network using gradient ascent with the following loss function:

$$L_\theta = \sum_{n=0}^{N} \sum_{t=0}^{T} \log \pi_\theta(a|s) \sum_{t=0}^{t} \gamma^{t-1} r_t. \tag{1}$$

Here $\gamma \in [0, 1]$, called discount factor, reduces the value of future rewards. $\pi_\theta(a|s)$ gives the probability of taking action a by policy π in state s. N is the number of episodes. Intuitively, the loss function guides the policy towards taking actions that improve the sum of discounted rewards. The updated policy π_θ is used to collect more data by interacting with the environment, and train the policy again. The training cycle continues until the policy converges. As each episode follows a different trajectory based on the actions taken, the discounted sum of rewards often have a high variance. Therefore, the policy gradient algorithm may not achieve the same performance each time we train a policy. To reduce variance, a bias is introduced with an advantage function $A(s, a)$ that estimates the relative benefit of taking an action from a given state:

$$L_\theta = \sum_{n=0}^{N} \sum_{t=0}^{T} \log \pi_\theta(a_t|s_t) A(s_t, a_t), \tag{2}$$

$$A(s_t, a_t) = r_t + \gamma V_\phi(s_{t+1}) - V_\phi(s_t). \tag{3}$$

Here, $V_\phi(s)$ estimates the expected discounted sum of rewards from the given state s if the agent acts as per policy π_θ. $V_\phi(s)$ is represented by a value neural network with parameters ϕ, and is optimized using gradient descent with the following loss:

$$L_\phi = \frac{1}{2} \left\| \sum_t V_\phi(s_t) - \Big(r_t + \gamma V_\phi(s_{t+1}) \Big) \right\|^2. \tag{4}$$

PPO modifies Equation 2 to stabilize the training of the policy with a trust region update [59]. It uses the following loss function:

$$L_\theta^{PPO} = \frac{1}{T} \sum_{t=0}^{T} \min(b_t(\theta)\hat{A}_t, \text{clip}(b_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t), \tag{5}$$

$$b_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}. \tag{6}$$

Here, $\pi_\theta(a|s)$ is the policy being updated with the loss function and $\pi_{\theta_{\text{old}}}(a|s)$ is the policy that was used to collect data with environment interaction. As the data collection policy differs from the policy being updated, it introduces a distribution shift in Equation 2. The ratio $b(\theta)$ corrects for this drift using importance sampling [26]. The ratio of two probabilities can blow up to large numbers and destabilize training, so the ratio is clipped with ϵ. The advantage

value $\hat{A}$ is estimated using a value network $V_\phi(s)$ as given in Equations 3 and 4. A variation of the algorithm uses generalized advantage estimation (GAE) [60] for computing $\hat{A}$. GAE takes into consideration the environment rewards across multiple time steps instead of just the most recent reward in Equation 3. We treat GAE as a hyper-parameter.

3 EMBER DESIGN AND METHODOLOGY

3.1 Objective and Scope

We target event-driven applications using batteryless energy-harvesting sensors. Particularly, we focus on detecting (i) motion using PIR sensors and (ii) environmental changes with regard to temperature, humidity, pressure, and light (referred to as THPL events hereafter). The objective of the system is to learn the environmental patterns and duty-cycle the sensors to maximize the event detection rate while avoiding energy storage depletion. If the node dies due to energy depletion, it can take several hours for the node to revive after recharging with harvested energy.

We focus on challenging real-world environments where energy harvesting availability is not enough for the sensors to stay always On. Thus, we select indoor sensing locations that are generally subject to low energy availability, such as *Door, Middle of Office, and Stairs Access/Corridor*. In the *Stairs/Corridor* case, lighting is low in intensity but always On for security reasons while the *Middle of an Office* and *Door* cases can be exposed to both interior lighting and exterior natural lighting coming through windows. We do not deploy sensors at locations such as *Windows* and *Interior Rooms*, as the former requires only trivial solutions where we can leave the sensors always On considering the abundance of energy available, while the latter has no energy available due to lack of windows and unoccupancy because of the Covid-19 pandemic.

3.2 Deep RL Problem Formulation

Objective: the objective of our system is twofold:

- Maximize event detection
- Avoiding node energy depletion

State Space:

- *Energy Storage:* The state of charge of the energy storage (continuous)
- *Hours of the day:* The current hour of the day (discrete)
- *Minutes of the day:* The current minute of the day (discrete)
- *Light:* The intensity of light in lux (continuous)
- *Weekday/Weekend:* Current day of the week (discrete). We included this information since human and environmental patterns vary significantly during a week inside buildings.

Action Space: The policy makes the following control decisions:

- *PIR On-Off*: a binary decision to turn *on* or *off* the PIR sensor (i.e., sensor ON or Off)
- *THPL On-Off*: a binary decision to turn *on* or *off* the temperature, humidity, pressure and light sensors; if *on* (THPL On), the node senses all these sensors every 1 min.

By controlling On-Off for PIR and THPL sensors, the policy decides whether to turn on the sensors. Each action is taken after *State Transition* period. By default, we use a *State Transition* time of 60 minutes, so actions are executed every hour. Using a large *State Transition* time, the system can decrease the energy used by communicating its action, but if the sensor is left Off it could miss events. Ideally, to catch a PIR event, the system should turn On the sensor right before the event happens and leave the sensor Off for the rest of the time. In this way, it will catch the event while maximizing energy savings to avoid energy storage depletion.

Table 2: Hyper-parameters and parameters used.

Hyper-Parameter	Value
Algorithm	PPO with Ray-RLlib [43]
State Transition (d)	60 min
Learning Rate (α)	0.01
Discount Factor (γ)	0.99
Episode (T)	24h
Max. Training Iters (N)	100
NN Model	2 hidden layers, 256 neurons each

These choices are empirically validated in Section 5.3. Rest of the hyper-parameters were left at default values in the Ray RLlib library.

Reward Function:

- +0.01 for each event detected
- -1 when a node depletes its energy storage

Our reward reflects the two-fold objective of our system: if the system catches an event it is positively rewarded; if the node dies it receives a large negative reward. Therefore, in order to maximize the reward, the system needs to find the best sequence of actions to catch as many events as possible while avoiding energy depletion. We opted for a large negative penalty if the super-capacitor is depleted because it can take several hours to charge, and no data can be transmitted during that time. We experiment with different parameter values of the reward functions in Section 5.2.4.

3.3 Methodology

We start from ideal training conditions with sufficient historical data and then move towards realistic situations by deploying a sensor at a new location without historical data.

Algorithm 1: Day-by-Day Algorithm

1: **if** Dataset D is empty **then**
2: Collect data D for one day using random policy
3: **end if**
4: **while** True **do**
5: Run the simulator with D till convergence using Algorithm 2 and obtain a trained policy π_θ
6: Collect data D' from one more day using trained policy, and update $D = D \cup D'$
7: **end while**

3.3.1 Learning with Historical Data. Ember uses a day-by-day learning approach, shown in Algorithm 1. Given a historical dataset D, we train an RL policy using PPO in a simulator as described in Algorithm 2. The simulator uses real-world lighting and event data, and simulates the energy consumed due to sensing and communication, energy gained from harvesting, based on agent actions. We validate that our simulator is representative in Section 5.1. Once the policy is trained, we deploy it on the real sensor node (line 5 of Algorithm 1). The policy duty-cycles the sensor node given the

Algorithm 2: PPO Simulator Algorithm

1: Initialize policy π_θ
2: **for** $i = 1, \ldots, N$ or till policy convergence **do**
3: Run Whole Episode using Current Policy
4: **for** $t = 1, \ldots, T$ **do**
5: $a_t \leftarrow \pi_\theta(s_t)$
6: $s_{t+1}, r_{t+1} \leftarrow Env(a_t)$
7: **end for**
8: Update Both Policy & Value Function using PPO
9: **for** $t = T, \ldots, 1$ **do**
10: $A(s_t, a_t) = r_t + \gamma V_\phi(s_{t+1}) - V_\phi(s_t)$
11: $b_t(\theta_i) = \frac{\pi_{\theta_i}(a_t|s_t)}{\pi_{\theta_{i-1}}(a_t|s_t)}$
12: **end for**
13: $\phi_{i+1} \leftarrow \phi_i + \alpha \nabla_\phi L_\phi(\phi_i)$
14: $\theta_{i+1} \leftarrow \theta_i + \alpha \nabla_\theta L^{\text{PPO}}(\theta_i)$
15: **end for**

environment state. We collect the interaction data in the form of (s, a, r, s') to further update the policy (line 6). At the end of each day, we use the data collected so far to re-train the agent in the simulator using Algorithm 2. The re-training does not restart with an empty policy but the previously learned policy is restored so that it can be further refined with the new data. Such re-training ensures the sensor node adapts to changing environmental patterns. The process is repeated for the duration of the experiment. Algorithm 2 shows the pseudo-code of the PPO algorithm used to generate our RL policy, details of which are described in Section 2.2. The training ends once Algorithm 2 converges: at the end of each episode the total reward obtained is compared with the previous total. If the current episode reward does not change by ± 3%, we consider the policy has converged. If the policy does not converge, the training will continue up to a maximum of 100 training iterations ($N = 100$).

With a day-by-day learning approach [23], the agent starts training using data currently available and runs till convergence. Once converged, it uses the policy to detect events on the following day, and the detection rate is evaluated for that day. The new data is added to the training dataset, and the agent re-trains the policy. This process is repeated throughout the experiment. Thus, testing is always performed on data *unknown* to the agent.

3.3.2 Self-Supervised Data Collection (SS Mode). But historical data might not be always available, especially if we deploy a sensor node in a new environment. In this case, the PPO agent will start collecting data using a random policy and it could take a while for the algorithm to discover new event patterns while taking random actions. We will see in Section 5, that by just using a random exploration policy the system will eventually keep catching the same event patterns over time and miss certain events because it was not able to collect the corresponding ground-truth data. To solve this problem, we design a *Self-Supervised Data Collection* mode (called SS Mode) that helps the node to collect missing events and to improve the capability of predicting data patterns. An alternative solution is to let the RL agent explore the state space in the real world and discover event patterns. However, exploration in the real world can take months/years to converge because of the large state-action space. The SS Mode overcomes this problem by following a pre-defined exploration strategy to discover event patterns systematically. The SS Mode works by leaving the sensor On for a certain amount of hours during the day (i.e., SS hours). It follows a round-robin approach in which each day the system keeps forcing the sensors On starting from the hour it ended in the day before. Once all the hours of the day are covered, it starts again from the beginning. In our design, the SS hours depend on the voltage level of the energy storage. Table 3 reports such correspondence. Our system checks the energy storage voltage level at each state transition, and updates the SS hours accordingly.

Table 3: SS Mode: to discover new events the node leaves its sensors On each day for a certain amount of hours (i.e., SS hours) depending on the energy storage voltage availability.

Energy (ES) Storage [%]	ES>100	100>ES ES>80	80>ES ES>60	60>ES ES>40	40>ES ES>20	20>ES
SS hours	24	18	12	6	0	0

Algorithm 3 reports the pseudo-code of the SS Mode. At the beginning of the experiment, the system sets the SS Mode to start from 12 am (line 1). Then, at each state transition the system reads the energy storage level, consults Table 3 to get *SS_hours* and sets the SS Mode (line 4). At this point, if the current hour of the day is in between *SS_start* and *SS_end*, then the sensors are forced to stay On (line 6). Outside the SS hours (line 9) the RL policy takes over and controls the sensors based on its prediction. It is important to notice that while SS hours can help in discovering new events, they can also result in energy waste if sensors are left On but no events happened during these hours. Therefore, it is important to carefully select SS hours based on the target environment. We selected these SS hours based on empirical experiments in simulation.

Algorithm 3: Self-Supervised Mode Algorithm

1: *SS_start* = 0
2: **while** True **do**
3: At every state transition, read energy storage level, get *SS_hours* from Table 3
4: *SS_end* = *SS_start* + *SS_hours*
5: **if** *SS_start* <= *curr_time* and *curr_time* < *SS_end* **then**
6: Force sensors On
7: *SS_start* = (*SS_start* + 1) modulo 24
8: **else**
9: Use RL Policy
10: **end if**
11: **end while**

4 HARDWARE AND SYSTEM ARCHITECTURE

4.1 Sensor Node

For our experiments, we use a general-purpose energy-harvesting batteryless sensor node for indoor applications. It harvests energy from ambient light with a small solar panel (AM-1454 [13]) and embeds the following sensors: motion, temperature, humidity, pressure,

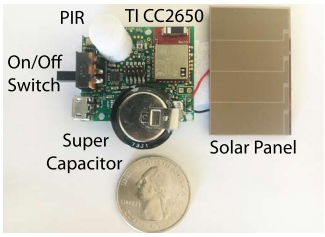

Figure 1: Energy-harvesting sensor node used in our study.

Table 4: Event defined with regard to sensor reading change.

Sensor	Measurement Change
PIR	person moving
Temperature	± 0.1C
Humidity	± 1 %
Pressure	± 100 Pa
Light	± 50 lux

light, microphone, IMU and a reed switch. For our experiments, we will only use the motion, temperature, humidity, pressure, and light sensors as they are commonly used in buildings [2]. It has a Bluetooth Low Energy (BLE) SoC (TI CC2650 Moda) on board. The board does not include any battery but can store part of the harvested energy in a supercapacitor for later use. The supercapacitor voltage is a 1.5F with 5.5V nominal voltage (Panasonic EEC-S5R5V155 [56]). When starting from full capacity, it can last up to 8 days by sensing and sending through BLE 1 data packet every 10 minutes without any lighting availability. The minimum usable voltage for all the sensors to correctly take their measurements is 3V. Therefore, for our experiments, we consider the energy storage to be depleted once it reaches 3V or lower. Figure 1 depicts our sensor node.

4.2 Event Definition

When the PIR is powered, an event is detected as soon as a person moves in its working range. When the temperature, humidity, pressure and light sensors are powered, the CPU wakes up every minute, polls all the sensors, and gets an event if any of the THPL measurements changes by at least the threshold values listed in Table 4. When an event is detected, all the sensors' data are transmitted.

4.3 Deployment Architecture

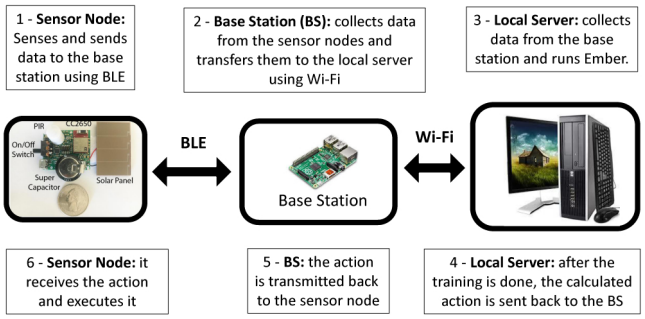

Figure 2: Deployment Overview.

Table 5: Current breakdown of the sensor node used. We consider all the current consumed by the system to wake up, read data from all the sensors, transmit the data using BLE, and the BLE message acknowledgment.

Feature	Current [μA]
Board Leakage	1
MCU in Sleep Mode	1
PIR On	1
Read THPL	1.2
PIR Detection	102
Read THPL + BLE Transmission	199
Solar Panel at 200 lux	35.2

Our wireless sensor network architecture includes three parts: the sensors, the base stations, and the local server as in a typical sensor network for buildings [2]. We only target one-hop sensor networks, where a node sends the data to the closest base station via BLE. Each base station sends the data to the local server using Wi-Fi for training. All our RL training is done on the local server and the base stations are only used to "pass" the data. After the training is done, the local server sends the calculated actions to the base station that further relays them to the sensors. The sensor nodes remain in the sleep mode until an event wakes up the sensors or the next communication schedule (i.e., state transition) is reached. Figure 2 illustrates the communication steps.

5 OFFLINE ANALYSIS

5.1 Modeling the Environment

Every time a sensor node communicates with the base station, the node sends its supercapacitor voltage level and the latest values from all the sensors (PIR value, light intensity, temperature, etc.). Our simulator includes all the node energy consumption parts (node sleeping, waking up, sensor reading, BLE communication, BLE message acknowledgment). All the energy consumption information is extracted from the datasheet of each component in the platform. Table 5 shows the current breakdown of the sensor node. To note that the majority of the energy is used to transmit data BLE data.

5.1.1 Modeling the Energy Storage Level: When lighting is available in the environment, the supercapacitor accumulates energy. We use the following formula to model its charging:
Energy Produced:

$$E_{Produced} = \text{Solar Power} \times \text{Time Period} \times \text{Light Intensity} \quad (7)$$

The solar panel mounted in the sensor node generates up to 35.2uA at 2.5V at 200lux[13]. To keep the model simple we ignore the solar plane inclination, the wavelength of light, and reflection of light. Even with our approximations, we show that the model is sufficient to capture real-world characteristics.
Energy Consumed: The supercapacitor dissipates energy while operating. We model its discharge as:

$$E_{Consumed} = E_{Send} + E_{Sleep} \quad (8)$$

E_{Send} is the energy consumed by the sensor node to wake up, read the sensors, and transmit a sensor data packet. E_{Sleep} is the energy consumed by the sensor node between two transmissions.

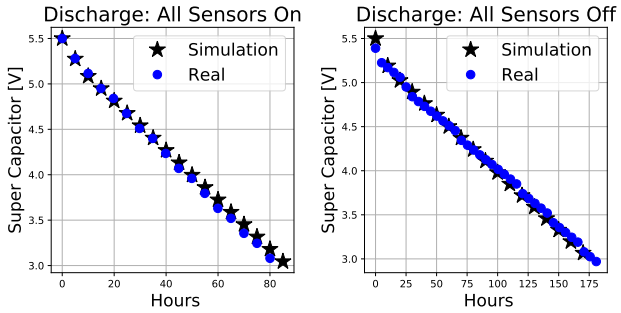

Figure 3: Supercapacitor discharge: a comparison between real-world and our simulator.

Figure 3 reports two examples of real super-capacitor discharges (blue circle dots) and the one modeled by our simulator (black stars). The figure on the left shows a discharge procedure of the supercapacitor when both the PIR and THPL sensors are On, while the figure on the right reports a discharge when the same sensors are Off. Nodes send a data packet every one hour with no lighting.

Our simple model closely follows the real-world node discharge. Turning On the sensors reduces the node lifetime by more than two-fold. Therefore, judicious use of energy is essential to managing the nodes in low energy conditions when it may not be able to fully charge the supercapacitor.

5.2 State and Action Spaces Analysis

We report on the simulation analysis that led to our state and action space selection. For all the simulations we use our day-by-day learning algorithm with historical data.

5.2.1 Evaluation Metrics. Over a specified period of time, we measure the performance of a policy based on the following metrics:

- Reward: accumulated rewards for the actions taken, calculated per our reward definition in Section 3.2;
- Event detection rate: the percentage of ground-truth events that are detected;
- Supercapacitor voltage difference: the difference (in percentage) in supercapacitor voltage at the end of the experiment vs at the beginning;

Note that the supercapacitor voltage difference indicates the energy used across the experiment (a positive value indicates energy gained). We use these metrics throughout the experiments in the rest of our study.

5.2.2 State Space. We report results using data-traces collected by a node placed in the middle of an office for 20 days. Results are averaged over 5 simulation runs. For the state space analysis we use a fixed 60-minute state transition. Table 6 reports the results.

We vary the input states and evaluate the policy using the metrics defined above. The more information we include in the state, the better the performance of the system as measured by these metrics. In all the cases, Ember spares energy indicating that to maximize the reward the policy is detecting events while avoiding storage depletion. *SC-Light-Week-Hours* obtain the highest reward. However, we choose *SC-Light-Week-Hours-Min* as our state space as it gains a small edge in energy while obtaining the same reward *SC-Light-Week-Hours*.

Table 6: State Space Analysis

Middle Office 5-Run Avg (Standard Deviation)	Reward [num]	Event [%] Detected	SC Voltage Diff [%]
SC	3.1 (0.1)	93.8 (1.4)	2.3 (0.1)
SC-Week	3.1 (0.1)	93.1 (0.6)	3.2 (0.1)
SC-Light-Week	3.4 (0.1)	93.9 (2.1)	8.1 (0.2)
SC-Light-Week-Hours	3.5 (0.2)	94.9 (2.1)	5.8 (0.1)
SC-Light-Week-Hours-Min	**3.5 (0.1)**	**94.8 (1.3)**	**6.5 (0.1)**
SC-Week-Hours	3.2 (0.1)	93.5 (1.5)	5.1 (0.1)
SC-Week-Hours-Min	3.3 (0.1)	94.7 (0.3)	4.2 (0.1)

Table 7: Action Space Analysis

Middle Office 5-Run Avg (std)	Reward [num]	Event [%] Detected	SC Voltage Diff [%]
Continuous (from 1 to 60 mins)	3.3 (0.1)	95.7 (1.2)	3.6 (0.1)
Discrete (1-15-30-45-60) mins	3.2 (0.1)	95.5 (0.9)	0.2(0.2)
Fix 60 mins	**3.5 (0.1)**	**94.8 (0.6)**	**6.5 (0.1)**
Fix 30 mins	1 (0.1)	87.8 (0.1)	-2.1 (0.1)
Fix 15 mins	-495 (na)	59.2 (Die)	-17.2 (na)

Table 8: Reward Space Analysis

Middle Office 5-Run Avg (std)	Reward [num]	Event [%] Detected	SC Voltage Diff [%]
Rew 0.1	82.8 (9.7)	93.3 (1.6)	-9.0 (0.1)
Rew 0.01	**3.5 (0.2)**	**94.9 (2.1)**	**5.8 (0.1)**
Rew 0.001	0.67 (0.1)	68.5 (2.3)	-6.2 (0.1)

5.2.3 Action Space. We vary the fixed state transition time, or make it part of the action space. In the *Continuous* and *Discrete* case, the agent picks the transition to be anywhere from 1 to 60 minutes. We use *SC-Light-Week-Hours-Min* as the state with the same 20 days of data collected for the previous experiment. Results in the table are averaged over 5 simulation runs.

The fixed 15-min state transition depletes the node's energy storage (marked as Die) as it sends too many data packets. Even if the highest event detection accuracy is achieved by *Continuous*, the highest reward is achieved by using a fixed 60-mins state transition. We further investigate why this may be the case in the next section.

5.2.4 Reward Function Design. We evaluate different choices of reward value Ember receives for each event detected. We keep "−1" as the reward received if the energy storage is depleted. For this experiment, we use *SC-Week-Light-Hours* as the input states, and a fixed 60-min state transition. Table 8 reports the results. The Table shows that the best results are obtained by using a reward of 0.01 for each event detected.

5.2.5 Non-Stationary Environment. Environmental event pattern changes over time, and the policy needs to adapt to such changes. Figure 4 illustrates PIR events for a node deployed in the middle of an office and shows how the policy adapts to detect the events. From the figure, the number of events shifts even in a few days of

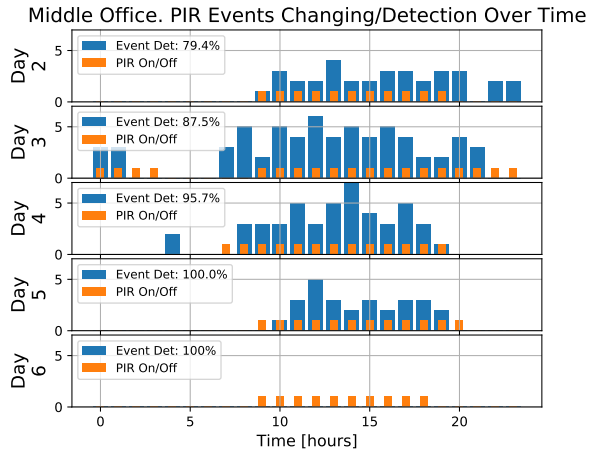

Figure 4: PIR events detected by a node deployed in the middle of an office and policy changes to detect the events.

deployment. But the policy learns the pattern and event detection improves from 79.4% to up to 100%.

5.3 Policy Learned using Historical data

We train RL policies using ground-truth PIR data collected by battery nodes for a week. Figures 5 and 6 show the results obtained at the three different locations. The experiments use the *Continuous* action space where the state transition varies from 1-60 min. We observe that Ember recognizes when events are happening and turns On the PIR sensor to detect them, and that the policy mostly uses a *60-min state transition* (as indicated in the bottom rows) as it is the most energy-efficient one. It is noteworthy that the policy recognizes environmental patterns based on the different input used and acts accordingly (note the state transition changes between 8-10 am on the left and between 12-16 in the middle due to changes in lighting pattern). In Figure 5-left (Middle of an Office), since events mostly happen when people are in the office and using the internal lights, the policy turns on the PIR sensor when lighting is available in the environment. In Figure 5-right (Stair Access), the internal lights are always On for security reasons, and the motion patterns are random. Even in these adverse conditions, Ember can capture 87% of the events. In Figure 6 (Door), since the average of events per day is very low and periodic, the policy learns the hourly pattern in which the events are happening. The PIR is turned On around the right time, achieving 100% of event detection. Figure 7 shows a zoomed-in version of the *Middle of an Office* case: the system uses faster state transitions when the PIR is Off, and turns it On right before events occur. It uses a slower state transition of 60 minutes when the PIR is left On as it is more energy-efficient.

While the actions selected by the policy are in general correct, they are not optimal and sometimes negatively affecting the energy consumption. Ideally, the policy will turn On the sensor right before the event happens and turn it Off after it catches the event. But at minute *20:39* Ember turns Off the sensor and uses two short state transitions (4 minutes and 1 minute) after which it turns On the sensor again. While the move is successful as the policy turns on the sensor right before the event happens, it used two BLE communications instead of one. As the BLE communication is the most energy-consuming part of our node (Table 5), we fix the state transition to 60 minutes as a more energy-efficient solution for all the remaining experiments.

PIR and THPL Events. Figure 8 shows an example of Ember taking actions to catch both PIR and THPL events while avoiding energy storage depletion. The policy correctly turns On both the sensors and catches up to 98.2% of the events. At the end of the experiment, the capacitor voltage level decreases by less than 1%.

RL Policy Convergence. In Figure 9, we show an example of the RL training progress using a single day of historical data. We define episodes of length 24 hours. The episodes cycle through all the historical data available during training. We randomly vary the supercapacitor voltage level values between its maximum (5.5V) and minimum (3V) values at the beginning of each episode. By doing so, the policy is exposed to different energy conditions and learns to avoid energy storage depletion during training. At the beginning of the experiment, the RL policy receives negative rewards as it does not know a valid sequence of actions that maximize the reward. The policy converges at the 40th training iteration. The whole training (i.e. 100 training iterations) takes ~5 minutes using a 2014 quad-core Intel Core-i7 CPU clocked at 3 GHz.

5.4 Simulating Self-Supervised Mode (SS Mode)

By keeping the sensors On for a certain amount of time, Ember can discover new events and add them in its training. We evaluate our self-supervised data collection mechanism in simulation, where the sensor is assumed to have no historical data. Figure 10 illustrates a simulation in which the system leaves the sensor On during SS mode. SS hours are the blue dots in the THPL On/Off row.

The system imposes 6 SS hours per Table 3 as the voltage availability falls between 4 and 4.5 Volts. The only THPL events discovered are those during SS hours. Outside SS hours the system saves energy as this is the second day of simulation and the system does not know of events patterns yet. The next day, Ember correctly turns On the sensors from 12 pm to 6 pm as it expects events. As the SS hours from the next day are from 6 pm to 12 pm, the event detection accuracy jumps to 90.8% for the coming day, demonstrating the effectiveness of the SS hours to discover events.

Figure 11 shows the trends of the event detection accuracy with and without SS Mode. We also report on the result of training using historical data as a comparison with the previous methods. Results are averaged per week. The use of the SS Mode drastically helps the system in discovering new events pattern over time. After the first week, without the SS Mode, the system detects 59% of the events, while with SS Mode it rises to 74%. After 3 weeks, the SS Mode increases its event detection rate to 89% while training with ground-truth data achieves 92%. On the other hand, when SS Mode is absent, the system achieves 77% event detection rate during week 2 and 78% on week 3, unable to improve in event detection due to lack of exploration via the SS mode.

5.5 System Performance

We evaluate Ember by running multiple experiments using 14 days of real-world data traces of PIR and THPL events from battery-powered nodes deployed in three different locations. The results from the same type of locations are averaged and shown in Table 9.

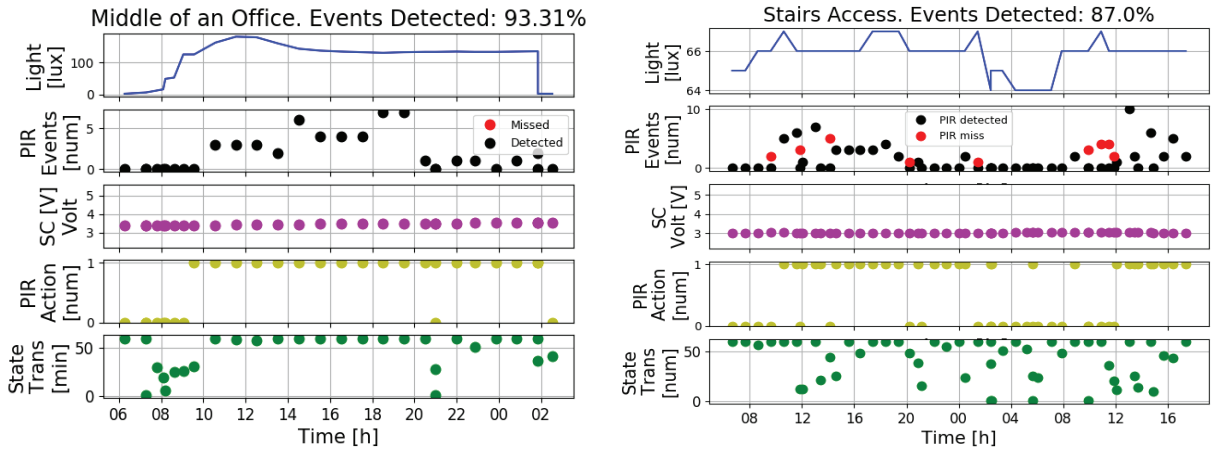

Figure 5: Ember results at different indoor locations: Ember controls nodes to detect PIR events. One node is located in the *Middle of an Office* (left) and one in the *Stair Access* (right).

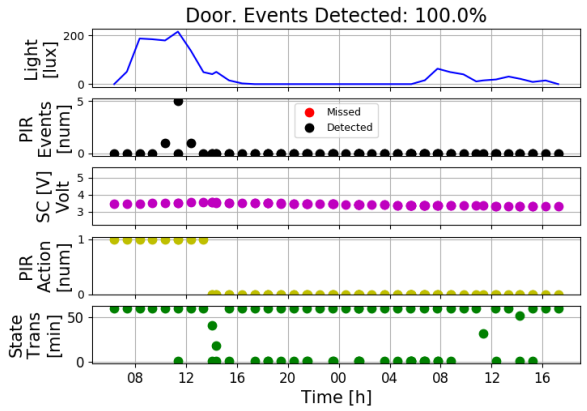

Figure 6: Results of Ember controlling a sensor node to detect PIR events in a *Door* case. All the events are detected.

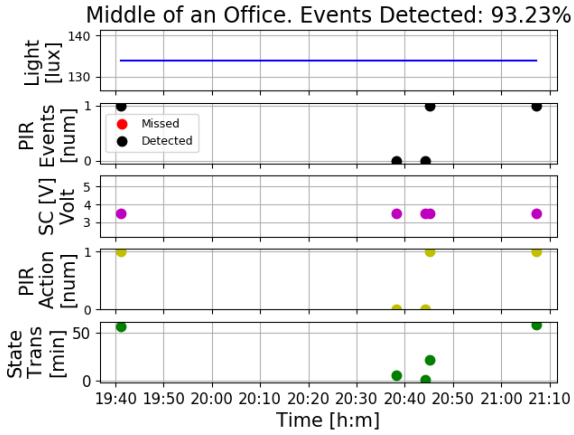

Figure 7: Zoomed-in view of the *Middle of Office* case: Ember at the right time switches the PIR On and Off to catch the events while saving energy whenever possible.

By using the SS Mode, Ember can detect 20.3% more events—78.1% vs 57.8%—in the Stairs/Corridor case and 11.3% more events on average on the three locations. At the same time, the energy difference of the supercapacitor slightly changes in all three cases, confirming the system's ability in managing the resources.

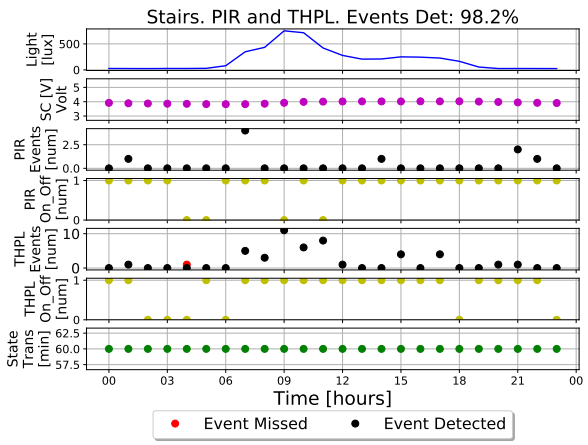

Figure 8: Ember can capture both PIR and THPL events while maintaining working order.

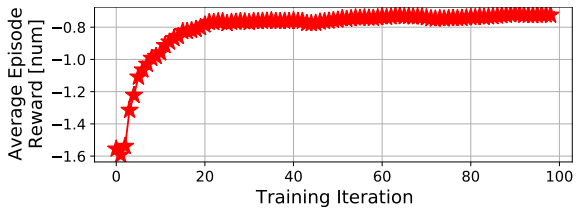

Figure 9: RL Policy Convergence Example. The policy converges around the 40th training iteration. The entire training procedure of 100 iterations takes around 5 minutes using a 2014 quad-core Intel Core-i7 CPU clocked at 3 GHz.

5.6 Ember v.s. State-of-the-Art Solutions

We compare Ember against several baseline techniques: (i) a fixed schedule where the sensors is active only from 6 am to 6 pm; (ii) a version of Ember that uses PPO with 20% random exploration rate.

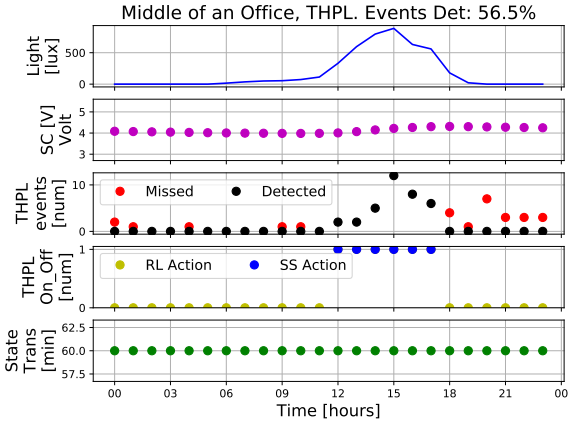

Figure 10: Self-Supervised Mode (SS Mode) simulator. The SS mode runs for six hours in this example, based on the super-capacitor voltage availability. Outside SS hours, the system runs the RL policy to decide On or Off for sensors.

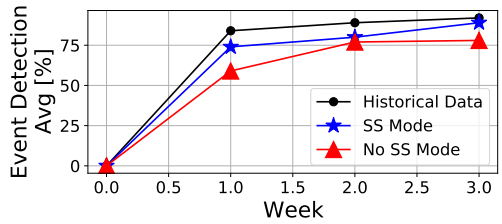

Figure 11: SS Mode trend: The use of the SS Mode helps the system in discovering new events over time.

Table 9: System Performance Evaluation. Simulations using 14 days of real-world data traces from nodes deployed at different locations. By using the SS Mode, Ember can on average detect 11% more events than without the SS Mode.

Event Det [%] (SC Volt diff [%])	Middle of Office	Stairs/ Corridor	Door	Average
Train with GT data	95.3 (-0.9)	87.1 (-1.5)	93.8 (-6.9)	92.1 (-3.1)
Not Using SS Mode	67.4 (-2.9)	57.8 (+4.4)	73.9 (-4.5)	66.4 (-1.1)
Using SS Mode	76.2 (-2.5)	78.1 (+0.9)	78.6 (-6.9)	77.6 (-2.7)

Using a higher exploration rate can help the system in discovering and detecting more events at the risk of depleting its energy; (iii) SpotON which uses Q-learning with the objective to turn On the sensors just before events happen and save energy [46]; (iv) ACES which duty cycles event sensors to maximize the On time of the sensors using Q-learning; it does not attempt to turn Off the sensors during idle periods [22]. We discretized the state space for SpotON and ACES, as the Q-learning algorithm does not support continuous state or action spaces. However, for a fair comparison, we used the PPO algorithm for all the RL baselines. SpotON only trains the policy once using historical data. We assume 1 day of historical data is available for SpotON. ACES uses a day-by-day learning

Table 10: Ember v.s. State-of-the-Art Solutions

Middle Position	Events [%] Detection	Storage [times] Depleted	SC Volt Diff [%]
Fixed Schedule (6am-6pm)	49.3	3	-15.7
PPO with 20% exploration	54.6	2	-12.3
SpotOn [46]	38.7	5	-9.1
ACES [22]	59.6	1	+7.1
Ember w/ SS Mode	76.2	0	-2.5

strategy similar to Ember. We evaluate these methods with 14 days of real data traces from a sensor at a *Middle of Office* location sensing THPL events. Compared to Fixed Schedule, Ember achieves higher event detection percentage (76.2% vs 49.3%) as Ember learns the environmental patterns and adapts the sensor activation whenever events are happening. Furthermore, Fixed Schedule is not able to slow down its operations even when there is no energy available in the environment, causing the node to stop operating 3 times during the experiment. Using PPO with a 20% random exploration rate can detect up to 54.6% of events as the system explores and finds more events but also wastes energy as the exploration is random and does not consider energy availability, which leads the node to deplete its energy storage 2 times during the experiment. SpotOn detects only 38.7% of the events. The results are strongly influenced by the lack of a day-by-day learning mechanism that helps to adapt to varying environmental patterns, and its performance drastically reduces after the initial few days of simulation. Not adapting to any changes, the system depletes its energy storage 5 times during the experiment causing even more events to be missed. Finally, ACES achieves 59.6% in event detection and depletes its energy storage once. Its improved results compared to Fixed Schedule and SpotOn are attributed to its capability to adapt to the changing patterns using day-by-day learning. On the other side, the events detected are 16.6% fewer than Ember using SS Mode as ACES approximates the environment using Q-learning and it does not embed any mechanism to discover new event patterns over time.

6 EMBER IN THE FIELD

We deployed 56 nodes in our building to collect information about both PIR motion and THPL events. 16 nodes are battery-powered to collect ground-truth data. The remaining nodes are distributed among *Door*, *Stairs/Corridor* and *Middle of an Office* locations. Figure 12 shows the location of the sensors in our building floor plan.

6.1 Training with Historical Data

We train our system with ground-truth data collected by a battery-powered sensor node and then deploy the trained RL policy on a sensor node deployed side by side with the battery-powered node. The two sensors collect both THPL events and are deployed in the middle of an office. We shall note that, even though we did our best to put the nodes as close as possible and in similar lighting conditions, there are still differences in the lighting the nodes are subject to. These variations can be due to both the sensors having a slightly different inclination when attached to the wall and the variations in factory calibration.

Figure 13 illustrates the comparison between a battery-powered node (in black circled dots) and a node managed using Ember (in

Figure 12: Node locations in our test building department.

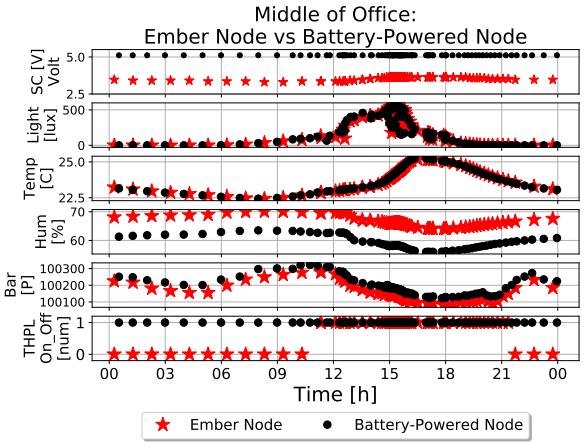

Figure 13: Comparison of a battery-powered node and a node controlled by Ember. The nodes are deployed side by side in the *Middle of an Office* collecting THPL data.

red star dots) in collecting THPL events. Each dot represents a data packet that the node transmits to the base station.

We first remark on the observations from the battery-powered sensors. The majority of the events occur between 11 am and 10 pm. Outside this interval, data are sent at the minimal rate of every hour, meaning no events occurred during this time. THPL events occur in quick succession once the light enters the room from the window in the morning. The light increases in intensity, the temperature rises, the humidity and pressure decrease. Even if the two sensors are placed side by side, the humidity sensors in the two boards differ in their absolute values. However, the trend is the same over time, and therefore, does not affect our experiments.

We now note the observations from the Ember nodes. The policy recognizes THPL event patterns and turns On the sensors only when many events occur. Overall, it can capture up to 93.2% of the events for the day shown in Figure 13. During the whole experiment, Ember was able to collect up to 94.7% of the events. Events are

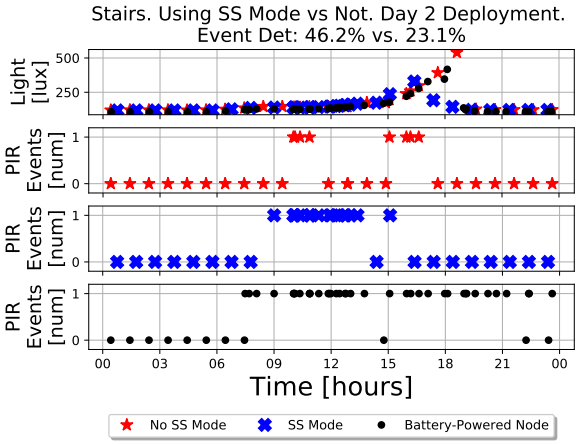

Figure 14: Real-world results on the second day of deployment of three nodes deployed side by side at a *Stair Access* location detecting PIR events. One node uses Ember without the SS Mode, one node with the SS Mode, and a third battery-powered node for ground truth comparison. By using the SS Mode, the detection rate doubles from 23.1% to 46.2%.

missed for two reasons: a wrong policy generated at time 12:00 in the paper and network disconnections at time 17:00.

6.2 Self-Supervised Mode

To evaluate the SS Mode, we deploy two sensor nodes side by side in the building and drive them with two different algorithms: one runs Ember without using the SS Mode and one with the SS Mode. We also deploy a third battery-powered node to collect the ground-truth events. Nodes were deployed for a week collecting PIR events. Figure 14 reports the real-world results on the second day of deployment of the three nodes deployed at a *Stair Access*. The light is always On at this location for security reasons and it changes only around 4 pm as light enters through a window. The use of the SS Mode (in blue crosses), compared to not using it (red stars), improves the discovery of new events. During the second day of deployment, the events detected doubles from 23.1% to 46.2%.

6.3 Summary of Real-World Deployment

In total, we deployed 40 sensor nodes running Ember at different locations in the test building for 10 days. Table 11 reports the results averaged over the nodes deployed at the same type of location, with some collecting only PIR events, some only THPL events, and some collecting both PIR and THPL events.

The SS Mode improves event detection in all types of locations with varying lighting conditions. For example, in the *Door* case the SS Mode detects 92% of events on average while without SS Mode, the rate is 76.2%. The results confirm the importance of using the SS Mode to help the system discover new events over time. All the nodes deployed managed the available energy resources throughout the experiment. In the worst-case scenario, the nodes lost only 3.9% of their voltage level compared to the beginning of the experiment. We note that the event detection rate degraded in the real world deployment environment compared to simulation (from 95% to 78% for the office case). There are several reasons: (i) Simulations were

Table 11: Performance of our 40-node sensor network. Results are averaged per location category over 10 days with both PIR and THPL events considered.

Summary of Real-World Deployment	Event Detection [%]	SC Volt Diff [%]	Avg Light [lux]	Avg Packets [num]
Office - No SS	70.4	+1.4	182	46
Office - SS Mode	78.1	-3.9	207	50
Stairs - No SS	74.5	1.5	118	29
Stairs - SS Mode	87.3	-3	130	41
Door - No SS	76.2	1.9	213	32
Door - SS Mode	92.0	1.1	220	39

conducted using data collected pre-pandemic while the real-world experiments were conducted after the building was closed due to Covid-19. Therefore, during the real-world experiment, people's access to the building was limited, indoor light availability (i.e. energy availability) was reduced, and event patterns were more sporadic, all making it more difficult for Ember to learn the environmental patterns. (ii) Simulations used 20 days of data while the real-world experiments lasted for only 10 days. As shown in Figure 11, Ember needs several days to learn environmental pattern: the event detection rate is low in the first few days, and increases with time.

6.4 Real-World Experience and Limitations

While we have shown that our system can learn event patterns and control sensor nodes, we note a few drawbacks of Ember.

Not All Events Are Equal. As seen in Figure 13, many of the events missed by Ember are either the first or the last events of the day. Ember tends to turn On the sensors whenever the majority of the events occur during the day to maximize its reward while reducing its energy consumption. But for many building applications (e.g., HVAC, lighting control), these events are critical for daily operations. To solve this problem, we could give more rewards to the system for the first and last event of a day. This way, Ember should spare enough energy to catch these critical events.

Working at the Limit. While working in low energy conditions, Ember drives the sensors close to their storage depletion limit. Even though the nodes did not deplete their storage in our 10-day experiment, a sudden change in the environment (e.g., if lights malfunction, rainy weather) could cause the nodes to deplete their energy as the policy was not trained for such situations. To overcome this problem, we can set a safety margin in our energy storage level so a minimal amount of energy is left for critical operations. The safety margin can also be encoded into the reward function.

SS Mode - Round Robin. We proved the use of the SS Mode is critical in helping the policy discover new events, especially during the first days of deployment. However, with Table 3, we introduced a round-robin heuristic approach that could be wasteful in some situations. As an example, if the sensor node is placed in an office where the occupancy schedule follows a 8am to 5pm pattern, it will be wasteful of energy to keep a PIR sensor *on* during the night to discover new events. Instead, the sensor could learn to discover new events from 6 am to 6 pm the majority of the time. An automated discovery mechanism is an interesting avenue of future work.

Node Energy Consumption Model. We built an accurate model of the sensor node energy consumption by carefully examining the specification of each component in the datasheet and by measuring the current consumed by the node in different modes of operation. While such a model is a one-time effort for each hardware, the method will be difficult to scale for more complex embedded systems. Automated methods to build accurate simulators are an interesting direction for future work. Furthermore, as some components are expected to change their behaviors over time (e.g. supercapacitor), their simulator's model has to be updated accordingly to avoid poor real-world performance.

Battery Replacement. During our experiments we had to replace batteries at a constant rate and many results were ruined by the lack of access to the building due to the Covid-19 pandemic induced shutdown. In only a few weeks of experiments, we replaced 27 coin-cell batteries for 16 battery-powered nodes as the lifetime of the batteries were around 1 month. This experience anecdotally reaffirms the challenges of relying on batteries to power IoT devices. We hope our work encourages adoption and further research of batteryless, energy-harvesting systems.

Planning Deployment. If the node is subject to very low energy conditions the node will systematically deplete its energy storage event by using the minimum operations. Therefore, the position of the nodes has to be planned before deployment.

Adversary Threats. If an adversary generates a fake event or environmental patterns, the system will waste energy in trying to capture these spurious patterns, resulting in poor performance.

Training Data Timespan. When trained on data over too long a period, the system could end up learning to detect obsolete event patterns. As events can shift across seasons, the agent should focus on learning only the most recent patterns. Adding a "seasonal" input in the observation space can potentially overcome this problem.

7 CONCLUSION

We present Ember, an energy management system for event detection with batteryless sensors using deep reinforcement learning. We evaluated Ember using real-world data traces and showed that Ember can recognize event patterns in the environment and control motion, temperature, humidity, pressure, and light sensors to catch environmental events with ambient light energy harvesting. We implemented a self-supervised ground-truth data collection mechanism that helps the system discover new events when there is no historical data to train with. We evaluated our system in both simulations and real-world experiments as we report the results of a 40-node wireless sensor network deployment in an office building. We show that Ember outperforms previous methods and can catch up to 27% more events compared to state-of-the-art techniques that also uses reinforcement learning while avoiding energy storage depletion in different indoor lighting conditions. As future work, we plan to explore the use of data collected by multiple nodes to speed up the learning of newly deployed sensor nodes.

ACKNOWLEDGMENTS

The research reported in this paper was sponsored by the CONIX Research Center, one of six centers in JUMP, a Semiconductor Research Corporation (SRC) program sponsored by DARPA.

REFERENCES

[1] [n.d.]. https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/.

[2] Yuvraj Agarwal, Bharathan Balaji, Rajesh Gupta, Jacob Lyles, Michael Wei, and Thomas Weng. 2010. Occupancy-driven Energy Management for Smart Building Automation. In *Proceedings of the 2Nd ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Building (BuildSys '10)*. ACM, New York, NY, USA, 1–6. https://doi.org/10.1145/1878431.1878433

[3] OpenAI: Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, et al. 2020. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research* 39, 1 (2020), 3–20.

[4] Fayçal Ait Aoudia, Matthieu Gautier, and Olivier Berder. 2018. RLMan: an Energy Manager Based on Reinforcement Learning for Energy Harvesting Wireless Sensor Networks. *IEEE Transactions on Green Communications and Networking* 2, 2 (2018), 408–417.

[5] Bharathan Balaji, Sunil Mallya, Sahika Genc, Saurabh Gupta, Leo Dirac, Vineet Khare, Gourav Roy, Tao Sun, Yunzhe Tao, Brian Townsend, et al. 2020. DeepRacer: Autonomous Racing Platform for Experimentation with Sim2Real Reinforcement Learning. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2746–2754.

[6] Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Dębiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, et al. 2019. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680* (2019).

[7] Bradford Campbell, Joshua Adkins, and Prabal Dutta. 2016. Cinamin: A Perpetual and Nearly Invisible BLE Beacon. In *Proceedings of the 2016 International Conference on Embedded Wireless Systems and Networks (EWSN '16)*. Junction Publishing, USA, 331–332. http://dl.acm.org/citation.cfm?id=2893711.2893793

[8] Bradford Campbell, Meghan Clark, Samuel DeBruin, Branden Ghena, Neal Jackson, Ye-Sheng Kuo, and Prabal Dutta. 2016. perpetual Sensing for the Built environment. *IEEE Pervasive Computing* 15, 4 (2016), 45–55.

[9] Bradford Campbell and Prabal Dutta. 2014. An Energy-harvesting Sensor Architecture and Toolkit for Building Monitoring and Event Detection. In *Proceedings of the 1st ACM Conference on Embedded Systems for Energy-Efficient Buildings (BuildSys '14)*. New York, NY, USA, 100–109. https://doi.org/10.1145/2674061.2674083

[10] Qingping Chi, Hairong Yan, Chuan Zhang, Zhibo Pang, and Li Da Xu. 2014. A reconfigurable smart sensor interface for industrial WSN in IoT environment. *IEEE transactions on industrial informatics* 10, 2 (2014), 1417–1425.

[11] Man Chu, Hang Li, Xuewen Liao, and Shuguang Cui. 2018. Reinforcement learning-based multiaccess control and battery prediction with energy harvesting in IoT systems. *IEEE Internet of Things Journal* 6, 2 (2018), 2009–2020.

[12] Man Chu, Xuewen Liao, Hang Li, and Shuguang Cui. 2019. Power Control in Energy Harvesting Multiple Access System With Reinforcement Learning. *IEEE Internet of Things Journal* 6, 5 (2019), 9175–9186.

[13] Sanio Semiconductor CO. 2007. https://media.digikey.com/pdf/Data%20Sheets/Sanyo%20Energy/Amorphous_Br.pdf.

[14] Karl Cobbe, Oleg Klimov, Chris Hesse, Taehoon Kim, and John Schulman. 2019. Quantifying Generalization in Reinforcement Learning. arXiv:cs.LG/1812.02341

[15] Alexei Colin and Brandon Lucia. 2016. Chain: tasks and channels for reliable intermittent programs. In *Proceedings of the 2016 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications*. 514–530.

[16] Samuel DeBruin, Bradford Campbell, and Prabal Dutta. 2013. Monjolo: An Energy-harvesting Energy Meter Architecture. In *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems (SenSys '13)*. ACM, New York, NY, USA, Article 18, 14 pages. https://doi.org/10.1145/2517351.2517363

[17] Gabriel Martins Dias, Maddalena Nurchis, and Boris Bellalta. 2016. Adapting sampling interval of sensor networks using on-line reinforcement learning. In *Internet of Things (WF-IoT), 2016 IEEE 3rd World Forum on*. IEEE, 460–465.

[18] Alexey Dosovitskiy, Philipp Fischer, Jost Tobias Springenberg, Martin Riedmiller, and Thomas Brox. 2015. Discriminative unsupervised feature learning with exemplar convolutional neural networks. *IEEE transactions on pattern analysis and machine intelligence* 38, 9 (2015), 1734–1747.

[19] Joshua F Ensworth and Matthew S Reynolds. 2017. Ble-backscatter: ultralow-power IoT nodes compatible with bluetooth 4.0 low energy (BLE) smartphones and tablets. *IEEE Transactions on Microwave Theory and Techniques* 65, 9 (2017), 3360–3368.

[20] Benjamin Eysenbach, Abhishek Gupta, Julian Ibarz, and Sergey Levine. 2018. Diversity is all you need: Learning skills without a reward function. *arXiv preprint arXiv:1802.06070* (2018).

[21] Francesco Fraternali, Bharathan Balaji, Yuvraj Agarwal, Luca Benini, and Rajesh K. Gupta. 2018. Pible: Battery-Free Mote for Perpetual Indoor BLE Applications. In *Proceedings of the 5th ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Building (BuildSys '18)*. ACM.

[22] Francesco Fraternali, Bharathan Balaji, Yuvraj Agarwal, and Rajesh K. Gupta. 2020. ACES: Automatic Configuration of Energy Harvesting Sensors with Reinforcement Learning. *ACM Trans. Sen. Netw.* 16, 4, Article 36 (July 2020), 31 pages. https://doi.org/10.1145/3404191

[23] Francesco Fraternali, Bharathan Balaji, and Rajesh Gupta. 2018. Scaling Configuration of Energy Harvesting Sensors with Reinforcement Learning. In *Proceedings of the 6th International Workshop on Energy Harvesting & Energy-Neutral Sensing Systems (ENSsys '18)*. ACM, New York, NY, USA, 7–13. https://doi.org/10.1145/3279755.3279760

[24] F. Fraternali and et al. 2018. Pible: Battery-Free Mote for Perpetual Indoor BLE Applications: Demo Abstract. In *Proceedings of the 5th Conference on Systems for Built Environments (BuildSys '18)*. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3276774.3282823

[25] Spyros Gidaris, Praveer Singh, and Nikos Komodakis. 2018. Unsupervised representation learning by predicting image rotations. *arXiv preprint arXiv:1803.07728* (2018).

[26] Peter W Glynn and Donald L Iglehart. 1989. Importance sampling for stochastic simulations. *Management science* 35, 11 (1989), 1367–1392.

[27] Graham Gobieski, Brandon Lucia, and Nathan Beckmann. 2019. Intelligence beyond the edge: Inference on intermittent embedded systems. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, 199–213.

[28] Josiah Hester, Lanny Sitanayah, and Jacob Sorber. 2015. Tragedy of the Coulombs: Federating Energy Storage for Tiny, Intermittently-Powered Sensors. 5–16. https://doi.org/10.1145/2809695.2809707

[29] Josiah Hester and Jacob Sorber. 2017. Flicker: Rapid prototyping for the batteryless internet-of-things. In *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems*. ACM, 19.

[30] Josiah Hester and Jacob Sorber. 2017. New Directions: The Future of Sensing is Ba eryless, Intermi ent, and Awesome. (2017).

[31] Josiah Hester, Kevin Storer, and Jacob Sorber. 2017. Timely execution on intermittently powered batteryless sensors. In *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems*. 1–13.

[32] Jason Hsu, Sadaf Zahedi, Aman Kansal, Mani Srivastava, and Vijay Raghunathan. 2006. Adaptive duty cycling for energy harvesting systems. In *Proceedings of the 2006 international symposium on Low power electronics and design*. 180–185.

[33] Roy Chaoming Hsu, Cheng-Ting Liu, and Wei-Ming Lee. 2009. Reinforcement learning-based dynamic power management for energy harvesting wireless sensor network. In *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*. Springer, 399–408.

[34] R. C. Hsu, C. T. Liu, and H. L. Wang. 2014. A Reinforcement Learning-Based ToD Provisioning Dynamic Power Management for Sustainable Operation of Energy Harvesting Wireless Sensor Node. *IEEE Transactions on Emerging Topics in Computing* 2, 2 (June 2014), 181–191. https://doi.org/10.1109/TETC.2014.2316518

[35] R. C. Hsu, C. T. Liu, K. C. Wang, and W. M. Lee. 2009. QoS-Aware Power Management for Energy Harvesting Wireless Sensor Network Utilizing Reinforcement Learning. In *2009 International Conference on Computational Science and Engineering*, Vol. 2. 537–542. https://doi.org/10.1109/CSE.2009.83

[36] Neal Jackson, Joshua Adkins, and Prabal Dutta. 2019. Capacity over capacitance for reliable energy harvesting sensors. In *Proceedings of the 18th International Conference on Information Processing in Sensor Networks*. ACM, 193–204.

[37] Gregory Kahn, Adam Villaflor, Bosen Ding, Pieter Abbeel, and Sergey Levine. 2018. Self-supervised deep reinforcement learning with generalized computation graphs for robot navigation. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 1–8.

[38] Aman Kansal, Jason Hsu, Sadaf Zahedi, and Mani B. Srivastava. 2007. Power Management in Energy Harvesting Sensor Networks. *ACM Trans. Embed. Comput. Syst.* 6, 4, Article 32 (Sept. 2007). https://doi.org/10.1145/1274858.1274870

[39] Bryce Kellogg, Aaron Parks, Shyamnath Gollakota, Joshua R Smith, and David Wetherall. 2014. Wi-Fi backscatter: Internet connectivity for RF-powered devices. In *ACM SIGCOMM Computer Communication Review*, Vol. 44. ACM, 607–618.

[40] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*. 1097–1105.

[41] Feng Li, Tianyu Xiang, Zicheng Chi, Jun Luo, Lihua Tang, Liya Zhao, and Yaowen Yang. 2013. Powering indoor sensing with airflows: a trinity of energy harvesting, synchronous duty-cycling, and sensing. In *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems*. ACM, 73.

[42] Yan Li, Zicheng Chi, Xin Liu, and Ting Zhu. 2018. Passive-ZigBee: enabling ZigBee communication in IoT networks with 1000x+ less power consumption. In *Proceedings of the 16th ACM Conference on Embedded Networked Sensor Systems*. 159–171.

[43] Eric Liang, Richard Liaw, Robert Nishihara, Philipp Moritz, Roy Fox, Ken Goldberg, Joseph Gonzalez, Michael Jordan, and Ion Stoica. 2018. RLlib: Abstractions for distributed reinforcement learning. In *International Conference on Machine Learning*. 3053–3062.

[44] Vincent Liu, Aaron Parks, Vamsi Talla, Shyamnath Gollakota, David Wetherall, and Joshua R Smith. 2013. Ambient backscatter: Wireless communication out of

thin air. *ACM SIGCOMM Computer Communication Review* 43, 4 (2013), 39–50.
[45] Brandon Lucia, Vignesh Balaji, Alexei Colin, Kiwan Maeng, and Emily Ruppel. 2017. Intermittent Computing: Challenges and Opportunities. In *LIPIcs-Leibniz International Proceedings in Informatics*, Vol. 71. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.
[46] Yubo Luo and Shahriar Nirjon. 2019. SpotON: Just-in-Time Active Event Detection on Energy Autonomous Sensing Systems. *Brief Presentations Proceedings (RTAS 2019)* (2019), 9.
[47] Amjad Yousef Majid, Patrick Schilder, and Koen Langendoen. 2020. Continuous Sensing on Intermittent Power. In *2020 19th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*. IEEE, 181–192.
[48] Hongzi Mao, Mohammad Alizadeh, Ishai Menache, and Srikanth Kandula. 2016. Resource management with deep reinforcement learning. In *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*. ACM, 50–56.
[49] Artem Molchanov, Tao Chen, Wolfgang Hönig, James A Preiss, Nora Ayanian, and Gaurav S Sukhatme. 2019. Sim-to-(Multi)-Real: Transfer of Low-Level Robust Control Policies to Multiple Quadrotors. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 59–66.
[50] C. Moser, L. Thiele, D. Brunelli, and L. Benini. 2010. Adaptive Power Management for Environmentally Powered Systems. *IEEE Trans. Comput.* 59, 4 (April 2010), 478–491. https://doi.org/10.1109/TC.2009.158
[51] Abdulmajid Murad, Frank Alexander Kraemer, Kerstin Bach, and Gavin Taylor. 2019. Autonomous Management of Energy-Harvesting IoT Nodes Using Deep Reinforcement Learning. *arXiv preprint arXiv:1905.04181* (2019).
[52] Saman Naderiparizi, Aaron N Parks, Zerina Kapetanovic, Benjamin Ransford, and Joshua R Smith. 2015. WISPCam: A battery-free RFID camera. In *RFID (RFID), 2015 IEEE International Conference on*. IEEE, 166–173.
[53] Matteo Nardello, Harsh Desai, Davide Brunelli, and Brandon Lucia. 2019. Camaroptera: A Batteryless Long-Range Remote Visual Sensing System. In *Proceedings of the 7th International Workshop on Energy Harvesting & Energy-Neutral Sensing Systems (ENSsys'19)*. Association for Computing Machinery, New York, NY, USA, 8–14. https://doi.org/10.1145/3362053.3363491
[54] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. 2018. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748* (2018).
[55] OpenAI, Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, Jonas Schneider, Szymon Sidor, Josh Tobin, Peter Welinder, Lilian Weng, and Wojciech Zaremba. 2019. Learning Dexterous In-Hand Manipulation. arXiv:cs.LG/1808.00177
[56] Panasonic. 2018. https://media.digikey.com/pdf/Data%20Sheets/Panasonic%20Electronic%20Components/SG_Series_LowTemp.pdf.
[57] Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. 2017. Curiosity-driven exploration by self-supervised prediction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 16–17.
[58] Lerrel Pinto and Abhinav Gupta. 2016. Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours. In *2016 IEEE international conference on robotics and automation (ICRA)*. IEEE, 3406–3413.
[59] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. 2015. Trust region policy optimization. In *International Conference on Machine Learning*. 1889–1897.
[60] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. 2015. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438* (2015).
[61] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347* (2017).
[62] Vinod Sharma, Utpal Mukherji, Vinay Joseph, and Shrey Gupta. 2010. Optimal energy management policies for energy harvesting sensor nodes. *IEEE Transactions on Wireless Communications* 9, 4 (2010), 1326–1336.
[63] Evan Shelhamer, Parsa Mahmoudieh, Max Argus, and Trevor Darrell. 2016. Loss is its own reward: Self-supervision for reinforcement learning. *arXiv preprint arXiv:1612.07307* (2016).
[64] Shaswot Shresthamali, Masaaki Kondo, and Hiroshi Nakamura. 2019. Power Management of Wireless Sensor Nodes with Coordinated Distributed Reinforcement Learning. In *2019 IEEE 37th International Conference on Computer Design (ICCD)*. IEEE, 638–647.
[65] Rishi Shukla, Neev Kiran, Rui Wang, Jeremy Gummeson, and Sunghoon Ivan Lee. 2019. SkinnyPower: enabling batteryless wearable sensors via intra-body power transfer. In *Proceedings of the 17th Conference on Embedded Networked Sensor Systems*. 68–82.
[66] Sujesha Sudevalayam and Purushottam Kulkarni. 2011. Energy harvesting sensor nodes: Survey and implications. *IEEE Communications Surveys & Tutorials* 13, 3 (2011), 443–461.
[67] Richard S Sutton and Andrew G Barto. 1998. *Introduction to reinforcement learning*. Vol. 135. MIT press Cambridge.
[68] Adrian Udenze and Klaus McDonald-Maier. 2009. Direct reinforcement learning for autonomous power configuration and control in wireless networks. In *Adaptive Hardware and Systems, 2009. AHS 2009. NASA/ESA Conference on*. IEEE, 289–296.
[69] Christopher M Vigorito, Deepak Ganesan, and Andrew G Barto. 2007. Adaptive control of duty cycling in energy-harvesting wireless sensor networks. In *2007 4th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks*. IEEE, 21–30.
[70] Xiaolong Wang and Abhinav Gupta. 2015. Unsupervised learning of visual representations using videos. In *Proceedings of the IEEE international conference on computer vision*. 2794–2802.
[71] Christopher JCH Watkins and Peter Dayan. 1992. Q-learning. *Machine learning* 8, 3-4 (1992), 279–292.
[72] Kok-Lim Alvin Yau, Peter Komisarczuk, and Paul D Teal. 2012. Reinforcement learning for context awareness and intelligence in wireless networks: Review, new features and open issues. *Journal of Network and Computer Applications* 35, 1 (2012), 253–267.
[73] Kasım Sinan Yıldırım, Amjad Yousef Majid, Dimitris Patoukas, Koen Schaper, Przemyslaw Pawelczak, and Josiah Hester. 2018. Ink: Reactive kernel for tiny batteryless sensors. In *Proceedings of the 16th ACM Conference on Embedded Networked Sensor Systems*. 41–53.
[74] Andy Zeng, Shuran Song, Stefan Welker, Johnny Lee, Alberto Rodriguez, and Thomas Funkhouser. 2018. Learning synergies between pushing and grasping with self-supervised deep reinforcement learning. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 4238–4245.
[75] Pengyu Zhang, Dinesh Bharadia, Kiran Joshi, and Sachin Katti. 2016. Hitchhike: Practical backscatter using commodity wifi. In *Proceedings of the 14th ACM Conference on Embedded Network Sensor Systems CD-ROM*. 259–271.