Marble: Collaborative Scheduling of Batteryless Sensors with Meta Reinforcement Learning

Francesco Fraternali
University of California, San Diego
frfrater@ucsd.edu

Bharathan Balaji*
Amazon
bhabalaj@amazon.com

Dezhi Hong
University of California, San Diego
dehong@ucsd.edu

Yuvraj Agarwal
Carnegie Mellon University
yuvraj@cs.cmu.edu

Rajesh K. Gupta
University of California, San Diego
rgupta@ucsd.edu

ABSTRACT

Batteryless energy-harvesting sensing systems are attractive for low maintenance but face challenges in real-world applications due to low quality of service from sporadic and unpredictable energy availability. To overcome this challenge, recent data-driven energy management techniques optimize energy usage to maximize application performance even in low harvested energy scenarios by learning energy availability patterns in the environment. These techniques require prior knowledge of the environment in which the sensor nodes are deployed to work correctly. In the absence of historical data, the application performance deteriorates.

To overcome this challenge, we describe here the use of meta reinforcement learning to increase the application performance of newly deployed batteryless sensor nodes without historical data. Our system, called Marble, exploits information from other sensor node locations to expedite the learning of newly deployed sensor nodes, and improves application performance in the initial period of deployment. Our evaluation using real-world data traces shows that Marble detects up to 66% more events in low lighting conditions, and up to 25.6% more events on average on the first 3 days of deployment compared to the state-of-the-art.[1]

CCS CONCEPTS

• **Computing methodologies** → **Multi-agent systems**; • **Computer systems organization** → *Sensor networks.*

KEYWORDS

Batteryless, Wireless Sensor Network, Energy Harvesting, Smart Buildings, Deep Reinforcement Learning, Collaborative Learning, Perpetual Operations

*work done outside of Amazon

[1]Code and data available at https://github.com/francescofraternali/MAML_on_ray.git

BuildSys '21, November 17–18, 2021, Coimbra, Portugal

ACM ISBN 978-1-4503-9114-6/21/11…$15.00
https://doi.org/10.1145/3486611.3486670

ACM Reference Format:
Francesco Fraternali, Bharathan Balaji, Dezhi Hong, Yuvraj Agarwal, and Rajesh K. Gupta. 2021. Marble: Collaborative Scheduling of Batteryless Sensors with Meta Reinforcement Learning. In *ACM International Conference on Systems for Energy-Efficient Built Environments (BuildSys '21), November 17–18, 2021, Coimbra, Portugal.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3486611.3486670

1 INTRODUCTION

Embedded sensing and actuation devices are used in many applications such as to monitor traffic conditions, structural health of civil structures, environmental conditions in buildings, crops in agriculture. These deployments often consist of thousands of devices deployed in a distributed manner. For instance, in a typical commercial building, there are hundreds of individual zones being monitored [26]. To ease installation and reduce deployment cost, these devices are often battery-powered, thus requiring manual battery replacement to sustain their tasks over time. However, at the scale of hundreds of sensor nodes, manual battery replacement is a time-consuming and expensive maintenance task. To extend the lifetime and avoid battery replacement, several past efforts have sought to create *immortal* devices by a combination of energy harvesting and batteryless design [2, 13, 39, 50].

Such energy-harvesting systems have to make a careful trade-off in sensing, communication, and computation to maximize utility with available energy [8, 20, 24, 35, 55]. These design trade-offs change depending on the hardware, application requirements and energy availability in the environment. Prior works either perform manual configuration or use heuristics to identify the operating points [9, 43, 54]. Manual configuration does not scale and heuristics do not generalize well to every context. To this end, several recent efforts have attempted machine learning for energy management of batteryless energy-harvesting sensors: the system understands environmental patterns, learns both energy and application needs, and creates a policy to maximize both performance and sensor node lifetime. Along these lines, many recent works [1, 3, 8, 16, 31, 56] exploit reinforcement learning (RL) [48] to automatically optimize the operation of sensor nodes under uncertain energy availability.

In RL, an agent interacts with an environment and learns to make optimal decisions through experience. The domain expert identifies the objective (i.e. reward function in RL terminology) and the inputs (i.e. state) that affect the decisions (i.e. actions) of the agent. The agent tries different actions, observes their corresponding rewards, and learns a policy that maximizes the long-term cumulative rewards. In the case of energy harvesting, the agent duty-cycles the

energy harvesting sensors (action) based on the energy availability and probability of a sensing event (state). The agent receives high rewards when it detects all events without depleting its energy. The RL policies are typically trained in simulation using historical data because algorithms converge faster and exploratory actions in simulations do not impact real-world performance [1, 14, 16, 31].

While these techniques have proven effective when historical data *is* available (e.g. 95% event detection rate [16]), their performance drops consistently whenever historical data are missing because the accuracy of simulations is predicated on representing the energy and event patterns in the environment. Relying on historical data makes the sensor scheduling susceptible to changes in environmental patterns. To make these sensors effective in the real-world, the deployed sensors should quickly adapt to any environmental conditions even without historical data. This consideration is critical for both newly deployed sensors that do not have any previous information about the environment and for sensors which environment can suddenly change (e.g., change in schedules, internal light malfunction). Ideally, the system should intelligently explore the environment to discover event patterns while preserving energy. We envision a day in the future when the time for building managers to get data feeds reduces to seconds. Waiting for 3-7 days may deter adoption in such a connected world, our work aims to push the state-of-the-art so that batteryless sensors send valuable data from the first day of deployment.

We build on prior data-driven approaches and propose methods to minimize reliance on historical data for duty-cycling energy-harvesting sensors. Our key observation is that instead of relying on collecting historical data for individual sensor nodes, we learn from data across multiple sensors in the environment. Simple schemes such as learning a single policy for all the sensor nodes deployment cannot achieve good performance [15]. Thus, we need to learn a separate policy for each sensor node, while still taking advantage of the data collected across sensor nodes. We present *Marble*, an approach that uses Meta Reinforcement Learning (Meta RL) for fast adaptation to *collaboratively* learn a general "meta-policy" by using data collected from sensor nodes deployed in the environment. The meta-policy is used to learn a specific policy for each sensor node. Marble also addresses the *cold start* problem: when a new sensor is deployed in an existing sensor ecosystem, the learned meta-policy is used to collect the location-specific data. After a few data samples are collected, Marble re-trains the agent, and the meta-policy is adapted to a specific policy for the newly deployed sensor.

To assess the effectiveness of our method, we deploy eleven sensor nodes in four different indoor locations (middle of an office, corridor, stairs and door) and collect real-world data such as temperature, humidity, pressure, light intensity, and people occupancy for two consecutive months. Using simulations with real-world data traces, we show that Marble detects up to 25.6% more events on average on the first 3 days of deployment and up to 10.6% after a month of deployment when compared to state-of-the-art techniques. In the staircase scenario where the energy availability is lowest in our deployment, the use of Marble increases up to 66% the number of environmental events detected in the first 3 days of deployment compared with the state-of-the-art.

2 RELATED WORK AND BACKGROUND

2.1 Related Work

Several energy-harvesting solutions have been proposed in the literature [2, 17, 18, 30, 47, 53]. Some methods deplete available energy as soon as it becomes available, e.g., harvesting AC power lines for energy metering [7, 57], RFID-based battery-free camera [40], thermoelectric harvesting-based flow sensor [33]. Backscatter sensors are a special case that eliminates communication-based energy expenditure by using existing RF signals for both communication and harvesting [10, 25, 50]. Time separation of energy availability from sensing activity improves quality of sensing [23, 27]. Several works address intermittent operations of batteryless energy-harvesting systems [18, 19, 30, 32]. Along these lines, this work addresses the problem of combined optimization of computation, communication, and energy use without having to stop operation of the node.

Minimizing manual configuration of sensor nodes is an important task to scale deployments as underlined by many works [3, 4, 22, 35, 52]. Adaptive duty-cycling of energy-harvesting sensors has been used to achieve energy-neutral operations [20, 24, 35, 55] where sensor nodes adjust their duty-cycle parameters based on the predicted energy availability to maximize lifetime and application performance [47]. Moser et al. [35] adapt parameters of the application itself to maximize the long-term utility based on future energy availability prediction.

Energy availability and environment events are sporadic and challenging to predict, which can reduce the quality of the applications running on these nodes. As an example, Campbell et al. used opportunistic sensing, i.e. sense whenever energy is available, to detect door events and only achieved 66% accuracy due to low energy availability [2]. Similarly, Pible detects only 32% of the PIR events in low energy conditions while using a rule-based approach for energy prediction [13]. To increase the event detection accuracy, a system should turn *On* the sensor right before an event happens to detect the event while saving as much energy as possible for later usage[16, 31]. To do this, a system has to learn to predict both energy availability and event occurrence patterns in the environment to make decisions that maximize overall performance. With this goal, many works [1, 3, 5, 16, 20, 35] use machine learning for duty-cycling decisions, and they can increase event detection accuracy to up to 95% while operating in an energy neutral manner [16].

Reinforcement Learning (RL) has been identified by several prior works as a means to configure wireless sensors and shown promising results [1, 8, 11, 31, 36–38, 58]. RL algorithms learn from trial-and-error experience given a high-level objective rather than labeled examples as in supervised learning. Therefore, with RL trained policies we can learn sensor-specific duty-cycling schedules that take into account the particular energy availability and event patterns that the sensor is exposed to. RL has been successfully demonstrated use in duty-cycling energy harvesting sensors in both outdoor [11, 45] and indoor conditions [14, 16, 31]. However, it is impractical to train RL policies using real-world interactions as trial-and-error actions lead to poor performance and convergence to a good policy can take months to years [14] of data. Therefore, it is common to use simulations based on real-world data traces to train the RL policy [1, 14, 16, 31].

Reliance on historical data makes it challenging to adopt RL-based scheduling policies in practice. The historical data needs to be reliable, and therefore, requires data collection from a temporary battery powered sensor. Use of temporary sensors increases deployment complexity considerably at scale (e.g. 1000s of sensors). If the RL policy is trained using an energy-harvesting sensor instead, it will miss events while collecting data because it is running a sub-optimal duty-cycling policy. Ember [16] addresses this problem by collecting data using a simple rule-based policy in conjunction with an RL-based policy that is re-trained on a daily basis. Their evaluation shows that RL policy performance gradually improves over the course of deployment without requiring manual intervention. However, performance suffers during the initial days of deployment. We evaluated the Ember algorithm and show that event detection accuracy is only 48.4% after 3 days of deployment in low energy conditions. The key limitation of Ember is that it learns a separate RL policy for each sensor that only relies on data collected by that sensor for training. It is common for sensors to be deployed en masse, and we can exploit the data collected by all the sensors to train the RL policy.

Fraternali et al. [15] exploit the same idea to scale energy harvesting sensor node deployments. They show that learning a single policy for all the sensors deployed leads to a degradation in performance as the RL algorithm has to learn to predict the energy and event patterns for all the situations accurately. Prediction of a single sensor node's event patterns is much easier as we need to learn a much narrower distribution. To improve performance, Fraternali et al. [15] cluster the sensor nodes that have similar energy availability and show that learning a policy for each cluster results in much better performance than the general policy. We build on this idea, but instead propose using Meta RL to learn sensor-specific policies. When we learn a cluster-specific algorithm, the data collected across sensors need to be split per cluster, which reduces the data available to train each policy. Therefore, large clusters will have more historical data to train the policy, which also increases the data distribution that the policy has to learn and thus degrades performance. Smaller clusters, on the other hand, will reduce historical data available and deteriorate performance. With the Meta RL approach, we can achieve the best of both: we learn a sensor specific policy that exploits all of the historical data available. In Meta RL, we train a meta-policy using the entire dataset that learns how to train a sensor-specific RL policy. We show that our Meta RL based solution outperforms prior approaches.

The main differences between our work and current state-of-the-art techniques are reported in Table 1.

2.2 Meta Reinforcement Learning

In RL, an agent interacts with an environment in a state *s*. The agent takes an action *a*, transitions to its modified state *s'*, and receives a reward *r*. The goal of the agent is to find a sequence of actions that maximize the cumulative long-term reward. Typically, the agent is trained to maximize rewards across a given number of steps, called a horizon. The probability of taking action *a* given state *s* as input is determined by the agent's policy. The agent starts from a random policy and collects data in tuples of *s, a, s', r*. After a number of interactions, the resulting dataset is used to update the agent policy. The updated policy is used to collect more data and this procedure is continued until convergence. Each policy update is referred to as an iteration. As it can take many iterations before a working policy is developed, meta-learning techniques have been proposed to speed up policy learning. With meta-learning, a variety of learning tasks are used to train a model that can be exploited to quickly learn new tasks using only a small number of training samples. Meta-reinforcement learning combines both meta-learning and RL so that an agent learns to solve unseen tasks fast and efficiently. For our problem, we use the well established Model-Agnostic Meta-Learning (MAML) algorithm [12] for few-shot learning via meta-learning. Other meta-learning algorithms are equally applicable to our problem. In MAML, different tasks are used to build a meta-policy such that a small number of gradient steps with a small amount of training data from a new task will produce good generalization performance on that new task. In our scenario, we consider a distribution over tasks $?(\mathcal{T})$ that we want our model to be able to adapt to. The meta-policy is trained to learn a new task specific policy $\backslash_{8}^{'}$ whose distribution is selected from $?(\mathcal{T})$. The meta-policy is represented by a function $5_{\backslash}$ with parameters $\backslash$. When adapting to a new task $\mathcal{T}_{8}$, the meta-policy trains a task specific policy with parameters $\backslash_{8}$. The updated parameter vector $\backslash_{8}^{'}$ is computed using gradient descent for a number of *inner adaptation steps* updates on task $\mathcal{T}_{8}$:

$$\backslash_{8}^{'} = \backslash - U\mathrm{O}_{\backslash}\, !_{\backslash_{8}}(5_{\backslash})\bullet \tag{1}$$

where U is the learning step size. The meta-policy parameters are trained by optimizing for the performance of $5_{\backslash_{8}'}$ with respect to $\backslash$ across all the tasks sampled from $?(\mathcal{T})$. The meta-optimization across tasks is performed via stochastic gradient descent (SGD), and the meta-policy parameters $\backslash$ are updated as:

$$\backslash \leftarrow \backslash - V\mathrm{O}_{\backslash} \tilde{\mathrm{O}}_{\mathcal{T}_{8}\sim?(\mathcal{T})} !_{\mathcal{T}_{8}}(5_{\backslash_{8}'})\bullet \tag{2}$$

where V is the meta step size. Each RL task $\mathcal{T}_{8}$ contains an initial state distribution $@_{8}(G_1)$ and a transition distribution $@_{8}(G_{C+1}|G_C\bullet O_C)$, and the loss $!_{\mathcal{T}_{8}}$ corresponds to the (negative) reward function $'$. The loss for task $\mathcal{T}_{8}$ and meta-policy $(5_{\backslash})$ has the following form:

$$!_{\mathcal{T}_{8}}(5_{\backslash}) = -\, \mathbb{E}_{G_C\bullet O_C\sim 5_{\backslash}@_{\mathcal{T}_{8}}} \left[\tilde{\mathrm{O}}_{C=1}^{} '_{8}(G_C\bullet O_C) \right] \bullet \tag{3}$$

where is the horizon that defines the length of an episode.

We use the MAML algorithm [12] as-is in our work. MAML has been used in many domains such as image recognition [46], robotics [34], neural architecture search [29] and speech recognition [21]. Our main contributions is to formulate the problem of duty-cycling energy harvesting sensors in the Meta RL framework. The novelty of our approach is not to simply apply an updated algorithm to an existing approach, our formulation alters prior approaches altogether to take advantage of data collected by all the sensors deployed in a system while training a sensor specific policy to improve performance in the initial days of deployment. The meta model can be created for as many sensors as are available, there are no limitations with respect to sensor type or deployment area. However, the effectiveness of the approach will decrease if

Table 1: State-of-the-art techniques vs Marble comparison.

Platform	No Intermittent Operations	Hist Data Not Needed	Cold-Start Supported	Collaborative Learning	Event Det [%] in Low Energy
Intermittent Comp [2, 10, 17, 41]		X			NA
SpotOn [31]	X				38.7
Transfer Learning (TL) [14, 15]	X		X	X	68-2 - 69.2
Ember [16]	X	X	X		48.4 - 76.4
Marble	X	X	X	X	74.0 - 87.4

Table 2: Sensor Event Definition.

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

the sensors are too different from each other. This difference can be measured with the change in distribution of the datasets. The reduction in performance will degrade gracefully and still be better than starting from scratch. Such performance degradation has been extensively studied in other domains [51]. To the best of our knowledge, we are the first to apply Meta RL for this problem and evaluate it using real-world data traces.

3 DESIGN OF MARBLE

3.1 Objective and Problem Formulation

We target indoor event-driven applications for buildings using batteryless energy-harvesting sensors powered with ambient light. The system should detect (i) motion using PIR sensors and (ii) environmental changes with regard to temperature, humidity, pressure, and light (referred to as THPL events). Table 2 reports the threshold values we use to define the events: when the PIR is powered, an event is detected whenever 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 the listed threshold values. When an event is detected, all sensors' data are transmitted. This design decision is due to the PIR being an event sensor whereas the THPL sensor data can be polled any time. If we miss the PIR event, there is no way to recover it. Therefore, we treat them separately. Another reason we combine THPL sensors is that the energy consumption of reporting the sensor data is dominated by BLE communication. Energy to transmit one sensor measurement is roughly the same as transmitting all four.

The objective of the system is two-fold: *(i)* duty-cycle the sensors to maximize the event detection rate while *(ii)* avoid 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 select indoor sensing locations that are generally subject to low energy availability, so that there is not enough energy for the sensors to stay always *On*. Therefore, the system is forced to learn a policy to duty-cycle the sensors to catch events while saving energy whenever possible. The locations that we target are *Stairs*, *Corridors*, *Middle of an Office*, and *Door* cases. In the former two cases, lighting is low in intensity but always On for security reasons. For the latter two cases (Middle and Door), the sensors can be exposed to both interior lighting and exterior natural lighting coming through the windows. We do not deploy sensors at locations such as *Windows* and *Interior Rooms*, because these do not present energy availability challenges: either we can leave the sensors always On considering the abundance of available energy near a window, while in the latter case there is no energy availability due to lack of activity with the ongoing Covid-19 restrictions. The proposed approach does not make any assumptions on specific activity or human behavior patterns. The event detection performance hinges on the predictability of the patterns. If the patterns are inherently random, no machine learning approach will succeed in predicting them. Furthermore, we do not make domain specific assumptions in our experiments and the proposed algorithm should generalize to other domains and deployment. We open sourced our code and data to encourage further research in this direction.

3.2 Marble for Batteryless Sensors

The **objective** of Marble is two-fold: (a) maximize event detection and (b) avoid energy depletion.

The **State Space** consists of information such as the state of charge of the *energy storage* (continuous value), intensity of *light* (continuous value), the current hour of the day (represented as one-hot encoded bit vector) and whether the current day is a weekday or weekend (one-hot encoded vector).

The **Action Space** consists two knobs controlled by the energy policy: (a) *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* the node polls all these sensors every minute.

The **State Transition:** Each action is taken after *State Transition* period, which we set to 60 minutes. By using a large *State Transition* time, the system can decrease the energy for communicating its action, but if the sensor is left Off for too long 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 to save energy. We determine 60 minutes as a balance between too much communication and granularity of intervention.

The **Reward Function** is +0.01 for each event detected, and -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

Table 3: Hyper-parameters and parameters used.

Hyper-Parameter	Value
Algorithm	MAML with Ray-RLlib [28]
State Transition	60 min
Discount Factor	0.99
Horizon ( )	24h
Max. Training Iters (#)	100
Inner Adaptation Steps	10
MAML Optimizer Steps	10
Inner Learning Rate (U)	1e-4
SGD Learning Rate (V)	1e-3
NN Model	2 hidden layers, 256 neurons each

Remaining hyper-parameters are left at default values in the Ray library (RLlib) [28].

reward. We opted for a large negative penalty because if the supercapacitor is depleted it can take several hours to charge, and no data can be transmitted during that time. To maximize the reward, the system has to find the best sequence of actions to catch as many events as possible while avoiding energy depletion.

3.3 Hardware and System Architecture

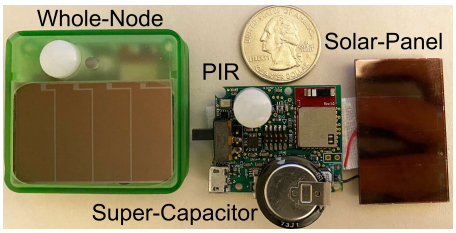

Figure 1: Energy-harvesting sensor node used in our study.

3.3.1 Sensor Node. for our experiments, we used a general-purpose energy-harvesting batteryless sensor node for indoor applications. It harvests energy from ambient light with a solar panel (AM-1454 [6]) and embeds the sensors: motion, temperature, humidity, pressure, and light. Bluetooth Low Energy (BLE) is used to relay data. The sensor node can store part of the harvested energy in a supercapacitor for later usage. The supercapacitor is 1.5F with 5.5V nominal voltage (Panasonic EEC-S5R5V155 [42]). When starting from full capacity, it can last up to 8 days by sensing and sending over 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 drops to 3V or lower. Figure 1 depicts our sensor node.

3.3.2 Deployment Architecture. Our wireless sensor network architecture includes three main parts: the sensors, the base stations, and the local server. 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 for training using Wi-Fi. RL training is performed on the local server, and the base stations are

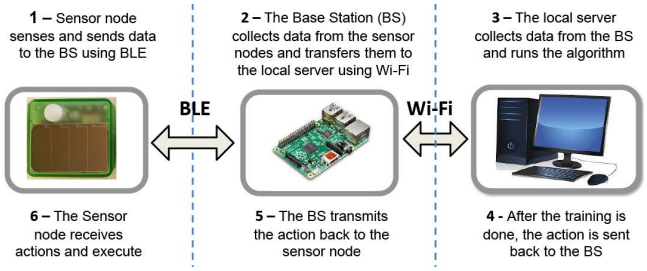

Figure 2: System architecture and communication process between the sensor node, the base station and local server.

only used to transmit 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. 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 (motion, light intensity, temperature, humidity, pressure). Figure 2 illustrates the communication steps. Even if many components in our architecture are wired (i.e. Base Stations), the importance of using batteryless sensor nodes is crucial to ease deployment. In our building, we have 60+ rooms on a single floor and we can get complete coverage with 10 base stations. So we can save powered sensors with a ratio of about 1:6 when we deploy one sensor per room. We would like to further increase density to 1 sensor per desk to detect fine-grained occupancy, and there are rooms with up to 10 desks. Another issue is that the ideal location of motion and temperature sensors need not be near a power source. Lack of reliable batteryless sensors limits such deployments.

The proposed algorithm is agnostic to the network protocol and architecture. It can be easily adapted to other settings.

4 METHODOLOGY

As a first experiment, we want to test if training a policy by using historical data from different locations can speed up, and improve the learning of a newly deployed node that does not have any information about the environment. Following that, we move towards a more challenging situation in which we simulate the deployment of a whole set of sensor nodes without historical data, and test if the learning of a node can be improved by exploiting the information gathered from all the deployed nodes. We perform our evaluation in simulations using real world data traces. We start with a brief explanation of our simulation setup.

4.1 Simulations

The simulator uses real-world data traces such as lighting and event data to simulate both the energy consumed due to sensing and communication, and the energy gained from harvesting. Table 4 reports the energy consumption breakdown of each of the components used in the board. Therefore, our simulator includes all the node energy consumption parts (node sleeping, waking up, sensor reading, BLE communication). The energy consumption information is extracted from the datasheet of each component in the platform. Not

Table 4: Current breakdown of the sensor node used. The energy consumption information is extracted from the datasheet of each component in the platform. For an accurate representation, we consider all the current consumed by the system to wake up, read all the sensors data, 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

surprisingly, transmitting data using BLE communication accounts for the majority of energy used.

The historical data we use - lighting and events - does not change with agent actions and is based on real-world data traces. Therefore, it is possible for us to generate counterfactuals using the simulator, i.e. we can compute the energy level of the sensor node and the number of events captured based on the agent actions.

In a real world setup, simulations are used to train the RL policy and deploy it to duty-cycle the sensors. In our evaluation setup, we have two levels of simulations. The low level simulation is the same as the simulator that is to train the RL policy. The high level simulation evaluates the proposed algorithm if they were to be deployed in the real-world. The two levels of simulations helps us compare algorithms in a fair manner using historical datasets without real-world deployment of sensors.

4.2 Cold-Start Learning

We train a meta-policy using all the historical data from known locations, and use it to drive a newly deployed sensor node. By doing so, we expect to have a better accuracy in event detection w.r.t. to driving a node without any previous information of the environment. Therefore, by cold start, we mean that the data for the specific location is not available. When we bootstrap learning with data from other locations, the performance is suboptimal and the agent performance improves as it collects data from that location.

To keep updating its policy to changes in the environment, the system uses a day-by-day learning approach as shown in Algorithm 1 where the the meta-policy and the RL policies are updated each day. Given a historical dataset , we train an RL policy using MAML in a simulator as described in Algorithm 2.

Once the policy is trained, we deploy it on the new sensor node (line 5 of Algorithm 1). The policy duty-cycles the sensor node given the environment state. We collect the interaction data in the form of (state, action, reward, next state) 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. We fine-tune the model with the new data, i.e., the re-training does not restart with an empty policy but the previously learned policy is restored and we backpropagate the gradients from the loss obtained with the new data [49]. Such fine-tuning 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

Algorithm 1: Day-by-Day Algorithm

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

Algorithm 2: MAML Simulator Algorithm

1: Initialize
2: **for** = 1 or till policy convergence **do**
3: Sample batch of tasks ∼ ( )
4: **for** All **do**
5: Sample K trajectories = ( ) using in
6: Evaluate ( ) in Eq 3
7: ′ = − ( )
8: Sample ′ = ( ) using ( ) in
9: **end for**
10: Update ← − ∑ ∼( ) (′)
11: **end for**

of the MAML algorithm used to generate our RL policy, details of which are described in Section 2.2 and the hyper-parameters in Table 3. The training ends once Algorithm 2 either converges or until it reaches a maximum of 100 training iterations. We say the training policy has converged when the episode reward changes by $< 3\%$ compared to the previous iteration's policy. With a day-by-day learning approach, the agent starts training using data currently available and runs till convergence (Figure 3).

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 executed on data *unknown* to the agent.

4.3 Collaborative Learning

Historical data might not be always available, especially if a set of sensors are deployed in a new building or if the environment changes suddenly (e.g. office re-allocation). For these cases, we propose a *collaborative learning* algorithm whose pseudo-code is reported in Algorithm 3. On day zero, as no previous data are available, Marble collects data using a random policy (line 1). As new data are gathered from the different locations it trains a meta-policy using all the available data using Algorithm 2 (line 3). After the training is complete, the system fine-tunes the policy by using only the data collected in a specific location (line 4) and once the training is done it uses the learned policy to collect the data (line 5). We

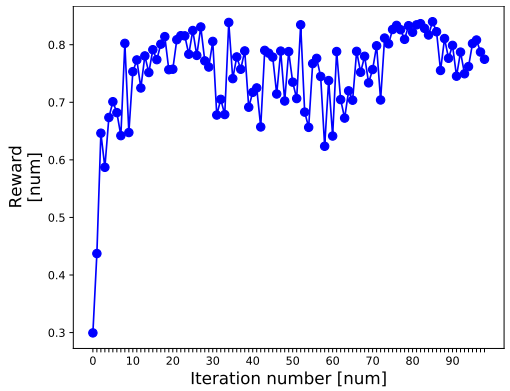

Figure 3: Convergence example: We deem the training policy has converged when the episode reward does not change by ± 3% compared to the previous day's policy or if it reaches a maximum of 100 training iterations. In this example, the policy converges at around 80 iterations.

fine-tune the meta-learned model with local data [49]. Both transfer learning and meta learning follow the same fine-tuning process. The improvement in performance is due to meta-learning from all sensors. Similar gains with meta-learning have been observed in several domains when compared to transfer learning [51]. We measured the overall training process to be ~30 mins wall-clock time using a standard i-7 CPU with 3.2GHz frequency. 15 mins are spent to train the meta-learning policy and 15 mins to fine-tune it.

Algorithm 3: Collaborative Learning Algorithm

1: On Day 0, collects data using a random policy
2: **while** True **do**
3: Trains a meta-policy using the data collected from all the deployed nodes using Algorithm 2
4: Fine-tunes the meta-policy using the data collected from a specific location
5: Deploys the policy and collect data for 1 day
6: **end while**

The key component of Marble that boosts performance compared to prior methods [15, 16] is Algorithm 2. The MAML algorithm trains a 'meta-policy' that is optimized for quickly adapting to a new task with few data points using fine-tuning. MAML trains task specific policy on a set of tasks using a common 'meta-policy', and computes the loss on test data with the trained policy. It then updates the 'meta-policy' based on this test loss using second order gradient updates. Therefore, unlike prior methods, the pre-trained network is optimized for fine-tuning with few data points.

5 EXPERIMENTAL RESULTS

5.1 Cold-Start Learning Results

To test if Marble can learn more effectively from multiple data sources, and correctly drive newly deployed nodes, we compare our system against (i) *Ember*[16] and (ii) *Transfer Learning (TL)*[14]. Ember uses a deep RL algorithm called proximal policy optimization [44] to learn a policy, and for a fair comparison we use the same input and actions spaces as our agent. To discover new events over time, Ember opportunistically keeps the sensors on with a simple round-robin schedule. MAML does not rely on such a heuristic, the exploration incentive is baked into the algorithm for fast adaptation. We use Ember as a baseline as it starts without any historical data. TL uses the same Ember algorithm, but it uses data from different locations to first learn a general policy, and then deploys it in the newly deployed sensor node. Both Marble and TL use historical data gathered from 5 sensor nodes deployed for a month to train a general policy. Each of the nodes is deployed at a different location (near a door, in the middle of an office, near a window, in a stair access and conference room).

Fraternali et al. [15] address a very similar problem, but we could not directly compare against their work because they synthetically created 1000 sensors while we only used real data from 5 sensors. If we did cluster, each of the 5 sensors are taken from distinct locations and would form their own cluster. Therefore, [15] would reduce to either Ember or transfer learning, the baselines we considered in the paper. Further, the algorithm in [15] shows benefits of clustering similar sensors together, but this clustering is done manually. We do not need to perform any manual steps with Marble.

When the policy converges, it is deployed to collect data for a sensor node deployed in a new location. All the systems use Day-by-Day learning (Algorithm 1), and the experiment is performed for 30 days after deployment. We repeat each experiment 5 times. Results are shown in Figure 4 and Figure 5. Figure 4-left reports the results obtained by deploying a new node in the middle of an office while collecting THPL events while Figure 4-right reports results of a node collecting PIR events in a different room. Note that the node on the left is subject to both internal and external lighting coming from a window, while the node on the right receives only internal lighting. Therefore, the energy patterns of the nodes on the left are more predictable. Figure 5 reports data while simulating a node deployed in stair access collecting both THPL and PIR events. All the results are reported with a rolling mean window of 3 days. From the figures we make the following observations:

(1) Marble outperforms both Ember and TL as it better exploits historical data from the other 5 locations and increases the event detection rate from the first day of deployment.
(2) In many cases, TL achieves even lower results than Ember. After carefully checking the data, we noticed that the general policy learned by TL before deployment is too conservative in turning-On the sensors to detect events as it prefers to avoid energy storage depletion.
(3) On Figure 4-left, after about 2 weeks, the event detection rate is similar for all the systems as the THPL events are easily predictable due to strong window lighting patterns.
(4) Whenever event patterns have higher stochasticity (e.g. THPL and PIR events in a stair access Figure 5), Marble increases the event detection accuracy as it makes better predictions.

We observe similar trends in Table 5 that reports the event detection average after 3 days, one week and one month of the experiment. Marble outperforms previous techniques in each conditions and detects up to 25.6% more events w.r.t. Ember in the stair access case (i.e. 74% vs 48.4%) which has particularly low level of lighting.

Figure 4: Cold-Start Learning Experiment. Le : results of a node deployed in the middle of an o ce while collecting THPL events while on the Right results of a node collecting data of PIR events. Both Marble and TL use 30 days of historical data from 5 locations to learn a meta-policy and then deploy to a new node. Results shows that Marble achieves better event detection rate w.r.t. the other methods especially in the rst days of deployment.

Figure 5: Cold-start learning results while collecting both THPL and PIR events in a Stair Access: Marble outperforms the other methods as it better exploits historical data from other locations to make better predictions.

Table 5: Cold-Start Learning Comparison (5 Runs Average).

	Event Det 3 Days [%] Em | TL | Ma	Event Det 1 Week [%] Em | TL | Ma	Event Det 1 Month[%] Em | TL | Ma
Middle/ Window (THPL)	76.4 | 68.2 | 87.4	81.4 | 79.3 | 88.6	91.6 | 90.3 | 93.7
Middle (PIR)	81.7 | 79.0 | 92.2	89.0 | 84.2 | 96.5	92.5 | 91.3 | 98.1
Stairs (PIR &THPL)	48.4 | 69.2 | 74.0	60.8 | 72.4 | 78.9	78.7 | 82.3 | 89.3

Legend: Em = Ember, TL = Transfer Learning, Ma = Marble

5.2 Collaborative Learning Results

For this experiment, we simulate the deployment of a set of 6 sensor nodes in 6 di erent locations without any historical data. The objective is to improve the performance of an arbitrarily chosen 6th sensor node. Marble exploits data collected from all the 6 nodes to rst learn a meta-policy, and once converged it ne-tunes the policy with only the data collected by the 6th deployed node. We compare the results obtained against Ember that does not use any historical data. Both the system use a day-by-day learning approach and we conducted the experiment for 30 days. Results of the experiment are shown in Figure 6 and Figure 7 using a rolling mean window of 3 days. We make the following observations:

(1) Exploiting collected data from multiple sensor nodes increases the event detection accuracy in all the three conditions from the rst day of deployment.
(2) From Figure 6-left, as THPL events are more predictable (i.e. lighting coming from a window), after 4 days Marble increases the event detection up to 99.4% w.r.t. 81.3% of Ember.
(3) In Figure 6-right and Figure 7 Marble result decreases after about a week w.r.t. Ember. After careful analysis, we noticed that Marble uses a lot of the energy storage available in the initial days to increase the event detection rate (100% on Figure 6-right) at the cost of turning-O the sensors more frequently later on to avoid energy depletion, and causing a drop in the event detection rate compared to Ember.

Table 6: Collaborative Learning Comparison (5 Runs Avg).

	Event Det 3 Days [%] Ember | Marb	Event Det 1 Week [%] Ember | Marb	Event Det 1 Month[%] Ember | Marb
Middle/Win (THPL)	76.4 | 75.9	81.4 | 92.6	91.6 | 94.2
Middle (PIR)	81.7 | 84.1	89.0 | 94.7	92.5 | 93.9
Stairs (PIR &THPL)	48.4 | 73.0	60.8 | 78.9	78.7 | 86.1

Similar observations can be made from Table 6 that shows the event detection average achieved by the two methods at di erent moments. On the stairs access case, event detection is improved up to 18.1% after 7 days of deployment as using data from other locations helps the policy in discovering more events.

5.3 Limitations and Future Work

We report the main limitations we encountered while executing our experiments and possible future improvements:

Same and Mulitple Sensors-Nodes Required. For the system to increase the event detection of a newly deployed sensor node, information from multiple locations are required. Furthermore, for the policy

Figure 6: Collaborative learning results: Le : Results of a node deployed in the middle of an o ce while collecting THPL events while on the Right results of a node collecting data of PIR events. By avoiding the use of historical data and by using only the data collected from 5 di erent locations, Marble detects more event w.r.t Ember in the rst days of deployment.

Figure 7: Collaborative learning results while collecting both THPL and PIR events in a Stair Access: Marble outperforms Ember for almost the whole duration of the experiment as it exploits data collected from the other locations to make better predictions.

to be e ective, the nodes have to share similar features such as similar hardware platform, energy consumption, goals, etc. The use of information from platforms/systems that are incompatible with the newly deployed nodes, could be ine cient and even degrade the performance achieved.

Malfunctions/Adversary Attacks: If one or more sensor nodes transmit incorrect data values due to hardware malfunctions or adversary-manipulation, the policy learned for newly deployed senor nodes will be a ected causing the damage to be propagated to more nodes. One solution could be to only use data collected from other nodes whenever local data are missing and rely on the local data once available.

Using all the Energy: In Figure 6-right, results decrease after 8 days of the experiment as the policy rst uses almost all the energy available and then turns O the sensor more after to recharge the energy storage. Such situation does not happen on Figure 4-right as the system uses historical data to catch the majority of the events while saving energy whenever possible. Reaching the energy voltage limit can lead to poor performance when there is a sudden change in the environment (e.g. light broken, people pattern change) that depletes the energy storage and stops operations. A solution is to increase the minimum voltage before the policy receives a negative reward giving the supercapacitor a margin of safety.

Energy Harvesting Range Limits: If a node is subject to either very low energy conditions or to an abundance of it, the node will systematically either deplete its energy storage by using the minimum operations or have the energy storage never depleted even by using the maximum operations allowed. In both cases, Marble will be useless. However, during our deployment, we found many strategic locations (i.e. corridors, doors) in which the amount of energy (i.e. light) availability was in between these two extreme conditions and the use of a learned policy to exploit the limited energy availability was still crucial to sustain reliable operations.

6 CONCLUSION

We present Marble, a meta-reinforcement learning-based system for duty-cycling batteryless energy-harvesting sensors for environmental monitoring. By exploiting data collected from other locations, Marble learns how to duty-cycle newly deployed nodes and increase the event detection accuracy of the sensors from the rst days of deployment. We have implemented and deployed sensing nodes. Results and simulations show that Marble outperforms state-of-the-art techniques even without historical collected data from other sensors. Our evaluation using real-world data traces shows that Marble detects on average up to 25.6% more events on the rst 3 days of deployment w.r.t. state-of-the-art techniques.

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] Fayçal Ait Aoudia at al. 2018. RLMan: An Energy Manager Based on Reinforcement Learning for Energy Harvesting Wireless Sensor Networks. IEEE Transactions on Green Communications and Networking (2018).
[2] 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-E cient Buildings (BuildSys '14). Association for Computing Machinery.
[3] Roy Chaoming Hsu, Cheng-Ting Liu, and Wei-Ming Lee. 2009. Reinforcement Learning-Based Dynamic Power Management for Energy Harvesting Wireless Sensor Network. In Next-Generation Applied Intelligence. Been-Chian Chien, Tzung-Pei Hong, Shyi-Ming Chen, and Moonis Ali (Eds.).
[4] Qingping Chi, Hairong Yan, Chuan Zhang, Zhibo Pang, and Li Da Xu. 2014. A Recon gurable Smart Sensor Interface for Industrial WSN in IoT Environment. IEEE Transactions on Industrial Informatics (2014).
[5] 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.
[6] Sanio Semiconductor CO. 2007. https://media.digikey.com/pdf/Data%20Sheets/Sanyo%20Energy/Amorphous_Br.pdf.
[7] 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 (Roma, Italy) *(SenSys '13)*. ACM, New York, NY, USA, Article 18, 14 pages. https://doi.org/10.1145/2517351.2517363

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

[9] echelon.com. 2018. https://www.echelon.com/.

[10] 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).

[11] R. C. Hsu et al. 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* (2014).

[12] Chelsea Finn, Pieter Abbeel, and Sergey Levine. 2017. Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks. arXiv:1703.03400 [cs.LG]

[13] 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.

[14] 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

[15] 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)*. https://doi.org/10.1145/3279755.3279760

[16] 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 *(SenSys '20)*. Association for Computing Machinery, New York, NY, USA.

[17] Josiah Hester, Lanny Sitanayah, and Jacob Sorber. 2015. Tragedy of the Coulombs: Federating Energy Storage for Tiny, Intermittently-Powered Sensors *(SenSys '15)*.

[18] Josiah Hester and Jacob Sorber. 2017. Flicker: Rapid Prototyping for the Batteryless Internet-of-Things *(SenSys '17)*. Association for Computing Machinery.

[19] 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 (SenSys '17)*.

[20] Jason Hsu, Sadaf Zahedi, Aman Kansal, Mani Srivastava, and Vijay Raghunathan. 2006. Adaptive Duty Cycling for Energy Harvesting Systems. In *ISLPED'06 Proceedings of the 2006 Int Symposium on Low Power Electronics and Design*.

[21] Jui-Yang Hsu, Yuan-Jui Chen, and Hung-yi Lee. 2020. Meta learning for end-to-end low-resource speech recognition. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 7844–7848.

[22] 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

[23] Hrishikesh Jayakumar, Kangwoo Lee, Woo Suk Lee, Arnab Raha, Younghyun Kim, and Vijay Raghunathan. 2014. Powering the internet of things. In *Proceedings of the 2014 international symposium on Low power electronics and design*. ACM.

[24] 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

[25] 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.

[26] Azam Khan and Kasper Hornbæk. 2011. Big Data from the Built Environment. In *Proceedings of the 2nd International Workshop on Research in the Large*. Association for Computing Machinery.

[27] Victor Lawson and Lakshmish Ramaswamy. 2015. Data Quality and Energy Management Tradeoffs in Sensor Service Clouds. In *Big Data (BigData Congress), 2015 IEEE International Congress on*. IEEE, 749–752.

[28] Eric Liang, Richard Liaw, Philipp Moritz, Robert Nishihara, Roy Fox, Ken Goldberg, Joseph E. Gonzalez, Michael I. Jordan, and Ion Stoica. 2018. RLlib: Abstractions for Distributed Reinforcement Learning. arXiv:1712.09381 [cs.AI]

[29] Hanxiao Liu, Karen Simonyan, and Yiming Yang. 2018. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055* (2018).

[30] Brandon Lucia, Vignesh Balaji, Alexei Colin, Kiwan Maeng, and Emily Ruppel. 2017. Intermittent computing: Challenges and opportunities. In *2nd Summit on Advances in Programming Languages (SNAPL 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

[31] Yubo Luo et al. 2019. Spoton: Just-in-time active event detection on energy autonomous sensing systems. *Brief Presentations Proceedings (RTAS 2019)* (2019).

[32] Kiwan Maeng, Alexei Colin, and Brandon Lucia. 2017. Alpaca: Intermittent execution without checkpoints. *Proceedings of the ACM on Programming Languages* 1, OOPSLA (2017), 1–30.

[33] Paul Martin, Zainul Charbiwala, and Mani Srivastava. 2012. DoubleDip: Leveraging Thermoelectric Harvesting for Low Power Monitoring of Sporadic Water Use. In *10th ACM Conference on Embedded Network Sensor Systems (SenSys '12)*.

[34] Nikhil Mishra, Mostafa Rohaninejad, Xi Chen, and Pieter Abbeel. 2017. A simple neural attentive meta-learner. *arXiv preprint arXiv:1707.03141* (2017).

[35] C. Moser, L. Thiele, D. Brunelli, and L. Benini. 2010. Adaptive Power Management for Environmentally Powered Systems. *IEEE Trans. Comput.* (2010).

[36] Abdulmajid Murad, Kerstin Bach, Frank Alexander Kraemer, and Gavin Taylor. 2019. IoT Sensor Gym: Training Autonomous IoT Devices with Deep Reinforcement Learning. In *Proceedings of the 9th International Conference on the Internet of Things*. Association for Computing Machinery.

[37] 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).

[38] Abdulmajid Murad, Frank Alexander Kraemer, Kerstin Bach, and Gavin Taylor. 2020. Information-Driven Adaptive Sensing Based on Deep Reinforcement Learning. In *10th International Conference on the Internet of Things (IoT '20)*.

[39] Saman Naderiparizi and et al. 2015. Self-localizing Battery-free Cameras. In *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing (UbiComp '15)*.

[40] 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.

[41] Matteo Nardello, Harsh Desai, Davide Brunelli, and Brandon Lucia. 2019. Camaroptera: A Batteryless Long-Range Remote Visual Sensing System *(ENSsys'19)*.

[42] Panasonic. 2018. https://media.digikey.com/pdf/Data%20Sheets/Panasonic%20Electronic%20Components/SG_Series_LowTemp.pdf.

[43] products.currentbyge.com. 2017. https://products.currentbyge.com/sites/products.currentbyge.com/files/documents/document_file/DT106-GE-Wireless-Occupancy-Sensor-Wall-Mount-Installation-Guide.pdf.

[44] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. (2017).

[45] Shaswot Shresthamali and et al. 2017. Adaptive Power Management in Solar Energy Harvesting Sensor Node Using Reinforcement Learning. *ACM Transactions on Embedded Computing Systems (TECS)* (2017).

[46] Jake Snell, Kevin Swersky, and Richard S Zemel. 2017. Prototypical networks for few-shot learning. *arXiv preprint arXiv:1703.05175* (2017).

[47] Sujesha Sudevalayam and Purushottam Kulkarni. 2011. Energy harvesting sensor nodes: Survey and implications. *IEEE Communications Surveys & Tutorials* (2011).

[48] Richard S Sutton, Andrew G Barto, Francis Bach, et al. 1998. *Reinforcement learning: An introduction*. MIT press.

[49] Nima Tajbakhsh, Jae Y Shin, Suryakanth R Gurudu, R Todd Hurst, Christopher B Kendall, Michael B Gotway, and Jianming Liang. 2016. Convolutional neural networks for medical image analysis: Full training or fine tuning? *IEEE transactions on medical imaging* 35, 5 (2016), 1299–1312.

[50] Vamsi Talla, Bryce Kellogg, Shyamnath Gollakota, and Joshua R Smith. 2017. Battery-free cellphone. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 1, 2 (2017), 25.

[51] Eleni Triantafillou, Tyler Zhu, Vincent Dumoulin, Pascal Lamblin, Utku Evci, Kelvin Xu, Ross Goroshin, Carles Gelada, Kevin Swersky, Pierre-Antoine Manzagol, et al. 2019. Meta-dataset: A dataset of datasets for learning to learn from few examples. *arXiv preprint arXiv:1903.03096* (2019).

[52] 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.

[53] Sennur Ulukus, Aylin Yener, Elza Erkip, Osvaldo Simeone, Michele Zorzi, Pulkit Grover, and Kaibin Huang. 2015. Energy harvesting wireless communications: A review of recent advances. *IEEE Journal on Selected Areas in Communicat* (2015).

[54] utc.com. 2018. https://www.utc.com/.

[55] 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.

[56] Christopher JCH Watkins and Peter Dayan. 1992. Q-learning. *Machine learning* 8, 3-4 (1992), 279–292.

[57] Thomas Weng, Bharathan Balaji, Seemanta Dutta, Rajesh Gupta, and Yuvraj Agarwal. 2011. Managing plug-loads for demand response within buildings. In *Proceedings of the Third ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings*. 13–18.

[58] Kok-Lim Alvin Yau and et al. 2012. Reinforcement learning for context awareness and intelligence in wireless networks: Review, new features and open issues. *Journal of Network and Computer Applications* (2012).