UtiliGEM: Energy Management Guided by Learned Application Utility

Hafiz Areeb Asad
Norwegian University of Science and Technology, NTNU
Trondheim, Norway
hafiz.a.asad@ntnu.no

Frank Alexander Kraemer
Norwegian University of Science and Technology, NTNU
Trondheim, Norway
kraemer@ntnu.no

Kerstin Bach
Norwegian University of Science and Technology, NTNU
Trondheim, Norway
kerstin.bach@ntnu.no

Bernd-Christian Renner
Hamburg University of Technology, TUHH
Hamburg, Germany
christian.renner@tuhh.de

Abstract

We present Utility Guided Energy Management (UtiliGEM), a novel energy management approach for wireless sensor nodes powered by energy harvesting. Central to the approach is the definition of a utility profile by the application, which enables time-varying application performance that can also evolve over time. This allows the utility-aware energy manager to allocate the energy budget more strategically, and application utility to be learned continuously, resulting in better adaptivity without requiring manual design effort. We show through extensive simulation that UtiliGEM increases application performance by up to 73% over the current state of the art and leads to a wider range of feasible device configurations.

CCS Concepts

• **Computer systems organization** → *Embedded and cyber-physical systems*; *Embedded software*; • **Computing methodologies** → Machine learning.

Keywords

Energy Management, Learning Application Utility, Self-adaptive Sensors, Batteryless Devices, Energy-Neutral Operations

ACM Reference Format:
Hafiz Areeb Asad, Frank Alexander Kraemer, Kerstin Bach, and Bernd-Christian Renner. 2024. UtiliGEM: Energy Management Guided by Learned Application Utility. In *14th International Conference on the Internet of Things (IoT 2024), November 19–22, 2024, Oulu, Finland.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3703790.3703796

1 Introduction

Energy management is crucial for energy harvesting devices because energy sources (like solar radiation) are not uniformly available and vary over time. Harvesting solutions therefore use energy buffers, like supercapacitors, so that an energy manager can decide to store energy for later instead of immediately using it. Ideally, an energy manager would take into account when there is useful work to do for the application, for instance, when there are relevant events to observe and application accuracy matters more. An example is to increase the sensing rates when values are expected to be in critical ranges [20]. Most existing approaches to energy management, however, do not take application utility into account or implicitly assume that it is uniform over time, in the sense that doing work now is as valuable as doing it later [6, 14, 15, 19, 21, 22, 25, 26]. Only a couple of recent energy management solutions consider time-varying utility [8, 11], so that applications can use more energy at certain times. But these approaches rely on static, manually developed utility profiles, assuming prior knowledge over the phenomenon to observe. Instead, we propose an approach that enables to learn the temporal aspect of utility dynamically. This reduces manual design effort, and also takes care of changes in utility over time in case there are changes in the environment or specific use case. In this setting, applications tell the energy manager when it would be beneficial to spend more energy, and the energy manager takes this knowledge into account when planning ahead.

Another approach towards energy management is approximate computing [5]. Here the idea is to reduce the energy consumption of an application and accepting lower accuracy but still doing useful work. An example is the use of machine learning models for inference that allow different levels of accuracy. When energy is scarce, inference uses fewer neural network layers, resulting in lower energy consumption while still producing acceptable results [7]. Applications can also learn about their environment to reduce the amount of work that needs to be done to achieve feasible results. In [3], we show how an application can learn a visual attention model in a sensor network to detect people using cameras, so that only those parts of an image need to be transmitted or processed that are most relevant. In contrast to the temporal utility described previously, we summarize these approaches under the term of *non-temporal* utility, since they usually do not affect *when* to allocate energy, but may lead to a reduction to how much energy is needed to achieve a certain application utility. However, current energy management approaches do not take such non-temporal utility into account.

Instead, we propose to use the concept of temporal and non-temporal utility for energy management in an explicit way. This allows for dynamic learning of utility and its incorporation into the energy allocation algorithm. As a result, energy management can

IoT 2024, November 19–22, 2024, Oulu, Finland

ACM ISBN 979-8-4007-1285-2/24/11
https://doi.org/10.1145/3703790.3703796

be performed more strategically, so that the resulting application utility is higher. In particular, we make the following contributions:

- We provide a novel definition for utility, which allows for a strategic allocation of energy.
- We show the learning of temporal utility using the concept of upper confidence bound (UCB) from reinforcement learning.
- We show an example for non-temporal utility learning using visual attention maps and its integration into the energy planning algorithm.
- We design and propose an energy allocation algorithm that takes the novel definition of utility into account.

Our case study shows that our approach increases application performance by up to 73%.

In the following, we provide a brief account of state-of-the-art solutions in Section 2 related to our work. In Section 3, we present the learning of application utility, while in Section 4 we present the UtiliGEM system model. Section 5 describes the proposed UtiliGEM planner. In Section 6, we evaluate UtiliGEM with state-of-the-art along with discussions and briefly conclude in Section 7.

2 Background and Related Work

Energy management can be broadly classified by two principles: intermittent computing [2, 4, 5, 9, 12, 17, 18, 24] and energy-neutral operation (ENO) [6, 14, 15, 19, 21, 22, 25, 26]. The two principles mainly differ in the amount of energy that they store in energy buffers. Intermittent computing systems try to consume harvested energy immediately and only buffer small amounts of it, if any. Tasks are executed intermittently as the available energy allows, either by breaking them down into atomic units that can be completed within one buffer cycle, or by storing progress in checkpoints. These systems are hence not ideal when harvested energy is unavailable or misaligned with the desired application utility, for example when solar energy is available at daytime but application utility is required at night.

To compensate the volatility of energy harvesting, ENO systems use comparatively larger buffers, and operate by the principle of consuming less energy than what is stored or harvested to avoid system failures. Kansal et al. [14] outline the foundations of ENO. The authors divided the day into time slots to use harvest predictions. These predictions, along with the buffer level, are then utilized to calculate the system's future energy consumption. However, in comparison to our work, their approach aims to find a single duty cycle for the finite horizon, assuming a uniform utility over the time. Renner et al. [21] proposed Depletion Safe (DS), a lightweight online load-adaptation algorithm that employs binary search to determine the maximum average node power consumption that avoids energy depletion over a finite horizon. Various other predictive works [6, 19, 22, 26] and reactive works [15, 25] have been proposed. However, these works are utility-agnostic, i.e., they do not consider time-varying utility.

As discussed, application utility may vary over time. The first energy manager to consider time-varying utility is PreAct [8]. The authors show a case of long-term energy-management of 1 year with time-varying utility, where a rechargeable battery is used. They assume a linear system for calculating the state of charge (SoC) based on the given temporal utility. They then employ a PID controller to track the SoC and capture non-linearities and deviations.

As outlined in [11], PreAct performs sub-optimal with small buffers over medium-term horizon of one day. EmRep [11] focuses therefore on small energy storage with medium-term energy management of up to one day. The authors introduced a notion of time-varying utility as a percentile of the average harvest. Moreover, EmRep reduces early saturation of the buffer by decoupling high-intake from low-intake phases. This approach increases the use of harvested energy over DS.

Both PreAct and EmRep require time-varying utility to be manually designed before deployment, which then also remains static for the whole deployment period. The authors suggest that domain experts can help define the utility profile, indicating which hours or days the utility should be high. In contrast, UtiliGEM is designed to learn the temporal utility autonomously, without prior data.

Another important aspect of application utility is that it varies with energy consumption. Many applications have adaptive tasks that can produce outputs even with partial execution or sensing rates can be adapted. The utility of these applications varies with the level of task execution and corresponding energy consumption. For instance, Farina et al. [7] show how TinyML machine learning models can be defined with multiple exits and hence adapt their execution in terms of computation effort and achieved accuracy. Other works [5, 12, 18] mostly target intermittent-computing systems and exploit this characteristic for scheduling inference tasks based on energy or employing approximate computing to trade-off accuracy with energy and enable more events application processing. In our work, we combine the idea to enable temporal utility from PreAct and EmRep with the adaptivity of approximate computing.

HarvSched [13] is a learning-based scheduler that uses Q-learning to maximize inference accuracy by determining the model's exit-layer (amount of computation) while ensuring energy-neutral operation. Their approach is reactive, making decisions about energy usage at the time of the event detection, based on runtime factors. Additionally, their approach either requires pre-deployment training of the agent to learn harvesting patterns to ensure ENO or necessitates a cumbersome trial-and-error process initially, which fails to ensure ENO at the start. In contrast, our approach is predictive. We anticipate event occurrences, consider temporal utility, and allow the energy manager to account for improved model accuracy over time.

3 Learning Application Utility

As use case, we consider the application of counting skiers in a cross-country skiing area, as detailed in [3]. Wireless camera devices capture images and transmit them to a central server, where a visual recognition algorithm counts the number of people. The devices are powered by solar panels, and the energy manager we focus on here determines how much of the images are transmitted and when. As said, the energy managers are controlled by application utility. We decompose application utility into two distinct components: *temporal utility*, i.e., *when* it is useful to spend more energy, and *non-temporal utility*, i.e., to *which extent* the use of energy improves application performance. In the following, we discuss how these

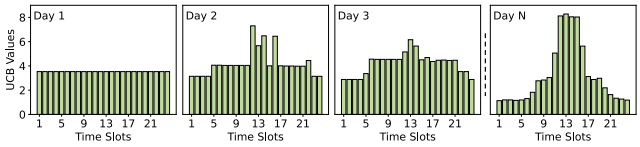


Figure 1: Evolution of time-varying utility learning

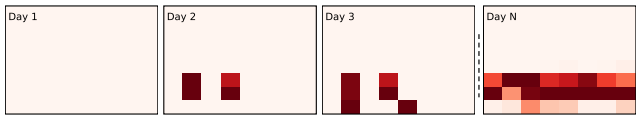


Figure 2: Evolution of visual attention map learning

components can be learned and then present our definition for utility.

3.1 Learning the Temporal Utility

To learn the temporal utility, we use the upper-confidence bound (UCB) method from reinforcement learning [23]. The UCB algorithm keeps track of how many times a certain time slot has been spent energy on and selected for sensing and its corresponding reward. If sensing at a certain time slot is done less frequently (i.e., less energy is spent), its uncertainty increases, leading to a higher UCB value for that time slot. This encourages exploration of less-visited time slots. In contrast, a greedy approach would consistently select the time slots that initially provided the highest rewards, ignoring the potential of less-explored time slots. Thus, UCB balances between exploiting historically high-reward periods and exploring lesser-visited ones. As more data is collected, UCB adapts and becomes more confident in its decisions. UCB values are calculated using the following equation:

$$\text{UCB}_t = \text{mean reward}_t + c \cdot \sqrt{\frac{2 \cdot \ln(\text{day})}{\text{counts}_t}} \quad \forall\, t \tag{1}$$

In the considered use case, the mean reward is calculated as the total number of persons detected in specific time slots divided by the total number of times those time slots have been observed. This can be generalized to interesting events detected, making the approach adaptable to other use cases.

The second term on the right-hand side of Eq. (1) handles exploration. Here, c is a constant, with higher values of c emphasizing exploration. In our case, c is set based on an estimated value. As days progress, the logarithm of time (in days) grows at a decreasing rate, ensuring that as more data is collected, the algorithm becomes more confident in its estimates and reduces the exploration term. $counts_t$ represents the cumulative count for a specific time period up to the current day. This count is not just a simple visitation count but is weighted by the energy spent during those visits, reflecting both the frequency of visits and energy expenditure.

Figure 1 illustrates the evolution of UCB values. Initially, all UCB values are similar, indicating that all time slots appear equally rewarding since no time slot has been explored yet. As the time slots are explored over the days, the algorithm becomes more confident about which time slots are more rewarding. For our use case, we use the UCB values to represent the temporal application utility.

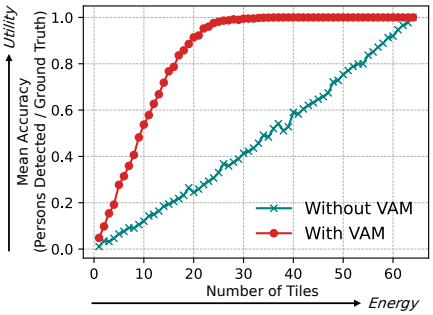


Figure 3: Mean accuracy vs number of tiles with and without the visual attention map (VAM)

3.2 Learning the Visual Attention Map

Cameras may cover areas like the sky where it is unlikely to detect people. Therefore, sending the entire image is unnecessary. The areas where people are concentrated can be learned, known as *visual attention map*. In [3], we show how the visual attention map can be learned through a multi-armed bandit ϵ-decreasing approach. In this work, we employ a context-aware approach where exploration is correlated with energy: higher energy levels lead to increased exploration, while lower energy levels results in higher exploitation. Figure 2 illustrates the evolution of the attention map over several days. On day 1, the sensor does not know which regions contain people and all tiles have the same value. Over time, the attention map becomes more refined and the application needs to transmit fewer (but selected) tiles to reach on average the same accuracy.

Figure 3 shows the achieved accuracy in terms of persons detected (i.e., application utility) over the number of tiles sent, which corresponds to the utilized energy. These graphs were produced with the simulator introduced in Section 4. Without the visual attention map (VAM), tiles are selected at random, and there is an almost linear relationship between energy spent and achieved utility. With the learned visual attention map, high application utility can, on average, be achieved by sending much fewer tiles. This shows how learning the attention model helps reducing the energy consumption. In this setting, excess energy can be used for exploration to constantly update the visual attention map.

3.3 Defining Utility Profiles

We first examine how EmRep and PreAct define and interpret temporal utility, offering insights that highlight the need for new utility semantics. Both EmRep and PreAct take user-defined temporal utility values as input, ranging from 0 (low) to 1 (high) for all time slots. These values are then mapped to the expected energy intake, in case of PreAct the daily average or for EmRep the long-term average. However, achieving an average harvest intake as consumption does not reflect the application's required budget, leading to ineffective

energy allocation. Additionally, knowing the average solar intake in the long run requires prior knowledge, which is often not available.

EmRep also suggests a more application-specific approach by mapping utility values directly to application consumption instead of average harvested energy. But estimating exact useful application consumption values for all time slots beforehand remains challenging. In Section 6, we show limitation of EmRep's utility-to-application consumption mapping approach, which results in resource wastage albeit energy surplus. Thus, the approach of translating temporal utility values directly to consumption simplifies energy management but has drawbacks.

If we look at the temporal utility learner values (as in Figure 1), they indicate which time slots should be considered for sensing based on which time slots have high rewards or are less explored. Hence, learner values cannot be fed to PreAct and EmRep and interpreted directly as application consumption. Therefore, we introduce UtiliGEM and a different semantic of utility as described next.

We define utility u_t for a time slot t as a quadruplet $(e_{min,t}, e_{max,t}, e_{req,t}, u_{pri,t})$. A utility profile $U_{T_0}^{T_n} = \{u_t | T_0 \leq t \leq T_n\}$ describes the expected utility for consecutive time slots t.

- $e_{min,t}$ describes the minimum energy budget within a time slot that is useful for the application. Lower energy allocation would mean that the device cannot do any useful work. This can be associated with device sleeping mode consumption, which remains constant for the whole deployment.
- $e_{max,t}$ describes the maximum energy budget within a time slot that can be used. Higher energy allocation adds no additional benefit. In most cases, the maximum application consumption also remains constant during deployment.
- $e_{req,t}$ describes the energy required for achieving sufficient application utility, taking into account any benefits from potentially learned non-temporal utility as described in Section 3.2.
- $u_{pri,t}$ describes the potential utility of the different time slots. Temporal utility learner values are interpreted as a priorization between the time slots. Higher values of UCB indicate that slots should be prioritized when allocating energy. Time slots with higher priority should be allocated with energy (until $e_{max,t}$ is reached) before lower-priority time slots.

Note that $e_{min,t} < e_{req,t} < e_{max,t}$. The priority semantics implied by $u_{pri,t}$ align with the temporal utility learning approach, ensuring that more rewarding or less visited time slots receive energy first, which particularly is important when energy is scarce.

When only temporal utility learning is used and the energy-to-utility relation remains constant, UtiliGEM follows $u_{pri,t}$ for energy allocation. When there is also non-temporal utility learning and the application can achieve the same accuracy with lower energy expenditure, $e_{req,t}$ takes care of this. UtiliGEM will then initially use $u_{pri,t}$, allocating energy until $e_{max,t}$ to facilitate exploration. Once the non-temporal utility matures, UtiliGEM ensures that all time slots receive their necessary energy ($e_{req,t}$) first and then allocates any available energy up to $e_{max,t}$ in order of $u_{pri,t}$.

4 UtiliGEM System Model

Our system model builds on existing work [11, 21] which also established its correspondence with realistic hardware. As shown in Fig. 4, the model consist of two main parts: energy generation and energy consumption. The first part includes a harvester, a supercapacitor, and a boost converter. The second part typically includes a microcontroller unit (MCU), radio and sensors.

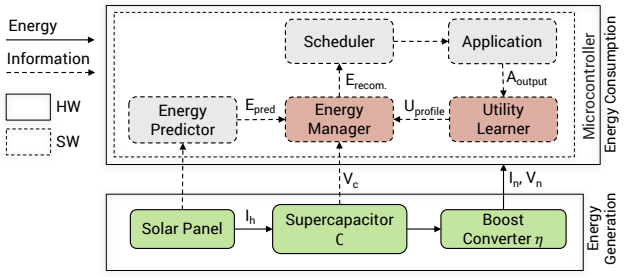


Figure 4: UtiliGEM System model

The supercapacitor receives its charge from the power generated by the harvester. To simplify the charging circuitry and reduce complexity, costs and size, a direct charging circuit is used. The solar cell's current remains almost constant across varying terminal voltages, enabling accurate intake forecasts. Further, the sensor node is connected with the storage-system via a boost converter to avoid fluctuations as it impacts the clock frequency of MCUs and consequently affects the duration of the code [1]. The boost converter ensures a constant voltage of V_n to the node with a power conversion efficiency of η [10].

The software stack comprises five modules: utility learner, predictor, scheduler, application, and an energy manager. The utility learner leverages the application's output results to identify rewarding time slots and tiles, creating a $U_{T_0}^{T_n}$ profile. The predictor estimates future harvest intake for a planning horizon. We employ a base predictor similar to the ideal predictor, averaging time slots for hourly predictions. However, this prediction has inherent errors due to its lower resolution compared to the ideal harvest. Both the utility profile and harvest predictions are fed to the energy manager together with the current state of charge of the buffer. The energy manager's ability to sustain the node's operation relies on accurate predictions of future harvest intake. The utility profile $U_{T_0}^{T_n}$ provided by the utility learner informs the energy manager on where to allocate resources.

The energy manager's primary goal is effective resource allocation based on $U_{T_0}^{T_n}$, while ensuring sustained node operation within a finite horizon. Consequently, the energy manager determines the available resources for the next time slot. The scheduler then utilizes this information to allocate tasks. In our case, we employ a classic scheduler that schedules six tasks per hour (approximately every ten minutes). These tasks are adaptive, meaning that even with limited resources, the scheduler still schedules them, albeit with reduced number of tiles and execution time.

To achieve energy-neutral operation or sustain a device for a predefined period, an energy manager must allocate no more energy than it harvests. Consequently, it becomes crucial to calculate the state of charge (SoC) profile for a supercapacitor. Specifically, the energy manager anticipates the future voltage course to ensure it remains above the minimum threshold for device operation.

Equation (2) represents the system model composite formula for calculating the voltage as proposed by [21].

$$C \cdot \dot{V}_c = I_h - \frac{I_n \cdot V_n}{V_c \cdot \eta(I_n, V_c)} \tag{2}$$

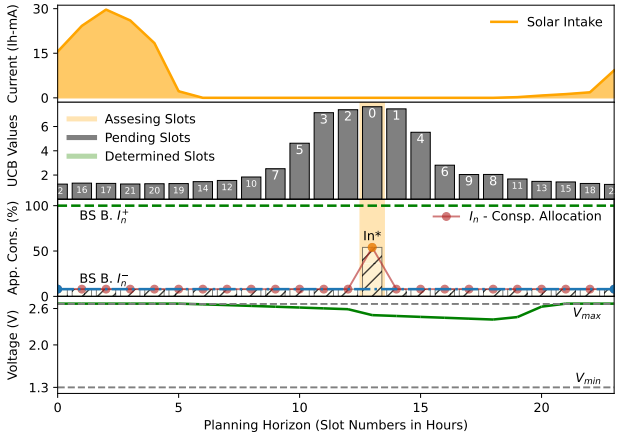
The model considers the non-linear energy-flow approach due to direct charging. In this model, C represents the capacitance of the supercapacitor and V_c is its voltage. I_h is the current supplied by the harvester, and $(I_n \cdot V_n)$ is the output power of the boost converter converted with efficiency η. However, predicting the time-dependent behavior of the capacitor voltage V_c with this equation is not feasible, as equation cannot be solved explicitly for V_c. Further, there is an overcharging protection and a cut-off voltage. The model does not have an explicit solution, but it is approximated with Newton's method assuming I_n and I_h piece-wise constant as described in [21].

5 Utility-Aware Energy Planner

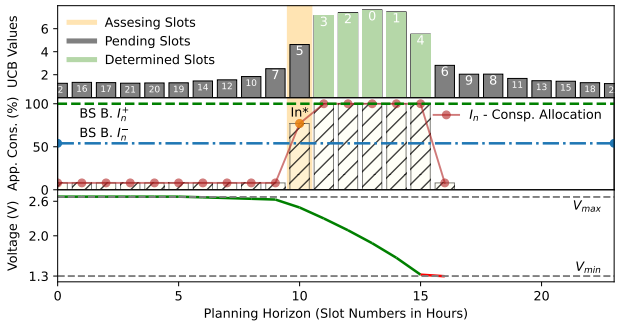
The energy manager works over a planning horizon, which we select to be 24 time slots of 1 hour in length, covering one diurnal cycle of the energy harvester. It tries to allocate the maximum energy to each time slot while keeping the energy buffer above a certain level to avoid depletion. As input, the energy manager takes the expected energy intake for each time slot, the projected state of the energy buffer, and, of course, the utility profile provided by the application. Since we cannot calculate the total available energy to distribute beforehand (see Sect. 4), we need to follow an iterative approach. Here, the energy manager first allocates energy to the different time slots and then simulates the resulting effect on the energy buffer. If the voltage V_c of the buffer falls below the minimum voltage V_{min}, the energy manager allocated too much energy, risking device shutdown. Otherwise, more energy can be allocated. For taking the utility profile into account, the sequence in which energy is allocated to the time slots is crucial.

Figure 5a illustrates the first iteration of the energy management process. The x-axis represents the planning horizon in hourly time slots. The first subplot shows the average hourly harvest prediction, the second subplot displays the UCB values of each time slot, marked according to their priorities, where a higher UCB value indicates higher priority. We show a case where solar harvest is not aligned with the temporal utility, which means that energy is available when there is less to observe and the effect of an energy manager is most critical. The third subplot shows the energy allocation for all time slots the energy manager is working on. Initially, they are all set to the minimum, which corresponds here to the sleep mode consumption.

The energy planner starts with the time slot that has the highest utility value $u_{pri,t}$, marked here with the highest priority 0. Performing binary search, it allocates the mean between the lowest and the highest value $I_n^* \leftarrow (I_n^+ + I_n^-)/2$. The energy planner then simulates the operation based on the current energy allocation. In this case, the voltage does not fall below V_{min}, indicating that the current allocation is valid, as illustrated in the fourth subplot. The energy manager increases therefore the allocation for that slot. The binary search is stopped when the difference between the upper and lower bounds is below a specified absolute tolerance. This tolerance limits the maximum number of iterations given by $log_2(I_n'^+/tol)$, where



(a) First iterative process of energy manager



(b) Iterative adjustment process for energy allocation

Figure 5: UtiliGEM's iterative processes for energy allocation

$I_n'^+$ represents the maximum upper bound. This process is repeated for each time slot, starting with the highest priority slot and moving to the next, ensuring that energy is allocated first to slots with high utility.

Figure 5b shows the state of the energy management algorithm when allocating energy to slot 5. We see that the slots before could all receive the maximum available energy consumption. The budget allocated currently to slot 5, however, causes the voltage to drop below V_{min}. The energy manager will therefore allocate less energy to that slot in the next iteration. Note that we do not reduce the budget of slots with already determined energy allocations.

In the approach explained above, we only allocated energy to single time slots at a time, called a *single-raise strategy*. However, this approach has a drawback: time slots with similar utility may not receive equal allocations. As a result, if resources are insufficient, the first slot in the sequence receives more allocation than the subsequent ones, leading to unequal energy distribution.

Alternatively, we can use a *multi-raise strategy*, which allocates energy to time slots with similar utility *simultaneously*. This ensures an equal allocation of resources to time slots that share similar utility. But also this strategy has a limitation: A time slot t_x may receive less energy when it is raised together with another time slot

Algorithm 1: EM UtiliGEM: Perform Binary Search

Data: Temporal utility $u_{pri,t}$, Number of slots $nS = 24$, Current voltage V_c, Slot length in seconds $l = 1$ *hour*, Initial load for all slots I_n, Application load per slot I_{app}, Allocation strategy: multiRaise | singleRaise
Result: Recommended Consumption for Slots

1 $I_n'^{-} \leftarrow$ Sleep mode consumption per slot I_{sleep}
2 $I_n'^{+} \leftarrow$ Application load per slot I_{app}
3 Sort. Prior. Slots Index: $PS'_{\text{indexes}} \leftarrow$ `sortSlots(` $u_{pri,t}$`)`
4 Determined Slots: $DS \leftarrow []$
5 Assessing Slots: $AS \leftarrow []$
6 **foreach** PS'_{index} *in* $PS'_{indexes}$ **do**
7 **if** $PS'_{index} \in Determined\_Slots$ **then**
8 **continue**
9 **if** *multiRaise* **then**
 `// Get similar temporal utility slots`
10 $AS \leftarrow [i \mid i \in PS'_{\text{indexes}}$ and $|PS'[i] - PS'[PS'_{index}]| \leq tol_u$
11 **else**
12 $AS \leftarrow PS'_{index}$
13 **end**
14 $I_n^{-} \leftarrow I_n'^{-}$
15 $I_n^{+} \leftarrow I_n'^{+}$
16 **while** $abs(I_n^{+} - I_n^{-}) \leq tol_b$ **do**
17 $I_n^{*} \leftarrow (I_n^{+} + I_n^{-})/2$
18 $v \leftarrow V_c$
19 $BinarySuccess \leftarrow True$
20 **forall** *slot in nS* **do**
21 **if** $slot \in AS$ **then**
22 $v \leftarrow$ `simulateVc(`$v, I_n^{*}, I_{h,slot}, l_{slot}$`)`
23 **else**
24 $v \leftarrow$ `simulateVc(`$v, I_{n,slot}, I_{h,slot}, l_{slot}$`)`
25 **end**
26 **if** $v < V_{min}$ **then**
27 $BinarySuccess = False$
28 **break**
29 **end**
30 **if** $BinarySuccess == True$ **then**
31 $I_n^{-} \leftarrow I_n^{*}$ `// Update BS lower bound`
32 **else**
33 $I_n^{+} \leftarrow I_n^{*}$ `// Update BS upper bound`
34 **end**
35 **end**
36 $I_{n,slot} \leftarrow I_n^{-}, \quad \forall\, slot \in AS$
37 $DS \leftarrow slot, \quad \forall\, slot \in AS$
38 **end**
39 return I_n

Algorithm 2: EM UtiliGEM: Hybrid Approach

Data: Application max load per slot $I_{\max}$ ($e_{max,t}$), Sleep mode consumption per slot I_{sleep} ($e_{min,t}$), Required application load per slot I_{req} ($e_{req,t}$)
Result: Recommended Consumption for Slots

1 $I_{n,slot} \leftarrow I_{\text{sleep}}, \quad \forall \quad slot \in$ number of slots
2 **if** $I_{req} < I_{max}$ **then**
3 $I_{n,mR.} \leftarrow$ `performBS(`$args^{*}, I_n, I_{req}, multiRaise$`)`
4 $I_{n,res.} \leftarrow$ `performBS(`$args^{*}, I_{n,mR.}, I_{max}, singleRaise$`)`
5 **else**
6 $I_{n,mR.} \leftarrow$ `performBS(`$args^{*}, I_n, I_{max}, multiRaise$`)`
7 $I_{n,res.} \leftarrow$ `performBS(`$args^{*}, I_{n,mR.}, I_{max}, singleRaise$`)`
8 **end**

t_y, if t_y causes a buffer depletion, which in the multi-raise strategy leads to the allocation for both time slots to be reduced.

To mitigate the issues caused by using either the single-raise or multi-raise strategy, we use a *hybrid strategy*. First, we perform a binary search to identify the optimal allocation level using the multi-raise strategy. Then, we try to increase the allocated energy to individual slots further using the single-raise strategy. This hybrid strategy leads to a good trade-off between a more balanced and maximum energy distribution.

Algorithm 1 outlines the step-by-step procedure. Lines 1–5 initialize the necessary variables. Line 6 indicates a loop to iterate through slots according to their priority. In lines 9–13, if the multi-raise strategy is chosen, slots with similar utility (i.e., within a 5% tolerance, tol_u) are marked as assessing slots for performing binary search. Otherwise, the current slot in highest priority list is marked as assessing slot. Lines 16–35 describe a loop that performs binary search on assessing slots, updating BS boundaries I_n^{+} and I_n^{-} accordingly. Finally, line 36 sets the result of the binary search.

As shown in Figure 3, after the convergence of the non-temporal utility (the visual attention model), less energy is required to achieve high accuracy. As described earlier, UtiliGEM takes into account any lower energy requirements $e_{req,t}$ from the the utility learner. The energy manager then distributes the newly required amount of energy (e.g., up to 50%) across all time slots. Any additional energy, from 50% to 100%, is still allocated. While this might not improve accuracy, it helps explore and enhance the temporal and non-temporal model where energy surplus is available.

Algorithm 2 shows the allocation strategy before and after non-temporal convergence. Before convergence (lines 6-7), UtiliGEM allocates energy up to the maximum application load ($e_{max,t}$), aiming to allocate 100% to all time slots using multi-raise strategy, then refining the allocation with single-raise strategy. Once the application energy requirements gets lower, (in lines 3-4) UtiliGEM first allocates energy up to the new requirement $e_{req,t}$ and then distributes the rest up to $e_{max,t}$ using the single-raise strategy. Given that energy = *current*·*voltage*·*time*, and with voltage being constant as provided by the boost converter and time slot being constant at 1 hour, here UtiliGEM performs binary search on the current I values. Alternatively, it can be applied directly to the energy.

6 Evaluation and Discussion

The effect of an energy manager depends on the overall dimensioning of the device, which are here mainly the size of the energy buffer and solar panel. If the buffer is too small, not enough energy can be transported from slots with energy surges to those with

little energy, and energy manager cannot act strategically. Or, if the solar panel is so large that it always produces excess energy, energy management becomes less significant. For a meaningful evaluation of the energy manager we therefore examine a wide range of device configurations in terms of buffer and solar panel size. For this, we use the simulator introduced and validated in [10]. We converted the simulator from C++ to Python and support the execution of different configurations in parallel to speed up execution time.

For our experiments, we use the same real-world solar traces as those employed by EmRep, recorded at 3-second intervals and averaged to a 5-minute resolution for simulations. As discussed in Section 4, we provide the energy managers with energy predictions at a 1-hour resolution for a planning horizon of 24 time slots. The energy managers are executed hourly, replanning for the next 24 time slots each time, to ensure continuous device operation and accommodate prediction errors. The hardware parameters were kept mostly the same as EmRep's, including a boost converter providing a stable voltage of 3.3 V with an efficiency of 78% to 93%, a supercapacitor voltage up to 2.7 V, a minimum device voltage of 1.3 V, and a turn-on voltage of 1.6 V after shutdown. The simulator also models device shutdown scenarios (no consumption, i.e., I_n = 0 mA but no loss of memory).

We utilize a simple scheduler similar to EmRep's. It assumes the program consists of a set of activities, treating the program as a single entity as tasks and considering average duration $Task_{duration}$ and consumption $Task_{consp}$. In our case, the scheduler manages 6 tasks per hour, uniformly distributing energy across the tasks. The values of $Task_{duration}$ and $Task_{consp}$ need to be obtained through in-lab measurements of real-hardware and then utilized into the simulator. However, for our experiments, we used estimated values for these parameters. We simulate the operation for 80 days. The complete code of our experiments is publicly available[1]

6.1 Scenario Comparison with Utility Profiles

Figure 6 shows the recommended consumption allocations by the energy managers DS, EmRep and UtiliGEM, evaluated with two distinct utility profiles.

DS results in equal consumption allocation across all time slots. This is because DS does not take the utility profile as input, assuming uniform utility over the planning horizon. Although DS also employs binary search, it does so once for all time slots together, i.e., like an extreme version of the multi-raise approach explained in Sect. 5. This limits the maximum allocation to all timeslots.

In the first plot, we show a case where EmRep and UtiliGEM are provided with a learned temporal utility profile (normalized to 0 to 1, with 1 indicating high utility). EmRep maps 1 to the maximum application requirement, treating it as actual utility, and thus fails to allocate more resources to time slots with energy surplus. This interpretation of utility values as consumption requirements is impractical. This limitation also prevents EmRep from being used with the utility learner. Additionally, a manually defined user profile remains static and fails to utilize resources during surplus phases.

For a fair comparison with DS and to highlight EmRep's shortcomings, EmRep and UtiliGEM are also provided with a maximum

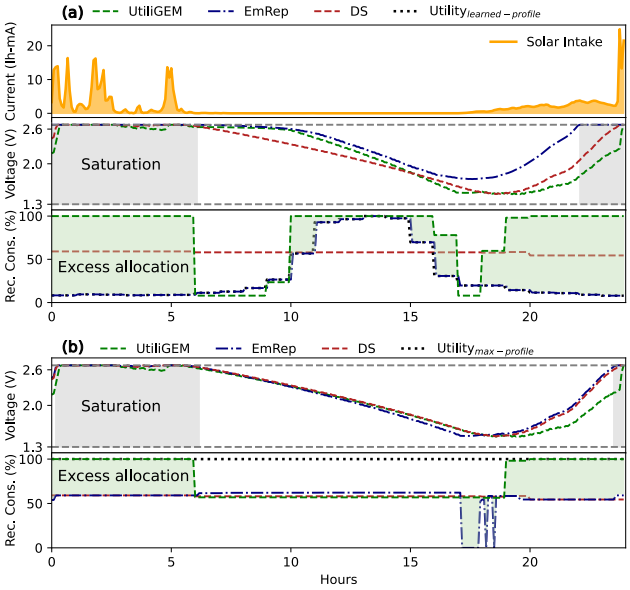


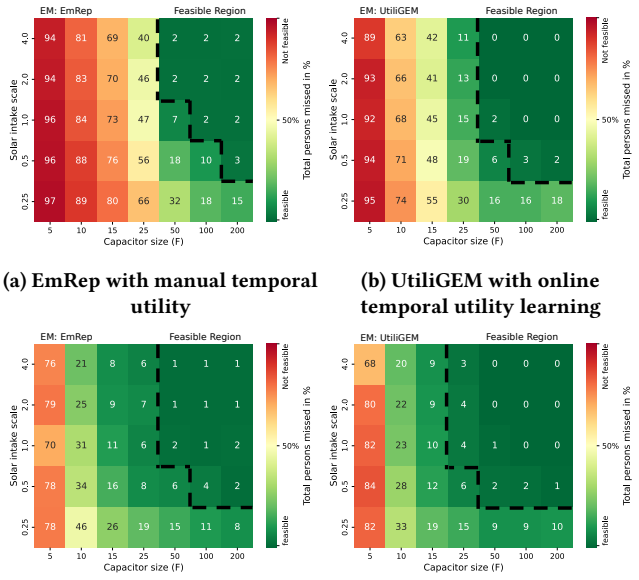
Figure 6: Comparison of energy managers with different utility profiles. (a) Shows the comparison against the learned utility profile, and (b) Shows the comparison against the maximum utility profile.

utility profile where all slots are equally important (EmRep with 1 utility in all time slots, UtiliGEM with similar priority for all time slots), even though they typically account for time-varying utility. EmRep fails to allocate more resources during buffer saturation and high intake phases because it decouples these phases, relying on DS SoC. It tracks the maximum and minimum SoC curve time of DS but is sensitive to it, trying to achieve the targets. As illustrated in Figure 6, between 15 to 20 hours, EmRep's time-sensitive issue is evident as it adjusts allocation to achieve its target SoC. Additionally, we anticipate that reason of failing to allocate more resources during high intake phases is due to not recognizing these phases effectively. Thus, both DS and EmRep fail to fully utilize the available energy surplus, resulting in significant energy wastage.

In contrast, UtiliGEM allocates more resources effectively by performing BS on one slot at a time (using multi-raise and then single-raise), addressing the limitations of both DS and EmRep. This approach allows UtiliGEM to better handle varying utility and allocate resources more effectively during periods of high solar intake and buffer saturation.

Furthermore, in cases where utility does not align with solar intake, allocating more energy during saturation and high solar intake phases helps explore time slots that do not contribute much initially. This helps the UCB learner to become more certain about these low-contribution slots and lower their UCB values, thereby increasing the priority of other slots the next day. In scenarios with smaller buffer sizes that replenish daily, saving energy is less practical since any energy surplus leads to buffer saturation.

[1]https://github.com/areebasad/Simulator-for-Utility-Aware-Energy-Management-in-Energy-Harvesting-and-Battery-Free-Devices



(a) EmRep with manual temporal utility

(b) UtiliGEM with online temporal utility learning

(c) EmRep, modified with manual temporal utility and with non-temporal utility learning

(d) UtiliGEM with online temporal utility learning and with non-temporal utility learning

Figure 7: Comparison of UtiliGEM with state-of-the-art (EmRep) in terms of end-to-end application performance.

6.2 Extensive Configuration Simulations

To evaluate our energy manager against state-of-the-art methods, we focus on end-to-end application performance, i.e., detecting persons. Our primary metric is hence the number of missed persons.

We evaluate energy managers under two main settings: one where only the temporal utility impact is considered (first row) and another where both temporal and non-temporal utility impacts are considered (second row), as shown in Figure 7. The x-axes represent different capacitor sizes, and the y-axes different solar intake scales. Each combination of these represents a configuration, and the value indicates the percentage of persons missed for each configuration. For all cases a) to d) we see that configurations with larger solar panels and larger energy buffers miss fewer persons, which comes of course with additional hardware costs.

The deployment of wireless sensor devices comes with potentially many equally dimensioned devices, which find themselves in different environments with regard to energy harvesting opportunities. Since it is unlikely that each device gets configured optimally, it is important that they operate in a "feasible" manner also in sub-optimal configurations. If we assume that it is acceptable to miss 5% of the persons, we can define feasible operational regions, indicated by the dashed lines in the diagrams.

Figure 7a shows the current state-of-the-art and serves as baseline, where EmRep is fed with manual temporal utility, specifying high utility from 08:00 to 20:00. The feasible region covers 9 configurations. In contrast, UtiliGEM, shown in Figure 7b, learns the temporal utility without any prior information, initially treating all time slots as equally important. Importantly, we do not employ non-temporal utility in a) and b). As a result, the application accuracy levels vs energy relation is as when tiles are selected randomly (without VAM) in Figure 3 and remains the same for the whole duration. UtiliGEM shows an average improvement of 48% with fewer persons missed across all configurations over the baseline in a). In addition, it expands the number of feasible configurations to 11. Note that the results also include the learning phase, which is often overlooked in other studies.

Next, we explore the impact of combining non-temporal learning with temporal learning for UtiliGEM, as illustrated in Figure 7d. This combination results in an overall improvement of 73% compared to the baseline. Most importantly, UtiliGEM results in 15 feasible configurations. If we just look at the required buffer sizes, we observe a reduction in the necessary buffer size from 50 F (EmRep) to 25 F (UtiliGEM), a reduction of 50% in size.

To understand the effect of the non-temporal utility learning in our use case, we also provided EmRep with non-temporal utility learning, with results shown in Figure 7c. As discussed in Section 3.2, non-temporal utility learning reduces the application's energy requirements. Specifically, after the convergence of the visual attention model,[2] considering only the most valuable 20 tiles as evident in Figure 2 ensures sufficient accuracy. In our use case, this enables even a simpler energy manager such as EmRep to perform relatively well, where the feasible region now counts 11 configurations in Figure 7c. However, this is still below the performance of UtiliGEM with 15 configurations, since only UtiliGEM adapts to the reduced energy requirement by first distributing the energy for these 20 tiles across all time slots and then allocating any remaining energy to further improve the model.

7 Conclusion

We have shown how contextual knowledge about an observed phenomenon can be exploited to make the operation of wireless sensing devices powered by energy harvesting more efficient. For that, we explicitly use the concept of temporal and non-temporal utility. Instead of pre-defining the utility of an application manually, we show how the utility can be learned, which reduces manual design effort and takes care of adaptation over time. We then introduced the first energy manager that takes the temporal and non-temporal aspects of utility into account. Our simulations show that UtiliGEM allocates more resources as compared to prior approaches. In our real-world case study of estimating person counting, we observe that UtiliGEM achieves significantly better end-to-end application performance as compared to prior energy managers.

We expect that the concept of utility will only get more important with the increasing capabilities of sensor nodes of performing machine learning and inference on-board, like with TinyML. TinyML models can mark significant events and their timings for temporal learning, and on-device data adaptation [16] or retraining with local data can improve non-temporal utility, achieving the acceptable accuracy with fewer layers. This requires also energy management to take utility better into account, as shown here with UtiliGEM.

[2]Observed when the attention map shape remains consistent, set empirically here at around 20 days based on findings in [3])

References

[1] Saad Ahmed, Abu Bakar, Naveed Anwar Bhatti, Muhammad Hamad Alizai, Junaid Haroon Siddiqui, and Luca Mottola. 2019. The betrayal of constant power× time: Finding the missing joules of transiently-powered computers. In *Proceedings of the 20th ACM SIGPLAN/SIGBED International Conference on Languages, Compilers, and Tools for Embedded Systems*. 97–109.

[2] Saad Ahmed, Naveed Anwar Bhatti, Muhammad Hamad Alizai, Junaid Haroon Siddiqui, and Luca Mottola. 2019. Efficient intermittent computing with differential checkpointing. In *Proceedings of the 20th ACM SIGPLAN/SIGBED International Conference on Languages, Compilers, and Tools for Embedded Systems*. 70–81.

[3] Hafiz Areeb Asad, Frank Alexander Kraemer, Kerstin Bach, Christian Renner, and Tiago Santos Veiga. 2022. Learning attention models for resource-constrained, self-adaptive visual sensing applications. *Proceedings of the Conference on Research in Adaptive and Convergent Systems* (2022), 165–171. https://doi.org/10.1145/3538641.3561505

[4] Domenico Balsamo, Alex S Weddell, Anup Das, Alberto Rodriguez Arreola, Davide Brunelli, Bashir M Al-Hashimi, Geoff V Merrett, and Luca Benini. 2016. Hibernus++: a self-calibrating and adaptive system for transiently-powered embedded devices. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 35, 12 (2016), 1968–1980.

[5] Fulvio Bambusi, Francesco Cerizzi, Yamin Lee, and Luca Mottola. 2022. The case for approximate intermittent computing. In *2022 21st ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*. IEEE, 463–476.

[6] Bernhard Buchli, Felix Sutton, Jan Beutel, and Lothar Thiele. 2014. Dynamic power management for long-term energy neutral operation of solar energy harvesting systems. In *Proceedings of the 12th ACM conference on embedded network sensor systems*. 31–45.

[7] Pietro Farina, Subrata Biswas, Eren Yıldız, Khakim Akhunov, Saad Ahmed, Bashima Islam, and Kasım Sinan Yıldırım. 2024. Memory-efficient Energy-adaptive Inference of Pre-Trained Models on Batteryless Embedded Systems. *arXiv preprint arXiv:2405.10426* (2024).

[8] Kai Geissdoerfer, Raja Jurdak, Brano Kusy, and Marco Zimmerling. 2019. Getting more out of energy-harvesting systems: Energy management under time-varying utility with preact. In *Proceedings of the 18th International Conference on Information Processing in Sensor Networks*. 109–120.

[9] Marco Giordano, Philipp Mayer, and Michele Magno. 2020. A battery-free long-range wireless smart camera for face detection. In *Proceedings of the 8th International Workshop on Energy Harvesting and Energy-Neutral Sensing Systems*. 29–35.

[10] Lars Hanschke and Christian Renner. 2020. Scheduling recurring and dependent tasks in EH-WSNs. *Sustainable Computing: Informatics and Systems* 27 (2020), 100409.

[11] Lars Hanschke and Christian Renner. 2022. EmRep: Energy management relying on state-of-charge extrema prediction. *IET computers & digital techniques* 16, 4 (2022), 91–105.

[12] Bashima Islam and Shahriar Nirjon. 2019. Zygarde: Time-sensitive on-device deep inference and adaptation on intermittently-powered systems. *arXiv preprint arXiv:1905.03854* (2019).

[13] Seunghyeok Jeon, Yonghun Choi, Yeonwoo Cho, and Hojung Cha. 2023. Harvnet: resource-optimized operation of multi-exit deep neural networks on energy harvesting devices. In *Proceedings of the 21st Annual International Conference on Mobile Systems, Applications and Services*. 42–55.

[14] Aman Kansal, Jason Hsu, Sadaf Zahedi, and Mani B Srivastava. 2007. Power management in energy harvesting sensor networks. *ACM Transactions on Embedded Computing Systems (TECS)* 6, 4 (2007), 32–es.

[15] Trong Nhan Le, Olivier Sentieys, Olivier Berder, Alain Pegatoquet, and Cecile Belleudy. 2012. Power manager with PID controller in energy harvesting wireless sensor networks. In *2012 IEEE International Conference on Green Computing and Communications*. IEEE, 668–670.

[16] Seulki Lee and Shahriar Nirjon. 2020. Learning in the wild: When, how, and what to learn for on-device dataset adaptation. In *Proceedings of the 2nd International Workshop on Challenges in Artificial Intelligence and Machine Learning for Internet of Things*. 34–40.

[17] 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.

[18] Alessandro Montanari, Manuja Sharma, Dainius Jenkus, Mohammed Alloulah, Lorena Qendro, and Fahim Kawsar. 2020. ePerceptive: energy reactive embedded intelligence for batteryless sensors. In *Proceedings of the 18th Conference on Embedded Networked Sensor Systems*. 382–394.

[19] Clemens Moser, Lothar Thiele, Davide Brunelli, and Luca Benini. 2009. Adaptive power management for environmentally powered systems. *IEEE Trans. Comput.* 59, 4 (2009), 478–491.

[20] Abdulmajid Murad, Frank Alexander Kraemer, Kerstin Bach, and Gavin Taylor. 2024. Uncertainty-aware autonomous sensing with deep reinforcement learning. *Future Generation Computer Systems* 156 (2024), 242–253.

[21] Christian Renner, Stefan Unterschütz, Volker Turau, and Kay Römer. 2014. Perpetual data collection with energy-harvesting sensor networks. *ACM Transactions on Sensor Networks (TOSN)* 11, 1 (2014), 1–45.

[22] Shaswot Shresthamali, Masaaki Kondo, and Hiroshi Nakamura. 2017. Adaptive power management in solar energy harvesting sensor node using reinforcement learning. *ACM Transactions on Embedded Computing Systems (TECS)* 16, 5s (2017), 1–21.

[23] Richard S. Sutton and Andrew G. Barto. 2018. *Reinforcement Learning: An Introduction* (second ed.). The MIT Press.

[24] Sumanth Umesh and Sparsh Mittal. 2021. A survey of techniques for intermittent computing. *Journal of Systems Architecture* 112 (2021), 101859.

[25] 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.

[26] Fan Yang, Ashok Samraj Thangarajan, Gowri Sankar Ramachandran, Wouter Joosen, and Danny Hughes. 2021. AsTAR: Sustainable energy harvesting for the Internet of Things through adaptive task scheduling. *ACM Transactions on Sensor Networks (TOSN)* 18, 1 (2021), 1–34.