tinyMAN: Lightweight Energy Manager using Reinforcement Learning for Energy Harvesting Wearable IoT Devices

Toygun Basaklar
basaklar@wisc.edu
University of Wisconsin-Madison
Madison, Wisconsin, USA

Yigit Tuncel
tuncel@wisc.edu
University of Wisconsin-Madison
Madison, Wisconsin, USA

Umit Y. Ogras
uogras@wisc.edu
University of Wisconsin-Madison
Madison, Wisconsin, USA

ABSTRACT

Advances in low-power electronics and machine learning techniques lead to many novel wearable IoT devices. These devices have limited battery capacity and computational power. Thus, energy harvesting from ambient sources is a promising solution to power these low-energy wearable devices. They need to manage the harvested energy optimally to achieve energy-neutral operation, which eliminates recharging requirements. Optimal energy management is a challenging task due to the dynamic nature of the harvested energy and the battery energy constraints of the target device. To address this challenge, we present a reinforcement learning based energy management framework, tinyMAN, for resource-constrained wearable IoT devices. The framework maximizes the utilization of the target device under dynamic energy harvesting patterns and battery constraints. Moreover, tinyMAN does not rely on forecasts of the harvested energy which makes it a prediction-free approach. We deployed tinyMAN on a wearable device prototype using TensorFlow Lite for Micro thanks to its small memory footprint of less than 100 KB. Our evaluations show that tinyMAN achieves less than 2.36 ms and 27.75 μJ while maintaining up to 45% higher utility compared to prior approaches.

KEYWORDS

Energy harvesting, reinforcement learning, battery management, IoT, energy efficiency, resource allocation

1 INTRODUCTION

The emergence of small form-factor and low-cost wearable Internet of Things (IoT) devices lead to many novel edge-computing use cases [4, 6, 19]. These include promising applications at the edge ranging from remote health monitoring to smart livestock monitoring systems [11, 13, 16, 20]. The devices that run these applications must operate with a tight energy budget (~μW) and computational power due to limited battery capacity and small form-factor to be practical [2, 8].The small battery capacity limits

the battery lifetime and requires frequent recharging, deteriorating the user experience. To mitigate this effect, energy harvesting (EH) from ambient sources, such as light, motion, electromagnetic waves, and body heat, has emerged as a promising solution to power these devices [12, 15].

Energy-neutral operation (ENO) is achieved if the total energy consumed over a given period equals the energy harvested in the same period. EH solutions should achieve ENO to ensure that the device maintains a certain battery level by continuously recharging the battery. However, relying only on EH is not sufficient to achieve energy neutrality due to the uncertainties of ambient sources. The application performance and utilization of the device can tank in low EH conditions [9]. Energy management algorithms need to use the available energy judiciously to maximize the application performance while minimizing manual recharge interventions to tackle this challenge [17]. These algorithms should satisfy the following conditions to be deployed on a resource-constrained device: *(i)* incurring low execution time and power consumption overhead, *(ii)* having a small memory footprint, *(iii)* being responsive to the changes in the environment, and ideally, *(iv)* learning to adopt such changes. To this end, our goal is to develop a lightweight energy manager that enables ENO while maximizing the utilization of the device under dynamic energy constraints and EH conditions.

This paper presents a reinforcement learning (RL) based energy management framework, tinyMAN, for resource-constrained wearable edge devices. tinyMAN takes the battery level and the previous harvested energy values as inputs (states) and maximizes the utility of the device by judiciously allocating the harvested energy throughout the day (action). It employs Proximal Policy Optimization (PPO) algorithm, which is a state-of-the-art RL algorithm for continuous action spaces [14]. Hence, the energy allocation values that tinyMAN yields can take continuous values according to the current energy availability. Over time, by interacting with the environment, the agent learns to manage the harvested energy on the device according to battery energy level and the harvested energy. To achieve this, we first develop an environment for the RL agent to interact with. This environment makes use of the light and motion EH modalities and *American Time Use Survey* [18] data from 4772 different users to model the dynamic changes in the harvested energy and battery. Then, we design a generalized reward function that defines the device utility as a function of the energy consumption. The nature of the reward function also enables compatibility with any device and application.

tinyMAN is trained on a cluster of users with randomly selected initial battery energy levels and EH conditions. Therefore, it is responsive to various EH and battery energy level scenarios. We compare our approach to prior approaches in the literature and also

with an optimal solution. This comparison shows that tinyMAN achieves up to 45% higher utility values. Furthermore, we deploy our framework on a wearable device prototype to measure the execution time, energy consumption, and memory usage overhead. *The major contributions of this work are as follows:*

- We present tinyMAN, a *prediction-free* RL based energy manager for resource-constrained wearable edge IoT devices,
- tinyMAN achieves 45% higher device utilization than the state-of-the-art approaches by learning the underlying EH patterns for different users while maintaining energy neutrality,
- tinyMAN is easily deployable on wearable devices thanks to its small memory footprint of less than 100 KB and energy consumption of 27.75 μJ per inference.

In the rest, Section 2 reviews the related work, while Section 3 introduces the problem formulation and describes the PPO algorithm. Section 4 formulates the environment dynamics and presents the proposed energy manager, tinyMAN. Finally, we evaluate and discuss the results in Section 5 and conclude the paper in Section 6.

2 RELATED WORK

Energy harvesting devices aim for ENO to achieve self-sustainability. Kansal et al. [12], ensure ENO if the total energy consumed in a given period is equal to the harvested energy in the same period. The authors propose a linear programming approach to maximize the duty cycle of a sensor node and a lightweight heuristic to help solve the linear programming with ease. Although their approach is lightweight, it does not consider the application requirements when deciding the duty cycle of the nodes. Bhat et al. address this issue by using a generalized utility function that defines the application characteristics [3]. They presented a lightweight framework based on the closed-form solution of the optimization problem that maximizes the utility while maintaining ENO. However, the framework can yield sub-optimal solutions since the closed-form solution is obtained by relaxing one of the constraints in the original problem. In addition, both approaches depend on a predictive model for the future EH values. *Thus, their performances are highly dependent upon the accuracy of the predictions.*

Prediction-free approaches do not rely on forecasts of the harvested energy, in contrast to the prediction-based approaches presented above [1]. RLMan is a recent prediction-free energy management approach based on reinforcement learning [1]. It aims to maximize packet generation rate while avoiding power failures. Although it shows significant improvements in average packet rate, the reward function in RLMan focuses on maximizing the packet rate in a point-to-point communication system, which does not generalize to other performance metrics and ignores application requirements. In addition, the authors do not discuss the deployability of their framework on edge devices. In complement to the previous studies, we present tinyMAN, a prediction-free energy
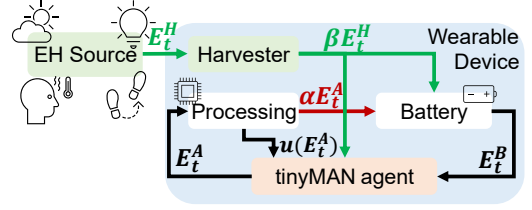
Table 1: Related work in energy management

Ref	Generalized Reward	Prediction Free	Deployable
[12]	✗	✗	✓
[3]	✓	✗	✓
[1]	✗	✓	✗
tinyMAN	✓	✓	✓

Figure 1: Illustration of the environment.

manager which uses a generalized reward function and is easily deployable on resource-constrained edge devices, as shown in Table 1. *Furthermore, we provide open-source access to the trained models and to our codebase.*

3 BACKGROUND

This section first introduces the battery energy dynamics and constraints to formulate the optimization problem. It also explains how various EH patterns are obtained. Then, it describes the Proximal Policy Optimization algorithm used to train the tinyMAN RL agent.

3.1 Problem Formulation

The proposed tinyMAN framework is deployed in an environment that consists of a target device and an EH source, as depicted in Figure 1. In the following, we define the battery energy dynamics, the relevant constraints, and the utility function of the device and explain the EH source model.

Battery dynamics and constraints: tinyMAN finds the optimum energy allocations that maximize the utilization of a target device under ENO and battery constraints. In this work, we use a prototype wearable device as the target platform to deploy tinyMAN. The device houses a flexible, small form-factor LiPo battery with a capacity of 12 mAh, and can charge the battery through energy harvesting. Therefore, the battery energy dynamics in the environment is a function of:

(1) E_t^B the battery energy level at the start of time interval t
(2) E_t^A the allocated energy at the start of time interval t
(3) E_t^H the harvested energy in time interval t

Our energy management framework uses an episodic setting where each episode corresponds to a single day ($T = 24$ hours), and each step t in an episode corresponds to an hour.

Using these definitions, we write the battery energy dynamics as follows:

$$E_{t+1}^B = E_t^B + \beta E_t^H - \alpha E_t^A, \ t \in T \tag{1}$$

where β corresponds to the efficiency of the harvester and α corresponds to the percent utilization of the allocated energy (i.e., αE_t^A is the actual consumed energy).

There are two physical constraints on the battery level. It is bounded from below at zero and from the top at the battery capacity (E_{cap}^B). Furthermore, we want the device to have an emergency reservoir at all times to serve as backup energy:

$$E_{cap}^B \geq E_t^B \geq E_{min}^B, \quad t \in T \tag{2}$$

To achieve ENO, tinyMAN ensures that the battery energy level at the end of an episode is equal to a specified target:

$$E_T^B \approx E_{target} \tag{3}$$

Table 2: Components used in the prototype wearable device.

Component	VDD	I_{idle}	I_{active}	Part #
Microcontroller	1.8-3.8V	0.9 μA	Sensor Cont.: 30 μA Active: 3.4 mA	CC2652R
IMU	1.7-3.6V	8 μA	Acc only: 450 μA Gyro only: 3.2 mA	MPU9250
Nonvolatile Ram	1.6-3.6V	10 μA	Rewrite: 1.3 mA Read-out: 0.2 mA	MB85AS4MT
Humid. & Temp. Sensor	2.7-5.5V	0.1 μA	1 Hz: 1.2 μA	HDC1000
Ambient Light Sensor	1.6-3.6V	0.3 μA	1.8 μA	OPT3001
Boost Converter for EH	2.5-5.2V	0.3 μA	-	BQ25504
LDO linear regulator	2.0-5.5V	35 μA	-	TLV702

For achieving ENO, we set $E_{target} = E_0^B$ such that the battery energy level at the end of the episode is equal to the battery energy level at the beginning of the same episode. We enforce these constraints using the reward function as explained in Section 4.1.

Device utility: The utilization of the device is a metric that represents the useful output produced by the device, such as accuracy or throughput, depending on the target application running on the device. For example, for human activity recognition, a state-of-the-art application that utilizes a low-power wearable device, the utility is defined by the classification accuracy. Nonetheless, tinyMAN supports any arbitrary utility function.

For the current work, we define the utility according to the minimum energy consumption of the device in an hour. Specifically, the device utility is zero (or negative) if the allocated energy is less than the minimum energy consumption of the device in a given interval. We list the components used in the wearable device prototype in Table 2 to calculate the minimum energy consumption in an hour. According to these values, the sum of the idle currents of the components amounts to 54.6 μA, and the idle energy consumption of the device in an hour is $E_{min}^A = 0.64$ J with a VDD of 3.3V. Therefore, the device utility will vanish if $E_t^A < E_{min}^A$ (i.e., the device does not produce any useful output). For $E_t^A > E_{min}^A$, *the utility function can have any shape according to the needs of the application.* For this work, we used a logarithmic utility function with a diminishing rate of return, as elaborated in Section 4.1.

EH Source: The EH source uses the dataset presented in [16] to generate EH scenarios according to different user patterns. This dataset uses the combination of light and motion energy as the ambient energy sources, and it combines power measurement data with the activity and location information of 4772 users from the American Time Use Survey dataset [18] to generate varying 24-hour EH patterns per user. We divide the EH dataset [16] into four clusters according to the users' EH patterns throughout the day. The hourly distributions of these four clusters are illustrated in Figure 2. These distributions are based on the mean and the standard deviation of EH patterns in the same cluster. Therefore, the EH source generates a harvested energy value at every hour according to the distributions in the dataset as the day progresses.

3.2 Proximal Policy Optimization

The main objective of an RL agent is to maximize the cumulative rewards by interacting with the environment. According to the state s of the environment and the current policy π, the agent chooses an action a. Based on this action, environment returns next state s' and reward r. Th environment is initialized with state s_0 (start of the day, $t = 0$) and terminates after $T = 24$ steps (end of the day, $t = 24$). The policy π is represented by a neural network with parameters θ. The agent interacts with the environment using the current policy π_θ and collects samples (s, a, r, s'). In policy gradient algorithms, the policy network is updated using the gradient of the policy multiplied with discounted cumulative rewards as a loss function and plugging it into the gradient ascent algorithm. This update is generally done using samples from multiple episodes. The discounted cumulative rewards can exhibit high variations since each episode follows a different trajectory based on the actions. To reduce this variance, a bias is introduced as an advantage function that measures the benefit of taking action at a given state. The loss function then takes the form:

$$L_\theta = \sum_{n=0}^{N} \sum_{t=0}^{T} log\pi_\theta(a_t|s_t)A(s_t, a_t) \tag{4}$$

$$A(s_t, a_t) = r_t + \gamma V_\phi(s_{t+1}) - V_\phi(s_t) \tag{5}$$

Here, $\pi_\theta(a_t|s_t)$ is the current policy which gives the probability of taking action a in state s. Advantage function is represented by $A(s_t, a_t)$ and is given by Equation 5 where $\gamma \in [0, 1]$ is the discount factor and $V_\phi(s)$ is the value network which estimates the expected discounted sum of rewards for a given state s. N is the number of episodes, and T is the number of steps in an episode. Value network $V_\phi(s)$ is also updated during training using gradient descent with the mean-squared error between target values and the estimated values as a loss function:

$$L_\phi = \frac{1}{NT} \sum_{n=0}^{N} \sum_{t=0}^{T} (V_\phi(s_t) - (r_t + \gamma V_\phi(s_{t+1})))^2 \tag{6}$$

PPO aims at improving the training stability by avoiding network parameter updates that change the policy drastically at each step of optimization. To this end, it modifies the policy loss (Equation 4) in such a way that the distance between new policy ($\pi_\theta(a|s)$) and the old policy ($\pi_{\theta old}(a|s)$) is enforced to be small. It achieves its goal using the following loss function:

$$L_\theta^{PPO} = \frac{1}{NT} \sum_{n=0}^{N} \sum_{t=0}^{T} min(\rho(\theta)A_t, \ clip(\rho(\theta), 1-\epsilon, 1+\epsilon)A_t) \tag{7}$$

$$\rho(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta old}(a_t|s_t)} \tag{8}$$

In this equation, $\pi_{\theta old}(a|s)$ is the policy that is used to collect samples by interacting with the environment and $\pi_\theta(a|s)$ is the policy that is being updated using this loss function. PPO imposes a limitation on the distance between $\pi_{\theta old}(a|s)$ and $\pi_\theta(a|s)$ by clipping the ratio $\rho(\theta)$ between two distribution with ϵ where ϵ is a hyperparameter of the algorithm. An entropy term may also be included in this loss function to encourage sufficient exploration.

4 PROPOSED ENERGY MANAGER – tinyMAN

This section provides the environment dynamics and introduces the RL framework, the core algorithm used in tinyMAN.
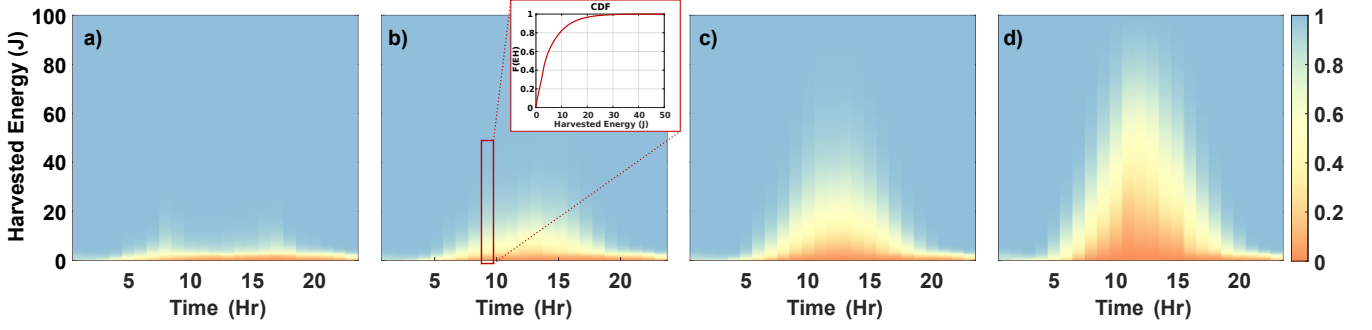
Figure 2: Cumulative distribution function of the harvested energy for a) Cluster 1, b) Cluster 2, c) Cluster 3, and d) Cluster 4

4.1 Environment Dynamics

Our goal is to maximize the utilization of the device under certain battery energy level constraints. In our framework, the environment dynamics are determined to enable the adaptability of tinyMAN by any device and application.

State Space: The state is a 5-tuple that consists of:

- **Current battery energy level** ($E_t^B \in [0, E_{cap}^B]$)**:** The energy level of the battery at the beginning of the current step t.
- **EH from the previous time step** ($E_{t-1}^H \in \mathbb{R}$)**:** Harvested energy during the previous step $t-1$.
- **Initial battery energy level** ($E_0^B \in \mathbb{R}$)**:** The energy level of the battery at the beginning of the episode ($t=0$).
- **Time** ($t \in \mathbb{Z}$)**:** The current step t, which corresponds to the current hour of the day.
- **Cumulative EH** ($\sum_{\tau=0}^{t-1} E_\tau^H \in \mathbb{R}$)**:** Cumulative harvested energy in the previous time steps.

Action Space: The action is the allocated energy at every time step ($E_t^A \in [E_{min}^A, E_t^B]$). Since the application on the device needs a minimum energy level to stay in the idle state, we set a minimum level constraint on the action (E_{min}^A).

Reward function: Our objective is to maximize the utility of the device under certain constraints on the battery energy level. tiny-MAN supports any arbitrary utility function, but to have a fair comparison with the literature [3], we use the following logarithmic utility function in this work:

$$u(E_t^A) = \ln \left(\frac{E_t^A}{E_{min}^A} \right) \tag{9}$$

In an RL setting, the constraints on the battery can be imposed by the reward function. There are two constraints that can be imposed to the reward function: *(i)* emergency reservoir energy constraint (Equation 2) and *(ii)* ENO constraint (Equation 3). Considering the objective and the constraints on the battery, the reward function becomes:

$$r_t = \begin{cases} u(E_t^A) & E_t^B \geq E_{min}^B \text{ and } t \neq T \\ u(E_t^A) - (E_{min}^B - E_t^B)^2 & E_t^B \leq E_{min}^B \text{ and } t \neq T \\ -(E_t^B - E_{target})^2 & t = T \end{cases} \tag{10}$$

Here, we impose the emergency reservoir energy constraint using the term $-(E_{min}^B - E_t^B)^2$ and the ENO constraint using the term

$-(E_t^B - E_{target})^2$. Moreover, an episode terminates if time T is reached or the battery is completely drained.

According to the environment dynamics explained in this section, we develop our environment in Python and register it as an OpenAI's Gym [5] environment.

4.2 Proposed RL Framework

Since the proposed tinyMAN framework is deployed on a wearable device, we first identify the characteristics of the target device such as battery capacity, minimum battery energy level (E_{min}^B), and minimum energy allocation (E_{min}^A). These characteristics do not change over time during the training. The EH dataset [16] is divided into four clusters according to the users' EH patterns throughout the day. The agent is trained separately on each cluster. Specifically, at the beginning of each episode n, we randomly choose an initial battery energy level. Then, we generate an EH pattern from the hourly distributions illustrated in Figure 2. The generated EH pattern is different for each episode. Thus, tinyMAN inherently learns the

---

Algorithm 1: tinyMAN - RL based Energy Manager

Initialize policy and value network with parameters θ_0 and ϕ_0
Initialize random policy π_{θ_0}, empty trajectory buffer $\mathcal{D}$ with size $\mathbb{D}$
for $n = 0$: N **do**
    Initialize environment with randomly chosen initial
    battery energy E_0^B and EH patterns
    while $\mathcal{D}$ *is not full* **do**
        for $t = 0$: T **do**
            Choose a_t according the current policy π_{θ_i}
            Collect samples $\{s_t, a_t, r_t, s_t'\}$ by interacting with
            the environment using action a_t
    Obtain A_t, $r_t + V_{\phi_i}(s_{t+1})$ and $\pi_{\theta_i}(a_t|s_t)$
    using policy and value networks (see Section 3.2 for details)
    for $k = 1$: K **do**
        for $b = 0$: $(\mathbb{D}/d)$ **do**
            $batch_{start} = d \times (b - 1)$
            $batch_{end} = d \times (b)$
            $minibatch \leftarrow \mathcal{D}[batch_{start} : batch_{end}]$
            $L \leftarrow -L_\theta^{PPO} + c_1 L_\phi + c_2 H(\pi_\theta)$
            Minimize the total loss L
    $\theta_{i+1} \leftarrow L_{\theta_K}^{PPO}$
    $\phi_{i+1} \leftarrow L_{\phi_K}$
    Clear $\mathcal{D}$

EH patterns of the users in that cluster. The initial conditions and the EH patterns can differ significantly between different episodes. This may result in a high gradient variance and unstable learning progress during the training. For this reason we employ PPO in our work, as it guarantees that policy updates do not deviate largely. In addition, PPO uses little space in the memory, which fits the resource-constrained nature of the target device.

Algorithm 1 describes the training of tinyMAN agent for a given cluster of users. The agent starts the first episode with a random policy π_{θ_0} with parameters θ_0. Using the current policy π_{θ_i}, the agent first collects samples until the trajectory buffer $\mathcal{D}$ with a predefined size of $\mathbb{D}$ is full. Note that this trajectory buffer is not the experience replay buffer commonly used in off-policy RL algorithms. Using the samples in the trajectory buffer, advantages A_t, target values $r_t + V_{\phi_i}(s_{t+1})$, and the probabilities $\pi_{\theta_i}(a_t|s_t)$ are obtained using the policy network π_{θ_i} and the value network V_{ϕ_i}. The algorithm updates both the policy and the value network parameters (θ, ϕ) according to the loss functions described in Section 3.2. We augment the loss function for different networks and add an entropy term $H(\pi_\theta)$ to increase the exploration of the algorithm. PPO updates the network parameters by generally taking multiple steps on minibatches. The number of optimization steps K and the minibatch size d, and the clipping value ϵ in the policy loss function are hyperparameters of the network. Both networks consist of fully connected layers with hyperbolic tangent as activation function. Additionally, the policy network also has a Gaussian distribution head to yield continuous values from a distribution. The number of hidden layers (N_{Layer}) and neurons (N_{Neuron}) are the same for both networks.

We implement tinyMAN in Python by utilizing PFRL [10] library for the PPO algorithm using Adam optimizer with a learning rate of 1E-4. The hyperparameters for tinyMAN are given in Table 3.

5 EXPERIMENTAL EVALUATIONS

This section evaluates the tinyMAN framework from three aspects: (i) it presents the evolution of the tinyMAN agent during training, (ii) it compares the performance of the tinyMAN framework to two prediction-based prior approaches [3, 12] in the literature, and (iii) it provides execution time, energy overhead and memory footprint measurements of the tinyMAN framework when deployed on a wearable device prototype.

Table 3: Definition of the hyperparameters and their values.

Hyperparameter	Description	Value
α	Percent utilization	1
β	Efficiency of the harvester	1
γ	Discount factor	1
N	Number of episodes	200000
T	Number of time steps	24
c_1	Value loss coefficient	0.5
c_2	Entropy coefficient	0.01
ϵ	Clipping factor	0.3
K	PPO optimization steps	10
d	Minibatch size	64
$\mathbb{D}$	Trajectory buffer size	2048
N_{Layer}	Number of hidden layers	1
N_{Neuron}	Number of hidden neurons	{16,32,64}

5.1 Training Evolution

We first evaluate our agent's performance during training to highlight the evolution of a generalizable agent. The harvested energy levels of the users are the lowest in cluster 1, and the highest in cluster 4, as depicted in Figure 2. This section illustrates the results for cluster 2 since the users in this cluster are representative of an average person with low to intermediate levels of harvested energy during the day. Other clusters produce similar results. Furthermore, we set the emergency reservoir energy as $E_{min}^B = 10$ J, which roughly corresponds to 5 minutes of active time for the components listed in Table 2. We stress that this parameter can be tailored according to the requirements of another device or application.

Figure 3 shows the allocated energy, battery energy level, and the expected/actual EH patterns for the median user in cluster 2 during training. We follow the training steps described in Section 4.2. The initial battery energy level E_0^B is set as 16 J, which corresponds to 10% of the battery. At the early stages of the training, tinyMAN takes conservative actions as shown in Figure 3 (1a). This suggests that the target energy level constraint (i.e., $E_T^B > E_{target}$) penalty is dominating the agent in these early stages. As the training progresses, actions that the agent takes are in correlation with the harvested energy since tinyMAN starts to learn a generalized representation of the EH patterns in this cluster. Specifically, energy allocations increase as the EH increases and decrease as the EH decreases. This behavior and the fact that the constraints are satisfied can be seen in Figure 3 (b) and (c).

In addition to the behavior of the tinyMAN agent, we also illustrate the energy allocations computed by two prior prediction-based approaches in the literature [3, 12]. As both of these approaches are prediction-based, they use the specific expected EH pattern for a user, depicted with the red line in Figure 3 (3a, 3b, 3c). On the contrary, tinyMAN implicitly learns the actual EH patterns during training, making it a prediction-free approach. Finally, we compare our results against the optimal solution obtained by an offline solver (e.g., CVX) using the actual harvested energy during the day. Although this solution is unfair and unrealistic, it provides an anchor point for assessing the quality of the energy allocations. It can be seen that tinyMAN's actions oscillate around the optimal values with the red line in Figure 3 (1b, 1c).

5.2 Performance Evaluation

We evaluate the performance of tinyMAN with three model sizes: $N_{Neuron} = \{16, 32, 64\}$. Similar to Section 5.1, we compare the performance of tinyMAN to two prior prediction-based approaches in the literature [3, 12], and also to an optimal solution. For a fair evaluation, we exclude randomly selected 10% of the users in a cluster during training. Then, using the energy harvesting patterns of these users, we compute the total utility obtained at the end of the day as follows:

$$U = \sum_{t=0}^{T} u(E_t^A) \tag{11}$$

For each cluster and tinyMAN model size, we evaluate the performance of our approach at four different initial battery energy levels: $E_0^B = \{16, 48, 112, 144\}$ J. Table 4 presents the average total utility obtained from these four conditions for all approaches. For a model size of 64, tinyMAN achieves up to 45% and 10% higher utility values
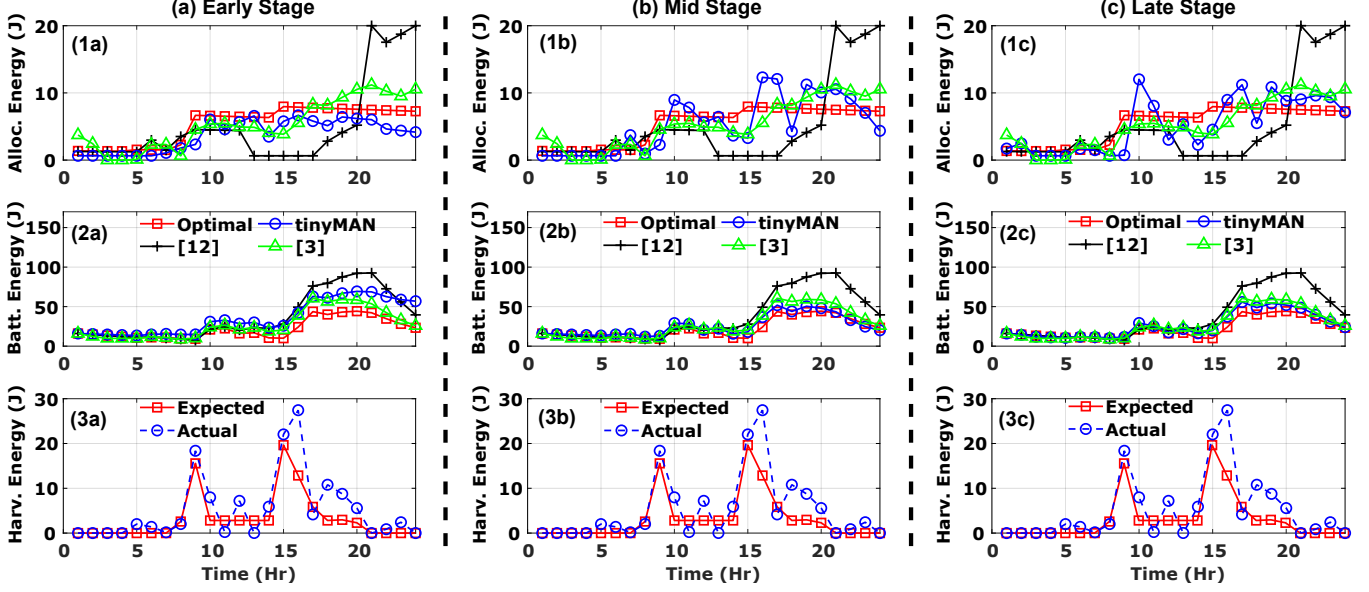
Figure 3: Policy followed by tinyMAN agent at different stages of training for the median user of cluster 2. (N_{Neuron} = 64).

than [12] and [3] while staying within at least 83% of the optimal utility. Similarly, tinyMAN achieves up to 44% higher utility values compared to prior approaches. The utility achieved by tinyMAN decreases with smaller model sizes. This behavior is expected as the information captured by the network degrades. Moreover, we observe that for all solutions, in general, as the harvested energy increases from cluster 1 to cluster 4, the total utility increases since the available energy to allocate on the device increases. We emphasize that tinyMAN is trained for various battery energy levels and EH patterns which are generated using only the cluster's EH distribution. This and the performance evaluation support that it can easily adapt to unseen user-specific EH patterns and battery energy levels, making it a preferred energy manager on an edge device with uncertainties in harvested energy.

5.3 Deployability

The TI CC2652R microcontroller used on our prototype device incorporates an ARM Cortex M4F running at 48 MHz and has 352KB of flash memory and 80KB of SRAM. These scarce resources highlight the importance of evaluating the trained models regarding their deployability on the target platform. Therefore, we evaluate the deployability of the trained models from three aspects: (i)

Table 4: Comparison of the average daily utility obtained by tinyMAN with different model sizes to other approaches.

	Optimal	[3]	[12]	tinyMAN		
				16	32	64
Cluster 1	29.5	25.5	18.7	25.1	26.4	25.5
Cluster 2	42.0	35.3	26.3	37.9	37.9	38.1
Cluster 3	52.4	43.1	34.5	35.9	41.8	44.7
Cluster 4	61.5	46.5	41.9	46.4	50.1	51.2
Cluster Avg.	46.4	37.6	30.3	36.4	38.8	39.9

The execution time per inference, (ii) the energy consumption per inference, and (iii) memory utilization of the target hardware platform. To do this analysis, we follow the Tensorflow Lite Micro (TFLM) flow to convert and deploy the trained models on the target device [7]. Then, we measure the current consumption of the TI microcontroller, as shown in Figure 4. Using these measurements, we calculate the execution time (t_{exe}) and energy consumption (E_{exe}) for different network sizes. Finally, we use the "Memory Allocation" report of TI Code Composer Studio to obtain the memory utilization of the device. Table 5 summarizes our results. The reported memory footprint is for the entire application, including necessary drivers and I/Os for debugging, such as UART and timers. We also provide the utility values averaged over all clusters normalized with the optimal utility. The device's execution time, energy consumption, and memory utilization decrease as the model size decreases. Specifically, for a model size of 64, tinyMAN has a memory footprint of 91 KB and it consumes 27.75 µJ per inference. When model sizes of 32 and 16 are used, tinyMAN's memory footprint reduces to 78 KB and 74 KB, respectively. In addition, the energy consumption also reduces to 11.66 µJ and 6.74 µJ. However, these reductions come at the expense of lower normalized utility. Specifically, as model size decreases from 64 to 16, there is a 7% reduction in the normalized utility. In any case, these results suggest that tinyMAN is easily deployable on a resource-constrained wearable IoT device.

Table 5: tinyMAN's overhead for different model sizes.

	Exec. Time	Energy	Memory (Flash+SRAM)	Norm. Utility*
N_{Neuron} = 16	0.75 ms	6.75 µJ	69KB+5KB	0.79
N_{Neuron} = 32	1.12 ms	11.66 µJ	73KB+5KB	0.84
N_{Neuron} = 64	2.36 ms	27.75 µJ	86KB+5KB	0.86

*The utility is normalized with respect to the optimal utility.

6 CONCLUSION

EH from ambient sources is an emerging solution to power low-energy wearable devices. The harvested energy should be managed optimally to achieve energy-neutral operation and eliminate recharging requirements. To this end, this paper presented tiny-MAN, an RL-based prediction-free energy manager for resource-constrained wearable IoT devices. tinyMAN judiciously uses the available energy to maximize the application performance while minimizing manual recharge interventions. It maximizes the device utilization under dynamic energy harvesting patterns and battery constraints. Additionally, tinyMAN is easily deployable on wearable IoT devices thanks to its small memory footprint being less than 100 KB. tinyMAN achieves up to 45% higher device utilization than the prior approaches in the literature by inherently learning the EH patterns of users while consuming less than 27.75 µJ energy per inference. As future work, we plan to extend our prototype device to log the harvested energy over a day. This will pave the way for adding online learning functionality to tinyMAN.

ACKNOWLEDGMENTS

Figure 4: Execution time and energy measurements of tiny-MAN with different model sizes.

REFERENCES

[1] Fayçal Ait Aoudia, Matthieu Gautier, and Olivier Berder. 2018. RLMan: An energy manager based on reinforcement learning for energy harvesting wireless sensor networks. *IEEE Transactions on Green Communications and Networking* 2, 2 (2018), 408–417.

[2] Toygun Basaklar, Yigit Tuncel, Shruti Yadav Narayana, Suat Gumussoy, and Umit Y Ogras. 2021. Hypervector Design for Efficient Hyperdimensional Computing on Edge Devices. *arXiv preprint arXiv:2103.06709* (2021).

[3] Ganapati Bhat, Jaehyun Park, and Umit Y Ogras. 2017. Near-optimal energy allocation for self-powered wearable systems. In *IEEE/ACM International Conference on Computer-Aided Design.* 368–375.

[4] Valentina Bianchi, Marco Bassoli, Gianfranco Lombardo, Paolo Fornacciari, Monica Mordonini, and Ilaria De Munari. 2019. IoT wearable sensor and deep learning: An integrated approach for personalized human activity recognition in a smart home environment. *IEEE Internet of Things Journal* 6, 5 (2019), 8553–8562.

[5] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. 2016. OpenAI Gym. arXiv:arXiv:1606.01540

[6] Maurizio Capra, Riccardo Peloso, Guido Masera, Massimo Ruo Roch, and Maurizio Martina. 2019. Edge computing: A survey on the hardware requirements in the internet of things world. *Future Internet* 11, 4 (2019), 100.

[7] Robert David, Jared Duke, Advait Jain, Vijay Janapa Reddi, Nat Jeffries, Jian Li, Nick Kreeger, Ian Nappier, Meghna Natraj, Shlomi Regev, et al. 2020. Tensorflow lite micro: Embedded machine learning on tinyml systems. *arXiv preprint arXiv:2010.08678* (2020).

[8] Ana Lígia Silva de Lima et al. 2017. Feasibility of Large-Scale Deployment of Multiple Wearable Sensors in Parkinson's Disease. *PLOS One* 12, 12 (2017), e0189161.

[9] Francesco Fraternali, Bharathan Balaji, Dhiman Sengupta, Dezhi Hong, and Rajesh K Gupta. 2020. Ember: energy management of batteryless event detection sensors with deep reinforcement learning. In *Proceedings of the 18th Conference on Embedded Networked Sensor Systems.* 503–516.

[10] Yasuhiro Fujita, Prabhat Nagarajan, Toshiki Kataoka, and Takahiro Ishikawa. 2021. ChainerRL: A Deep Reinforcement Learning Library. *Journal of Machine Learning Research* 22, 77 (2021), 1–14. http://jmlr.org/papers/v22/20-376.html

[11] Shivayogi Hiremath, Geng Yang, and Kunal Mankodiya. 2014. Wearable Internet of Things: Concept, architectural components and promises for person-centered healthcare. In *2014 4th International Conference on Wireless Mobile Communication and Healthcare-Transforming Healthcare Through Innovations in Mobile and Wireless Technologies (MOBIHEALTH).* IEEE, 304–307.

[12] 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.

[13] Billy Pik Lik Lau, Sumudu Hasala Marakkalage, Yuren Zhou, Naveed Ul Hassan, Chau Yuen, Meng Zhang, and U-Xuan Tan. 2019. A survey of data fusion in

smart city applications. *Information Fusion* 52 (2019), 357–374.

[14] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347* (2017).

[15] Yigit Tuncel, Shiva Bandyopadhyay, Shambhavi V Kulshrestha, Audrey Mendez, and Umit Y Ogras. 2020. Towards wearable piezoelectric energy harvesting: Modeling and experimental validation. In *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design.* 55–60.

[16] Yigit Tuncel, Toygun Basaklar, and Umit Ogras. 2021. How much energy can we harvest daily for wearable applications?. In *2021 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED).* IEEE, 1–6.

[17] Yigit Tuncel, Ganapati Bhat, Jaehyun Park, and Umit Ogras. 2021. ECO: Enabling Energy-Neutral IoT Devices through Runtime Allocation of Harvested Energy. *IEEE Internet of Things Journal* (2021).

[18] US Department of Labor. 2018. American Time Use Survey. https://www.bls.gov/tus/, accessed 1 March 2021.

[19] Xiaofei Wang, Yiwen Han, Victor CM Leung, Dusit Niyato, Xueqiang Yan, and Xu Chen. 2020. Convergence of edge computing and deep learning: A comprehensive survey. *IEEE Communications Surveys & Tutorials* 22, 2 (2020), 869–904.

[20] Nuzhat Yamin and Ganapati Bhat. 2021. Online solar energy prediction for energy-harvesting internet of things devices. In *2021 IEEE/ACM International Symposium on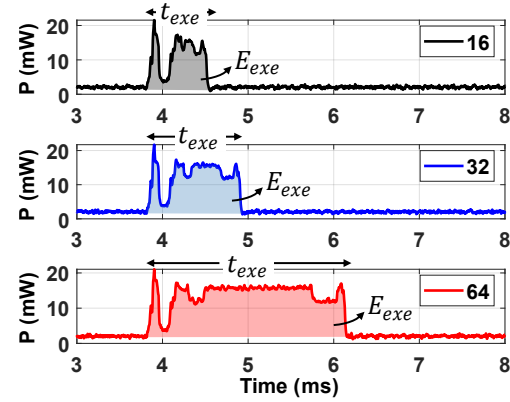 Low Power Electronics and Design (ISLPED).* IEEE, 1–6.