Deep Reinforcement Learning for Energy-efficient Selection of Embedded Services at the Edge

Hugo Hadjur
aivancity School for Technology, Business & Society, France
Inria, Univ Lyon, EnsL, UCBL, CNRS, LIP
hadjur@aivancity.ai

Doreid Ammar
aivancity School for Technology, Business & Society, France
Inria, Univ Lyon, EnsL, UCBL, CNRS, LIP
ammar@aivancity.ai

Laurent Lefèvre
Inria, Univ Lyon, EnsL, UCBL, CNRS, LIP
Lyon, France
laurent.lefevre@inria.fr

***Abstract*—Edge computing helps to release the tension at the center of IoT systems' networks, thus reducing the latency, optimizing the bandwidth, and providing new privacy and security solutions, among others. Despite its benefits, edge computing faces unique challenges, including latency, security, and resource constraints. Among these challenges, energy consumption has emerged in the research community, and the global objective is to do "more with less". Researchers explore diverse strategies to enhance sustainability, from hardware optimizations to intelligent algorithms. The quest for energy efficiency and to reduce several impacts aligns with broader efforts to create an environmentally conscious technology landscape.**

In this paper, we present a task selection model for the edge. We focus on energy consumption and aim to maximize the value given by tasks, all the while minimizing the energy consumed. To do so, we develop a computer environment to simulate outdoor energy-harvesting edge devices, contribute to research reproducibility by recreating a photovoltaic energy harvesting prediction model, and train deep reinforcement learning models to select the best set of tasks at the edge. Our best deep reinforcement learning model, which uses Trust Region Policy Optimization, outperforms our best heuristic and is a robust task selector under varying external conditions.

***Index Terms*—Deep Reinforcement Learning, Internet of Things, Task Selection, Edge Computing, Energy-Harvesting Systems**

I. Introduction

Cloud computing has been the reigning paradigm of the Internet of Things (IoT) thanks to massive centralized infrastructures that are often orchestrated by technology giants and that process the flow of data at the center of networks. Cloud data centers are subject to constraints that increase with the number of devices in their infrastructure. Saturation is the main source of latency, and some fields of application cannot function with a delay, such as autonomous vehicles [1]. When IoT devices are remote, such as in smart agriculture, the cloud becomes a less significant actor in the network, and each deployed device has a more important role to play [2].

Edge computing is a distributed computing paradigm that brings computational resources closer to the edge of the network rather than relying solely on centralized cloud servers. Edge devices, including sensors, gateways, and other IoT devices, perform data processing and analysis locally on or near the device itself. Edge computing does not solve every problem raised by saturated clouds — energy becomes one of the main challenges for isolated devices. These devices must harvest or store energy and work autonomously. They are affected by external parameters, such as the amount of sunshine and the weather. Due to their autonomous nature, they can either store the energy or directly use the harvested energy. In the first case, which will be the focus of this work, their size limits the total amount of energy stored. This leads to the significant challenge of energy efficiency. To gain energy efficiency, one can make use of low-power electronics, use network protocols that are optimized for IoT applications, implement energy harvesting optimizations that utilize different types of energy sources, manage the energy storage devices, and use task scheduling methods to ensure efficient utilization of the harvested and stored energy.

To tackle the energy efficiency challenge in edge devices, reinforcement learning can be a solution. Reinforcement learning (RL) is a category of machine learning methods developed in the 1990s and popularized by [3]. In RL, an agent (the model) uses trial and error to explore and gain knowledge about the dynamics of an environment and maximize its reward. Such a setup is particularly well-suited for the challenge of optimizing the use of services in environments that have varying energy budgets, like edge devices.

The work presented in this paper is part of the research on the Internet of Things (IoT) smart systems and tackles the energy efficiency challenge. This paper aims to provide a solution for the challenge of energy consumption optimization of task selection. It proposes a computer model, trained with specific parameters, but generalizable to all IoT systems that lie outdoors and harvest their energy. This model embeds a photovoltaic energy intake prediction node, based on existing research.

In this paper, we address energy-efficient service placement challenges within a subset of wireless sensor networks, the *autonomous energy-harvesting stationary IoT systems* (AEHSS). We introduce AEHSS as follows:

- Location: The edge device lies immobile, often next to other devices, in an outdoor environment subject to temperature and weather variations. It is, however, often protected by a sealed box.

- Hardware: The edge device relies on a low-cost microcomputer able to collect, process, and transfer data.
- Energy: One system includes one microcomputer, several sensors and one energy node. Due to the outdoor nature of AEHSS, the energy node relies on a solar panel and a battery and uses the *harvest-and-store* mechanism for energy harvesting [4]. The energy intake follows a daily cycle where most of the energy harvest is performed during the daytime.
- Network: On the communication side, systems must be able to carry close-range device-to-device communication and device-to-cloud communication.

The general purpose of AEHSS is to provide valuable information about the subject of interest where the system is deployed (a field, a tree, a beehive, a building, etc.). One system embeds many services that can collect, process and transfer the data.

The paper is organized as follows: Section II focuses on the state of the art of the different tackled subjects. Section III introduces our reinforcement learning environment. Section IV presents the photovoltaic energy prediction model. Section V describes the RL training methods. Section VI shows the results and performances of the different approaches to task selection in AEHSS. Finally, Section VII concludes the paper and describes future works.

II. Related Works

This section discusses the background and related works in task selection and scheduling for energy optimization of IoT systems, and reinforcement learning applied to IoT.

A. Task selection and scheduling

Energy-efficient designs are studied and developed, especially for edge computing, where the energy is more challenging to create than in a cloud environment. The authors of [5] focus on the optimization of duty cycles for edge devices. To ensure sustainable performance, they introduce a model to compute the minimum required size of battery for maintaining the embedded device's sustainable operation. An increase in harvested energy can lead to an adjustment of the duty cycle, enhancing performance while staying within the specified energy budget. In [6], the trade-off between latency and energy consumption in the case of mobile edge computing is found using a modified NSGA-II optimization algorithm. The authors of [7] follow a CPU-oriented approach where the tasks at the edge are distributed optimally among CPUs, based on the load of the CPUs. In [8], a strategy is introduced to distribute tasks among multiple nodes, allowing for local execution or delegation to peer nodes. The proposed mechanism efficiently determines task allocation through a two-step decision-making process, employing a classification technique and utility theory in the model.

B. Reinforcement learning applied to IoT

Classical reinforcement learning is used in energy optimization IoT research: Q-learning [9] and SARSA [10]. The tabular Q-learning approach, SARSA, and other traditional reinforcement learning methods face limitations when applied to problems with extensive state-action spaces. These limitations arise from their struggle to achieve effective generalization of the value function and policy function within such expansive environments. Deep reinforcement learning [11] and long short-term memory (LSTM) methods [12] generalize better high-dimensional state-action spaces.

In [13], the authors tackle the challenge of designing energy-efficient heating, ventilation and air conditioning policies under varying conditions of the building environment, which are hard to model and will vary from case to case. They adopt a deep deterministic policy gradient method to learn their thermal control policy, which outperforms classical machine learning techniques. Each agent assumes the role of making localized decisions based on its observations. The authors of [14] combine deep RL and LSTM networks to optimize the efficiency of task offloading. Their approach decreases the induced latency, the offloading cost and the task throw rate. A deep reinforcement learning-based resource allocation manager is proposed in [15] in the context of edge computing. They selected an Advantage Actor-Critic (A2C) model to assign workloads to edge data centers to minimize the energy consumed.

Our research differs from the cited papers because the proposed RL environment generalizes well for autonomous energy-harvesting stationary IoT systems (AEHSS). It allows the infrastructure or the hardware to change thanks to fine-tuning with updated parameters. Our deep RL model is also trained to select sets of tasks, rather than scheduling them. To our knowledge, this kind of model applied to AEHSS, or a similar class of IoT systems, does not appear in the literature.

III. Reinforcement Learning Environment Description

In contrast to supervised learning where the model is trained on labeled data to predict an output and to unsupervised learning where the model learns patterns and structures from unlabeled data with no explicit feedback signals, reinforcement learning (RL) works in an interactive setting where the agent (i.e., the model) receives rewards and updated observations based on its actions. RL does not require labeled data and can learn from raw sensory inputs, making it suitable for tasks where it is hard to obtain labeled data, or where the environment is continuously changing. The environment with which the agent interacts needs an action as input and outputs an updated state and a reward to the agent. The agent chooses actions with its policy.

The training phase is structured as follows: first, the exploration phase consists of exploring many strategies — often randomly — and find beneficial behaviors. After it is trained, the RL agent uses its knowledge and focuses on the best possible actions — less randomness.

In the case of RL applied to resource placement in autonomous energy-harvesting stationary IoT systems (AEHSS), the agent is a task allocator that must maximize the added value of tasks (i.e., services) performed by the system (Figure 1).

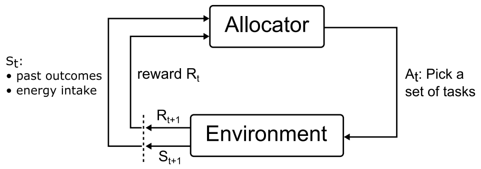


Fig. 1: Schema of reinforcement learning applied to task optimization of AEHSS.

A. Actions

The agent must select a set of tasks to be performed within an initial list of tasks $\{T_1, T_2, ...\}$. Every task T_i has an initial value V_i (from the end user's perspective) and an energy consumption cost E_i. Also, the cost of a set of tasks is equal to the sum of the costs of each task. For each iteration (one loop in Figure 1), the energy cost of the system's routine is equal to the sum of the costs of each task added to a base cost E_{base} that represents the energy consumption of the on and off transitions, the acquisition of data and the transfer of results. Although our approach is thought to generalize to any set of tasks, we share the data used in our implementation for the added values and energy costs in Table I. They are taken from measures of an actual AEHSS example, a precision beekeeping system [16].

Task name	Cost (joules)	Value (relative terms)
Task 1	95	60
Task 2	77	50
Task 3	87	55
Task 4	60	40
Task 5	70	45
Task 6	20	10
Base cost (always mandatory)	273	-

TABLE I: Costs and values of the six tasks.

B. Observations (States)

At the initialization of the environment and after every action, the environment generates an observable state made of two variables (far left side of Figure 1):

- The past outcomes: whether the system was correctly executed at the n last iterations, t referring to the current iteration;
- The energy output of the solar panel to the battery during the latest time interval: E_t. This value is estimated thanks to the model that will be described in Section IV.

The past outcomes are a list of size n. In our case, we set the value of n to 5. Together with our wake-up frequency of 5 minutes, it means that the agent can observe its history up to 25 minutes into the past. This middle-term memory is intended for the agent to estimate how well the system performed during the recent past iterations. A value of the list can take three different values that describe the state of the energy budget available. The first value, 0, is used when the current action cannot be performed because its energy cost is larger than the current energy budget B. The second, 1, describes a critical low-battery mode when the currently available budget falls between a threshold value B_{critic} and strictly over 0. The third value, 2, refers to a healthy energy budget: the resulting budget after the processing of the tasks is greater than B_{critic}. The threshold value B_{critic} is set to 25% of the maximal battery level. This value is selected because a Li-Polymer battery, which is used in the system in [16], has a flat discharge curve.

C. Rewards

Algorithm 1 represents one update of the environment. There are four main categories of rewards given to the agent after every action performed:

- A negative maximal penalty P_{max} if tasks cannot be performed because the energy budget is empty: $B = 0$ after the execution of the desired action (routine fails: line 2).
- 0 if tasks cannot be performed because of random fail (routine fails: lines 9 and 13).
- A negative penalty if the budget falls below B_{critic} but stays over 0 (routine succeeds: line 16). This penalty's absolute value is inversely proportional to the remaining budget and the multiplicative constant (0.05) is chosen so that this penalty linearly increases until the maximal penalty as the budget falls to 0.
- The sum of the positive values of all tasks if all tasks can be performed and the energy budget stays over B_{critic} (routine succeeds: lines 8 and 12). Each task's value is affected by the recentness of the last task's occurrence (O_i). The longer since the last occurrence, the larger the multiplier. This reflects the practical need to circle through all tasks regularly. The increase is capped after it reaches c iterations so that the multiplier is capped at m^c. In our case, based on our actual data and experience for AEHSS in precision beekeeping, m and c are set to 1.15 and 10, respectively.

D. Environment rules

The task allocation process is repeated every time a system wakes up. The duration between two consecutive iterations is not shown to the agent. This parameter is linked to the amount of energy gained by the system between iterations, thanks to the battery charge. In our case, the wake-up cycle is periodic, and its frequency is set to 5 minutes, which is a frequency that can realistically make the energy budget tend toward 0 if too many energy-heavy tasks are selected. Therefore, a greedy strategy that selects all tasks at all times would pull the energy budget to 0 in less than a day and generate negative rewards.

As Algorithm 1 represents one update of the environment, it takes as a parameter the action (a list of tasks) selected by the agent and returns a reward. The energy budget lies between 0% and 100% of the battery capacity. In our case, it corresponds to 0 and 266 400 J — the energy that can be stored in a 20 000 mAh / 3.7 V power bank battery.

Algorithm 1: Environment algorithm for processing one action

```
Input     : Action $\{T_a, T_b, \dots\}$
Output    : Reward $r$
Variables: Set of tasks: $\{T_1, T_2, \dots\}$, their values $V_i$ and their cost (energy)
           $E_i$ ; number of steps since their last successful occurrence $O_i$ ;
           task's value multiplier $m$ ; multiplication of iterations' limit $c$;
           energy budget $B$ ; critical budget $B_{critic}$ ; weather $W$ ; past
           outcomes $[S_{t-1}, S_{t-2}, \dots, S_{t-n}]$ ; current outcome $S_t$ ; the
           base cost of the system $E_{base}$ ; maximal penalty $P_{max}$
1  $B = max(0, B - (E_{base} + \sum_i E_i))$
2  if $B = 0$ then
3      $r = -P_{max}$
4      $S_t = 0$
5  end
6  else
7      if $W$ is "clear" then
8          95% of the time: $S_t = 2$; $r = \sum_i V_i \times m^{min(c, O_i - 1)}$
9          5% of the time: $S_t = 0$; $r = 0$
10     end
11     else
12         90% of the time: $S_t = 2$; $r = \sum_i V_i \times m^{min(c, O_i - 1)}$
13         10% of the time: $S_t = 0$; $r = 0$
14     end
15     if $B < B_{critic}$ then
16         $r = r \times (B - B_{critic}) \times 0.05$
17         $S_t = 1$
18     end
19 end
20 return $r$
```

If the sum of the tasks' energies is greater than the energy budget, the routine fails, the latest outcome (0 in this case) is pushed into the history and the maximal penalty is given as a reward (line 2). If the sum of the tasks' energies is lower than the energy budget (line 6), the routine succeeds 95% of the time if the weather category is "clear" (line 8) and the latest outcome pushed into the history is 2. For other weather categories (line 11), the completion rate is 90%. For the other 5% and 10%, the routine fails because of unknown reasons: the reward and the latest outcome are both equal to 0. This behavior mimics the deployed system introduced in [16]. When the routine succeeds, the energy budget decreases by the sum of the base cost (wake-up, data collection, transfer and shutdown) and the sum of the energies of the selected tasks (line 1). If the new energy budget B falls below B_{critic} (line 15), a negative penalty, smaller in absolute value than P_{max}, is returned and the value of 1 is pushed into the history of outcomes.

The environment uses the weather from the year 2022 to simulate its weather component. Between iterations (i.e., between two actions chosen by the agent and processed by the environment), the environment updates its date and time, the past outcomes and the energy budget. The first two are straightforward updates as they only depend on the routine frequency and the current outcome, respectively. The third will be introduced in Section IV.

IV. Prediction of Solar Power Intake

For the energy budget update, the photovoltaic energy case is considered, and a prediction model inspired by [17] is built. This cited article studies the different machine learning approaches for predicting solar power intake. The objective is to predict the solar power intake at a time interval using meteorological data and the position of the sun.

A. Sources of the predictor variables

We reproduce a dataset with weather observations and solar power intake over time. The weather data used as part of the predictor variables are from the Météo-France[1] station of Lyon-Saint Exupéry airport, France. 11 parameters, gathered every 3 hours, are selected to be part of the predictor variables: temperature, humidity, air pressure, precipitations of the last 3 hours, horizontal visibility, nebulosity (global) and the nebulosities of 5 different cloud layers. In addition to these 11 variables, the zenith, the azimuth and the number of seconds elapsed in the current day are also part of the predictors.

B. Experimental system for the collection of the target variable

The energy intake of the solar panel is the target variable — the metric that the models must predict.

The experimental system for the collection of photovoltaic data involves an east-oriented solar panel and an appropriate resistor that is directly wired the panel. Figure 2 shows the data collected over the 12th and the 13th of April 2023. Data points are sampled at 3.6 Hz and each represents the average current of 250 measured values. The two represented days have different weather conditions: the first is a fully rainy day and the second is a partially cloudy day, hence the higher energy production on the 13th.

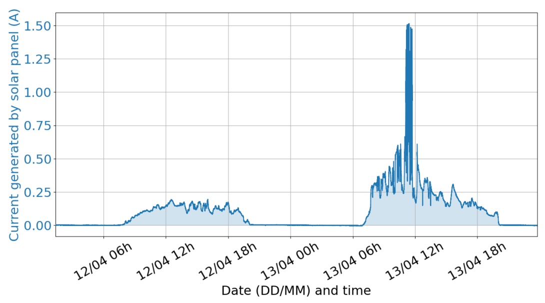


Fig. 2: Photovoltaic energy production of east-oriented solar panel

Although we collect energy production data at a rate of 3.6 Hz, the predictor variables — the weather data — are sampled in chunks of 3 hours. Therefore, for compatibility purposes during the training phase between the two sets of variables, we resample the energy production data in chunks of 3 hours, so that the goal is to predict the total amount of energy created from the previous hour and a half to the next hour and a half. In other words, for each date and time from the weather data, the corresponding value of the photovoltaic production is equal to the average over 3 hours putting this date and time at the middle of the 3-hour window.

[1]https://donneespubliques.meteofrance.fr

C. Prediction models

Before training, the predictor data is first centered and scaled by removing the mean and dividing it by the standard deviation. Then, the train-test split is set at 80%-20%, keeping both splits continuous — i.e., the test dataset is either all before or all after the training dataset. Several architectures of machine learning prediction models are selected to fit the data. For the accuracy metric, [17] chooses to scale the existing mean absolute percentage error (MAPE) metric to eliminate scale dependency, variations of seasonality, and prevent this metric from becoming very large when the produced energy is close to zero. We select MAPE as our main accuracy metric because it still eliminates seasonality, and we choose to take into account its large values in our models for small amounts of energy intake. Indeed, we do not want our solar energy intake estimator to give non-null values during the night. Table II shows the performances of models ordered from best MAPE to worst. The other performance metrics are mean squared error (MSE), R-squared (R^2) and the maximal error among the test set. The base unit for the target variable is the milliampere. The results confirm that random forest regressors (RFR) are quality models for solar intake prediction (as shown in [17]). In our case, 5 or 10 estimators, a hyperparameter of RFR, achieve the best results. This is lower than the 30 estimators, which are the best settings for the cited article. This difference in value can be explained by the difference between datasets as predictor variables differ between the two study cases. Also, differences can be explained by weather data observations being spaced by 3 hours in our case as opposed to 1 hour in the cited article.

Model	**MSE**	R^2	**Max Error**	**MAPE**
RFR (5 estimators)	4843	0.80	392.5	4.0e+14
RFR (10 estimators)	3981	0.84	360.6	4.3e+14
RFR (20 estimators)	6693	0.72	426.3	4.7e+14
RFR (30 estimators)	7156	0.70	432.3	5.2e+14
RFR (40 estimators)	7465	0.69	435.0	5.5e+14
RFR (50 estimators)	6953	0.71	414.2	5.7e+14
RFR (70 estimators)	7079	0.71	415.5	6.1e+14
RFR (200 estimators)	7248	0.70	435.2	8.7e+14
RFR (100 estimators)	7087	0.71	423.0	1.1e+15
Gradient Boosting Regressor	7374	0.69	471.1	8.7e+15
ElasticNet	12414	0.49	528.7	2.8e+16
SVR	27655	-0.14	656.0	3.3e+16
SGD Regressor (L1 norm penalty)	10184	0.58	464.9	5.2e+16
SGD Regressor (Elastic net penalty)	10192	0.58	465.1	5.2e+16
SGD Regressor (L2 norm penalty)	10193	0.58	465.1	5.2e+16
Ridge Regression	9238	0.62	417.7	5.8e+16
Linear Regression	9304	0.62	407.5	6.4e+16
Kernel Ridge Regression	31379	-0.30	582.5	1.8e+17

TABLE II: Summary of machine learning models' performances for the prediction of solar energy production. Models are ordered from lowest MAPE to highest.

Figure 3 shows the actual and predicted energy data in chunks of 3 hours. In this graph, the absolute average error is 37.8 mA per chunk of 3 hours. Also, the non-absolute average error (actual values subtracted from predicted) is -17.5 mA, showing that the model's predictions underestimate the actual values We also share the daily absolute average error for the nine full days displayed in Figure 3 from the 4th to the 12th of March 2022: 25.7 mA. The non-absolute error (actual values subtracted from predicted) of daily averages is -18.0 mA, which also shows the tendency of our model to underestimate photovoltaic production. It is crucial to obtain a modest model which rather underestimates the actual value. Otherwise, when this model is embedded into the RL environment, there will be a risk of non-null estimations of the energy budget when the battery is empty.

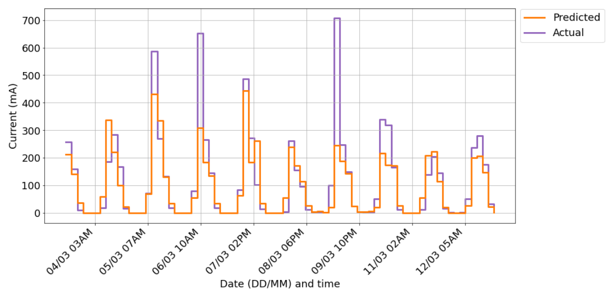


Fig. 3: Predicted and actual values of solar intake for the test set, using RFR with 5 estimators

For the following section, the RFR with 5 estimators is selected to be embedded into the RL environment to produce estimates of the solar energy intake and, therefore, update the energy budget at every step.

V. RL Methods for Task Selection Models

For the training algorithm, four reinforcement learning (RL) methods are selected for comparison: Proximal Policy Optimization (PPO), Recurrent PPO (RPPO), Trust Region Policy Optimization (TRPO) and Advantage Actor Critic (A2C). We use the implementation introduced by [18] for these methods.

A. Proximal Policy Optimization

PPO was first introduced by OpenAI[2] in 2017. It is a policy-based RL method that has proven to obtain state-of-the-art results for different benchmarks [19]. It relies on an exploration strategy that allows the agent to collect data during a certain number of iterations by updating its policy before processing this data. This exploration strategy reduces the variance of updates compared to methods that update the policy at every step in the environment. PPO is designed to be a simple, scalable and effective algorithm for learning policies that can operate in high-dimensional continuous action spaces. It uses a clipped surrogate objective function to update the policy parameters, which ensures that the new policy is not too far away from the previous policy to maintain stability during training. PPO's advantages include its simplicity and ease of implementation, fast training time and good sample efficiency.

B. Recurrent Proximal Policy Optimization

Recurrent PPO's ideas were introduced in [20]. RPPO is a policy-based RL method that uses recurrent neural networks in order to model temporal dependencies for time-series decision-making tasks. It uses the Kronecker-factored approximation

[2]https://openai.com/research/openai-baselines-ppo

(introduced in [21]) to efficiently compute the inverse Hessian matrix of the policy. RPPO's advantage is its ability to handle temporal dependencies and long-term credit assignments, as well as its good performance on tasks with high-dimensional inputs, such as video games and robotic control tasks. One of the drawbacks of RPPO is its computational complexity, which can limit its scalability. It has achieved state-of-the-art performance on benchmarks like the OpenAI Gym robotics control suite.

C. Trust Region Policy Optimization

TRPO was introduced in [22]. TRPO is close to PPO, as it uses a trust region constraint to ensure that the policy update is not too aggressive, which can prevent the policy from diverging during training. As opposed to PPO, TRPO uses the Kullback-Leibler divergence to limit the difference between the old and new policies. It can handle stochastic policies well and has good convergence properties, but can sometimes face the computational complexity challenge for large-scale problems.

D. Advantage Actor Critic

A2C (introduced in [23]) is an actor-critic RL method that uses parallel agents to collect experience and a central critic element to estimate the value function. A2C combines the advantages of policy gradient methods (such as PPO) with value-based methods (such as Deep Q-Network) by using both an actor and a critic network to estimate the policy and the value function, respectively. The use of parallel agents improves sample efficiency and speeds up training. A2C can suffer from high variance in the gradient estimates, which can lead to slow convergence or instability.

E. Training phase implementation and optimization

The performance of a model is defined as the cumulative reward (defined in Section III-C) at the end of the simulation. 108 models are trained for each RL method to obtain representative average values when grouping their performance scores. They are all trained through 1 000 000 iterations of the RL environment. The discount factor hyperparameter is set to 0.99 for all models.

During the training phase, we aim to avoid overfitting, which occurs when a model becomes too complex and is too closely fit to the training data, resulting in poor generalization to new data. To prevent this phenomenon, the performances of models are tested at regular iteration intervals during the training phase. The performance check relies on the performance of a given model at a one-month best-effort strategy simulation, starting at a random time, allowing to keep high granularity along the training phase and to save a model during the middle of the training phase if the cumulative reward is positive and better than at any previous check.

F. Implementation of heuristics for task allocation

We provide a point of comparison for the RL models with two heuristics implemented manually: a simple one and an advanced one. Both heuristics are fed the same observations of the environment as the RL agents: the past outcomes and the energy output of the solar panel.

The first heuristic, introduced in Algorithm 2 performs a random selection of two tasks if the success history is encouraging, and selects zero tasks otherwise.

Algorithm 2: Simple heuristic for selecting a subset of tasks at iteration t

```
Variables: Set of tasks: $T = \{T_1, T_2, ...\}$
Input    : List of past outcomes of size $n$: $[S_{t-1}, S_{t-2}, ..., S_{t-n}]$
Output   : Set of tasks $T_{selected}$
1 $confidence = \frac{\sum_{i=0}^{n-1} S_{t-i}}{2*n}$
2 if $confidence < 0.5$ then
3     return [ ]
4 end
5 else
6     $T_{selected} = randomSample(T, 2)$
7     return $T_{selected}$
8 end
```

In the second heuristic presented in Algorithm 3, the input parameters are the same as the RL agent's: the energy produced and the past outcomes. It selects a set of tasks based on a threshold that considers all parameters.

Algorithm 3: Advanced heuristic for selecting a subset of tasks at iteration t

```
Variables: Set of tasks: $T = \{T_1, T_2, ...\}$
           Their cost (energy) $E = \{E_1, E_2, ...\}$
           The best-case solar-produced energy over a time interval: $E_{max}$
Input    : Energy produced at time interval $t$: $E_t$
           List of past outcomes of size $n$: $[S_{t-1}, S_{t-2}, ..., S_{t-n}]$
Output   : Set of tasks $T_{selected}$
1 $R_{energy} = \frac{E_t}{E_{max}}$
2 $confidence = \frac{\sum_{i=0}^{n-1} S_{t-i}}{2*n}$
3 shuffle($T$)
4 $E_{selected} = 0$ ; $T_{selected} = [\ ]$
5 if $confidence < 0.5$ then
6     return $T_{selected}$
7 end
8 for $T_i$ in $T$ do
9     if
        $\frac{E_{selected}}{\sum E_i} > (confidence - 0.2)^{1.2} * \min(1, (1.3 - R_{energy}))$
      then
10        break
11    end
12    $E_{selected} = E_{selected} + E_i$
13    $T_{selected}.append(T_i)$
14 endFor
15 return $T_{selected}$
```

VI. Results

In this section, the benchmark chosen to evaluate the performances of models is a 30-day simulation taken during the summer of 2022. The performances of models are compared with the ones of two heuristic methods, which are defined in Algorithms 2 and 3.

Figure 4 shows the performances of all RL methods averaged over each category's 108 models. We observe that TRPO models produce, on average, the best performances and the ones with the smallest variances across different models.

Compared to the TRPO models, A2C, RPPO and PPO suffer from more inconsistency. The standard deviation over the different models (displayed in brighter colors on Figure 4) is higher than TRPO by a factor of 8.5, 5.5 and 3.7 for A2C, RPPO and PPO, respectively. However, the best of each method still achieves performances that are in the same order as TRPO's best performance – our best reinforcement learning model. Table III summarizes the best model's performance for each method over a 30-day simulation during July 2022. The discount factor is set at 0.99 for every model, but the number of iterations used to collect experience before updating the policy during the training phase, n_steps, can vary. The best models show a peak of performance for values of 512, 1024 or 2048. In our case, the successive routines performed in the environment are spaced by five minutes. In practice, it means that the best-performing agents collect exploration data between 1 day and 19 hours ($n\_steps = 512$), and 7 days and 3 hours ($n\_steps = 2048$) before updating their policy. This behavior is wanted because we prefer to feed several days of data rather than just a few hours, which would be very biased because of the instantaneous weather and the day-night cycle.

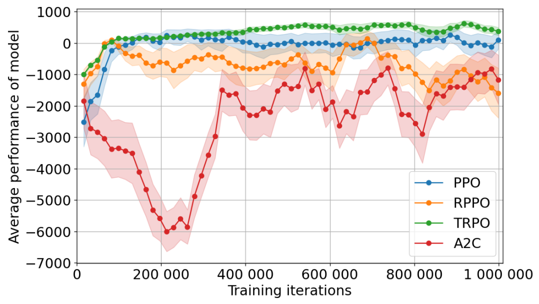


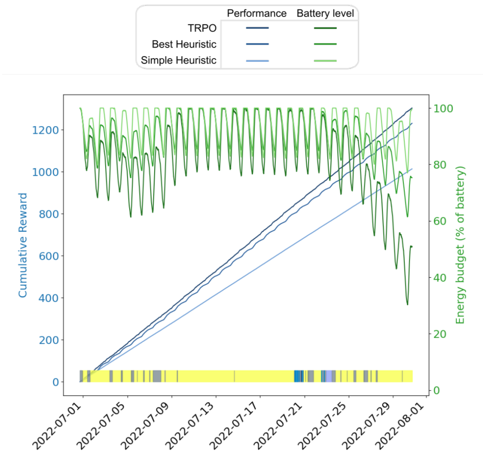
Fig. 4: Performance of each method over the number of training iterations

Algorithm	Training steps	n_steps	Best model's perf.
TRPO	999 424	2048	1291
PPO	835 584	512	1290
RPPO	622 592	1024	1267
A2C	901 120	2048	1231
Advanced heuristic	-	-	1226
Simple heuristic	-	-	1014

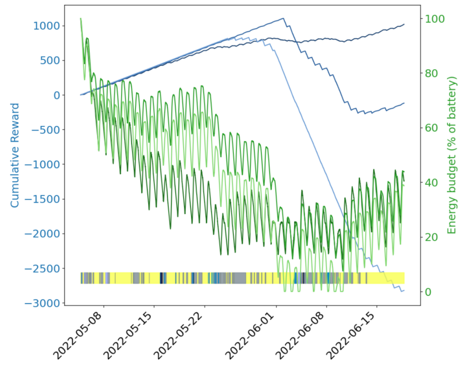
TABLE III: Best model's performance (cumulative reward) for every method tested on a 30-day simulation. Models are ordered from the highest performance to the lowest.

Another observation of Table III is the number of training steps at which the best models' performances are achieved. It is at first counterintuitive that the RPPO's best version is met after 622 592 rather than a value closer to 1 000 000 iterations, where the models' final form is met. However, the technique described in Section V used to prevent overfitting allows us to pinpoint best-performing models before the end of the training phase.

Figure 5 highlights the best obtained TRPO-based model with one 30-day simulation (Figure 5a) and one 45-day simulation (Figure 5b). It corresponds to the TRPO model whose metrics are displayed in Table III. We display the data of the best TRPO model (darker color), our advanced heuristic (middle color) and our simple heuristic (clearer color). The cumulative performance over time and the energy budget are shown in shades of blue and green, respectively. At the bottom of the graph, we display the weather category (introduced in Section III-D) over time. Yellow, gray, and blue correspond to clear, cloudy, and rainy weather categories.



(a) Cumulative reward and battery level for one 30-day simulation of the best TRPO-based trained RL agent



(b) Cumulative reward and battery level for one 45-day simulation of the best TRPO-based trained RL agent

Fig. 5: Comparison of performances and energy budget's evolutions for the two heuristics and our best model

The 30-day simulation (Figure 5a) is chosen to be during July 2022, which is an average summer month in terms of weather. In these conditions, we want to compare the three strategies when the weather conditions are at their sunniest for several weeks in a row, so the photovoltaic intake is maximized. TRPO outperforms the two heuristic strategies after 30 days. We note a difference between the two energy usage strategies: the heuristic cannot switch gears, whereas TRPO uses the battery to obtain battery levels close to the 25% limit.

The 45-day simulation (Figure 5b) starts at the beginning of May 2022 and ends around the end of June. We choose this 45-day window for its weather which is cloudier and rainier than the window of Figure 5a, resulting in a reduced energy intake. This difference explains the faster decrease of the battery percentage over time in Figure 5b for all strategies: the energy intake of the solar panel into the battery becomes lower under cloudy or rainy conditions than sunny conditions. The RL approach shows adaptation and even though the agent only sees the history of the five last iterations together with the instantaneous energy intake, the energy budget close to the critical value does not affect the cumulative reward as much as the heuristics: it stays, at worst, stable at the scale of a week (from the 1st to the 8th of June for example), and quickly recover when better weather conditions are back (after mid-June). Although close to practically damaging values, the energy budget is kept within the acceptable range as it never flirts with 0%. More importantly, the RL approach is more consistent over 45 days in terms of reward variations.

VII. Conclusion

In this paper, we defined a reinforcement learning environment applicable to the case of energy-efficient task allocation in AEHSS. In parallel, we validated the work presented in [17], identifying random forest regressors as the best predictors of photovoltaic energy intake.

We repeatedly trained four types of RL methods: PPO, TRPO, RPPO and A2C. During the training, the overfitting phenomenon is reduced by iteratively measuring the performance of a model. The best-proposed models outperform specifically designed heuristics. TRPO-based models produce the best results in terms of maximal performance: 1291 cumulative reward over a one-month simulation, which is an improvement of 5% compared to the advanced heuristic. Our RL-based solution is also better at managing energy levels close to values considered critical by the environment. Our model design would easily apply to a different environment, whereas heuristic would need to be manually tuned.

In future works, we aim to make the rules of the environment more complex to model weather and energy harvesting dynamics more realistically. The explicability of the strategies of the best models also remains to be analyzed in the future. Another important point is to implement a multi-client/multi-server environment where the agent will not only select tasks but also orchestrate them at the edge and in the cloud. Finally, the validation of our models will be performed through actual deployment in the field.

Author Contributions

Hugo Hadjur: Conceptualization, Methodology, Software, Validation, Writing - Original Draft, Writing - Review & Editing, Visualization. **Doreid Ammar**: Supervision, Writing - review & editing. **Laurent Lefevre**: Supervision, Writing - review & editing.

References

[1] D. A. Chekired, M. A. Togou, L. Khoukhi, and A. Ksentini, “5g-slicing-enabled scalable sdn core network: Toward an ultra-low latency of autonomous driving service,” *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 8, pp. 1769–1782, 2019.

[2] H. Hadjur, D. Ammar, and L. Lefèvre, “Toward an intelligent and efficient beehive: A survey of precision beekeeping systems and services,” *Computers and Electronics in Agriculture*, vol. 192, p. 106604, 2022.

[3] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*, 1st ed. Cambridge, MA, USA: MIT Press, 1998.

[4] H. G. Lee and N. Chang, “Powering the IoT: Storage-less and converter-less energy harvesting,” in *The 20th ASP-DAC*, 2015, pp. 124–129.

[5] A. Kansal, D. Potter, and M. B. Srivastava, “Performance aware tasking for environmentally powered sensor networks,” in *Proceedings of SIGMETRICS’04*, 2004, pp. 223–234.

[6] L. Cui, C. Xu, S. Yang, J. Z. Huang, J. Li, X. Wang, Z. Ming, and N. Lu, “Joint optimization of energy consumption and latency in mobile edge computing for Internet of Things,” *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4791–4803, 2019.

[7] S. Malik, S. Ahmad, I. Ullah, D. H. Park, and D. Kim, “An adaptive emergency first intelligent scheduling algorithm for efficient task management and scheduling in hybrid of hard real-time and soft real-time embedded IoT systems,” *Sustainability*, vol. 11, no. 8, 2019.

[8] K. Kolomvatsos and C. Anagnostopoulos, “Multi-criteria optimal task allocation at the edge,” *Future Generation Computer Systems*, vol. 93, pp. 358–372, 2019.

[9] E. Barrett and S. Linder, “Autonomous HVAC control, a reinforcement learning approach,” in *Machine Learning and Knowledge Discovery in Databases*, 2015, pp. 3–19.

[10] A. Zenger, J. Schmidt, and M. Krödel, “Towards the intelligent home: Using reinforcement-learning for optimal heating control,” in *KI 2013: Advances in Artificial Intelligence*, 2013, pp. 304–307.

[11] T. Wei, Y. Wang, and Q. Zhu, “Deep reinforcement learning for building HVAC control,” in *2017 54th ACM/EDAC/IEEE DAC*, 2017, pp. 1–6.

[12] Y. Wang, K. Velswamy, and B. Huang, “A LSTM recurrent neural network based reinforcement learning controller for office heating ventilation and air conditioning systems,” *Processes*, vol. 5, no. 3, 2017.

[13] G. Gao, J. Li, and Y. Wen, “Energy-efficient thermal comfort control in smart buildings via deep reinforcement learning,” 2019.

[14] Y. Tu, H. Chen, L. Yan, and X. Zhou, “Task offloading based on LSTM prediction and deep reinforcement learning for efficient edge computing in IoT,” *Future Internet*, vol. 14, no. 2, 2022.

[15] S. Pérez, P. Arroba, and J. M. Moya, “Energy-conscious optimization of edge computing through deep reinforcement learning and two-phase immersion cooling,” *Future Generation Computer Systems*, vol. 125, pp. 891–907, 2021.

[16] H. Hadjur, D. Ammar, and L. Lefevre, “Services orchestration at the edge and in the cloud for energy-aware precision beekeeping systems,” in *2023 IEEE IPDPSW*, 2023, pp. 769–776.

[17] F. A. Kraemer, D. Palma, A. E. Braten, and D. Ammar, “Operationalizing solar energy predictions for sustainable, autonomous IoT device management,” *IEEE Internet of Things Journal*, vol. 7, no. 12, pp. 11 803–11 814, 2020.

[18] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, “Stable-baselines3: Reliable reinforcement learning implementations,” *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021.

[19] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms,” *CoRR*, vol. abs/1707.06347, 2017.

[20] Y. Wu, E. Mansimov, S. Liao, R. Grosse, and J. Ba, “Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation,” in *Proceedings of the 31st NeurIPS*, 2017, pp. 5285–5294.

[21] J. Martens and R. Grosse, “Optimizing neural networks with kronecker-factored approximate curvature,” in *Proceedings of the 32nd ICML - Volume 37*, ser. ICML’15, 2015, pp. 2408–2417.

[22] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, “Trust region policy optimization,” in *Proceedings of the 32nd ICML*, vol. 37. PMLR, 2015, pp. 1889–1897.

[23] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, “Asynchronous methods for deep reinforcement learning,” in *International Conference on Machine Learning*, 2016, pp. 1928–1937.