Distributed and Real-Time IoT Task Scheduling Approach via Decision Tree Reinforcement Learning

Ali Hamedi*, Amin Soltan-Beigi*, Amirali Ghaedi, and Athena Abdi

Faculty of Computer Engineering, K.N.Toosi University of Technology, Tehran, Iran

Abstract

This paper presents "DARTS", a dynamic and real-time task scheduling for the edge of the Internet of Things (IoT), employing a reinforcement learning-based decision tree. It optimizes execution time and energy consumption during task scheduling while considering specific IoT constraints, including task compatibility, safety, load balancing, and handling environmental changes. The proposed DARTS employs a differentiable decision tree (DDT) architecture to bridge the gap between structured interpretability and real-time adaptability. Moreover, it operates within an asynchronous advantage actor-critic (A3C) framework, optimized with proximal policy optimization (PPO). DARTS considers an adaptive reward and punishment system that refines scheduling decisions based on the main objectives and constraints. It is also capable of managing network volatility, integrating new devices while de-prioritizing unreliable options through a probability-based selection mechanism. To demonstrate the effectiveness of the proposed DARTS, experiments are conducted on various synthetic and real-world benchmarks, comparing results to those of related studies. Based on these experiments, DARTS outperforms selected heuristic, meta-heuristic, and learning-based studies in terms of joint optimization of execution time and energy consumption. Moreover, its response time is less than that of learning-based schemes, which makes it an appropriate candidate for real-time interactive systems.

Keywords: Internet of Things, Edge task scheduling, Decision tree, Reinforcement learning, Energy consumption.

*These authors contributed equally to this work.

1. Introduction

The Internet of Things (IoT) has emerged as a vast technology of interconnected devices to create smart environments. In the contemporary digital age, the importance of the Internet of Things is magnified by its ability to collect real-time data and improve decision-making processes in diverse applications such as smart cities, healthcare, and industrial automation. The ongoing advances in networking technologies have facilitated the widespread adoption of the Internet of Things (IoT), the increasing number of smart devices, and the growing demand for integrating artificial intelligence (AI) in various domains. IoT systems are built on a three-layer architecture. The first layer consists of IoT devices that collect data from the environment through sensors and perform everyday tasks. The second layer comprises multi-access Edge Computing (MEC) devices, which act as intermediaries, processing data closer to its point of generation to reduce response time and lighten the load on central servers. The third layer is Cloud Computing (CC), located on central servers, providing the highest processing capability among the three layers. This hierarchical structure facilitates communication and maximizes efficiency, but requires exact management strategies [1, 2]

Executing various application tasks on the various IoT layers leads to different performance, power consumption, cost, and network traffic [3]. The task scheduling process determines the appropriate processing unit for each application's task to maximize efficiency, reduce delays, and ensure security. In this context, the selection of IoT devices improves performance and security due to their proximity to the data source. However, it provides limited computational power and battery life. MEC devices are moderate to heavy computing resources that provide localized processing closer to the end-users. This reduces latency, preventing typical delays associated with remote cloud servers and improving response times. The CC layer is appropriate for executing application tasks that require significant computational overhead due to their large-scale data analytics. Processing in this layer leads to significant latency and security risks as a result of data transmission across multiple networks. Using a multi-objective task scheduling scheme within the edge of IoT determines the best assignment to balance computational load, energy consumption, latency, and security [4, 5, 6, 7].

Task scheduling, considering optimizing the design challenges, is a known NP-hard problem. For task scheduling on the edge of the IoT, various heuristic, metaheuristic, and learning-based methods are presented. These meth-

ods are employed statically at design time or dynamically at the runtime operation of the system [8, 9, 10]. Offline schemes are designed based on the predetermined structure of the system and disregard the continuous changes during execution. While Online task scheduling methods are employed during system operation, their computations should be low and parallelized.

Task scheduling in IoT systems is more challenging due to their unique characteristics. The resource constraints, employed in dynamic and heterogeneous environments, the need for scalability, energy efficiency, security, and real-time processing are some of the most important characteristics of these systems. In this context, an appropriate scheduling approach must first consider each layer's limitations and computation attributes to prevent overburdening and aging of the devices. Meanwhile, balancing the main parameters, such as energy consumption and computational demands of executive tasks [4, 11]. Previous heuristic and meta-heuristic schemes are mainly static and are unable to handle the dynamic nature of IoT systems, where devices frequently join or leave the network. Moreover, their response time due to their search-based schemes is high, which contradicts the real-time and interactive requirements of IoT environments [8, 12, 13]. Learning-based schemes, especially the ones associated with dynamic features like reinforcement learning, cover the mentioned limitations of heuristic and metaheuristic methods. However, lacks interpretability and scalability and has a high cost of fine-tuning in frequent variations of the network [14, 15, 16].

Our proposed DARTS aims at addressing the mentioned challenges through a multi-agent decision tree reinforcement learning (DTRL)-based scheduling scheme. This method combines the interpretability of decision trees with the adaptive and dynamic features of reinforcement learning, enabling effective cooperation among multiple independent agents. The decision-maker core of our proposed scheduling scheme is a Differentiable Decision Tree (DDT) that directs the input based on its characteristics to different paths. To optimize various parameters during scheduling, each level of the tree prioritizes different features. At the end of the tree traversal, a probabilistic distribution is derived for the current task to determine the most suitable device for its execution. During scheduling, the execution time and energy consumption are optimized, while the system's constraints, including task safety, task type, load balancing, and battery depletion, are met. These objectives and constraints are considered as the learning factors in the designed reward and punishment functions during tree traversal. The optimization algorithm updates the weights and probabilistic distributions of the nodes in the traversed

path based on the environmental feedback received, allowing the model to continuously improve its decisions over time without overfitting. The proposed learning scheme combines the Asynchronous Advantage Actor-Critic (A3C) framework with the Proximal Policy Optimization (PPO) algorithm, providing a robust foundation for stable learning. The main contributions of the proposed method are summarized as follows:

- Presenting an adaptive and interpretable task scheduling approach in IoT edge called DARTS employing decision tree reinforcement learning;
- Joint Optimization of execution time and energy during task scheduling while meeting the existing static and dynamic constraints of the environment and devices;
- Providing dynamic device assignment capability and scalability of the system through probabilistic reasoning and modular updates in the designed differentiable decision tree scheduling approach;
- Presenting a lightweight and multi-agent task scheduling approach appropriate for resource-constrained IoT systems due to its tree-based and data-informed characteristics.

The rest of the paper is organized as follows: Section 2 presents the literature review, and the preliminaries of the proposed method are presented in Section 3. The details of the proposed task scheduling approach are explained in Section 4. Section 5 represents the experimental results, and the conclusion remarks and future trends are summarized in Section 6.

2. Related Study

Task scheduling problems at the edge of IoT systems aim to determine the optimal resource and time step for executing tasks of applications while optimizing the system's design parameters and achieving high efficiency. Due to the heterogeneity of the processing elements of the IoT environment, this multi-objective optimization problem becomes more complex. In this context, several methods are presented to solve the problem in a central or distributed manner based on static or runtime processing.

Central task scheduling schemes are classified into heuristic, meta-heuristic, and machine learning-based approaches. Heuristic schemes approximate the

problem using greedy and local search methods, providing near-optimal solutions at a low cost. These methods mainly convert the multi-objective optimization to a single objective by summing the criteria [17, 18, 19]. In [17], the execution time and power consumption are considered as two main objectives and are combined through a weighted sum. To explore the design space and minimize the considered cost function, this scheme employed dynamic voltage frequency scaling (DVFS) and a search-based task allocation method. Various meta-heuristic methods like genetic algorithm, ant colony algorithm, particle swarm optimization (PSO), simulated Annealing and Tabu search are considered to solve this problem. These schemes are extended to multi-objective optimization, considering the main design challenges jointly. Meta-heuristic methods employ many parameters and calculations, making their adjustment and reuse inefficient and slow in dynamic situations [20, 21, 22, 12, 13].

Along with the advancement of artificial intelligence in various fields, machine learning schemes are employed in edge task scheduling problems. These methods are capable of solving this problem in the dynamic environment but require precise fine-tuning, have a long learning time, and are not interpretable [7, 11, 23, 24]. To address this, distributed methods are considered that are mostly based on game theory and federated learning. Game theory-based methods design a game to analyze the interactions among entities to maximize their self-interests with low complexity [25]. Later, game theory is combined with other schemes to propose several markove decision processes (MDP) and deep reinforcement learning-based (DRL) methods [14, 11, 23]. In federated learning-based scheduling schemes, the training process is performed on several edges and servers without data exchange [26, 27]. This learning scheme combines with DRL and greedy centralized methods to make scheduling more efficient.

Embedding interpretable schemes, such as decision trees, in the dynamic scheduling learning process could be effective. As a representative of tree-based schedulers, in [28], a single global DDT policy that trades off execution time and power consumption on a heterogeneous system is proposed. This method employs the preference vectors and achieves Pareto-efficient schedules for static system-on-chip fabrics. In [29], the explainability of the DDT structure in RL policies is studied for dynamic energy management applications. In [30], the decision tree is added to the RL process for task offloading decision in the IoT environment. However, these methods are single-agent and do not address the dynamic and constraint-rich environment in terms of

Table 1: Main Features of Related Studies and Proposed Method Based on the Characteristics of Edge Task Scheduling.

Method	Optimization Scheme	Deadline[1]	Safety[2]	Load Balancing[3]	Device Churn[4]	Explainability[5]
TEMS[17]	Heuristic	✓	×	✓	×	✓
PSO [31]	Meta-Heuristic	✓	×	implicit	×	×
NSGA-II[32]	Meta-Heuristic	✓	×	implicit	×	×
DQN[33]	Value-based RL	×	×	implicit	implicit	×
DRL[34]	Policy-based RL	×	×	implicit	implicit	×
DDPG [35]	Actor/Critic	✓	×	implicit	implicit	×
DARTS	DDT A3C-PPO	✓	✓	✓	✓	✓

[1] Optimize task deadline, [2] Appropriateness of allocated resource to tasks,[3] Distribut load equally on resources, [4] Manage add/remove devices, [5] Be interpretable

heterogeneous devices, shifting network conditions, and implicit factors such as battery dynamics and device churn.

Summarily, heuristic and meta-heuristic schemes are simple and explainable but static and limited in computation. While the central and distributed machine-learning schemes are capable of fine-tuning, they are not interpretable. Table 1 compares the main features of some heuristic, meta-heuristic, and learning-based related studies that are further employed to compare with the proposed method. Since edge task scheduling is employed in various IoT applications, its precision, interpretability, scalability, and efficient interaction with the environment are crucial considerations in the proposed DARTS.

3. Preliminaries

3.1. ***IoT Architecture Model***

IoT forms the backbone of modern smart systems, where efficiency hinges on the seamless coordination of countless devices. A three-tiered architecture, consisting of devices, multi-access edge computing (MEC), and cloud servers, is proposed to harness the full potential of these interconnected systems [1, 2]. In this architecture, a wide range of capabilities, including the rapid data processing of the edge and powerful analytics of the cloud, are embedded. The three-layer architecture model as the reference system for our proposed DARTS is shown in Fig. 1.

As this figure shows, the device layer consists of several devices with heterogeneous processing capabilities. These devices are battery-dependent and can process the determined set of applications' tasks. The MEC layer provides distributed processing capability through some servers at the network's

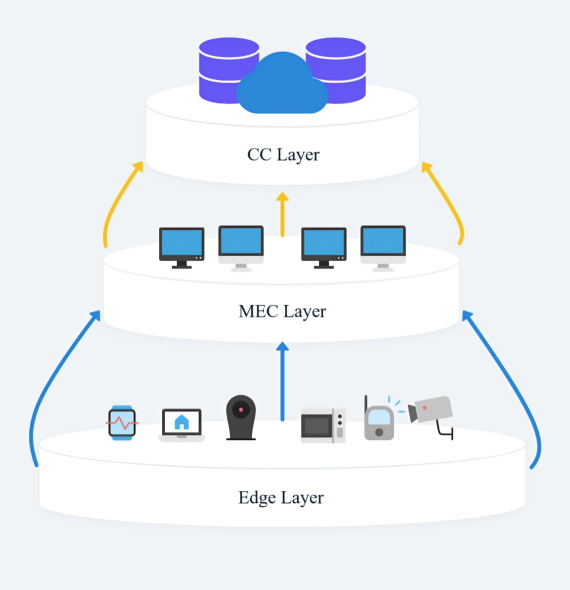


Figure 1: Architecture of the considered IoT system consisting of devices, MEC, and cloud server

edge. Lastly, the cloud layer provides high processing capability at the cost of network latency because it is far from the devices.

*3.2. **Design Challenges of IoT Systems***

IoT systems are utilized to perform services more efficiently in terms of execution time, energy consumption, cost, service quality, etc. [2, 36]. In this context, the execution time and energy consumption are the two main objectives of IoT systems, which are considered in our proposed scheduling method.

*3.2.1. **Execution Time***

Execution time is defined as the system's elapsed time to process the applications and their communication. Since each application consists of several connected tasks, the communication time of the tasks regarding their data transfer is also considered. This parameter is computed as follows:

$$T_{exe} = \frac{L}{f_c} + T_{comm} \tag{1}$$

where L, f_c, and T_{comm} represent the task's computation load, the operational frequency of the assigned processing unit, and the communication time, respectively. It should be noted that the communication time varies for transferring data among the different layers of the system. The communication times for data transfer from the device layer to the MEC layer via the 5G link and from the MEC layer to the cloud via the fiber are computed as follows:

$$T_{comm-MEC} = \frac{D_{in} + D_{out}}{R_{5G}} + 2 \times \delta_{5G} \tag{2}$$

$$T_{comm-CC} = \frac{D_{in} + D_{out}}{R_{fiber}} + 2 \times \delta_{fiber} \tag{3}$$

where D_{in}, D_{out}, R and δ represent the task's data entry and output sizes, the network transfer rates, and the transfer media delays, respectively. In our assumed architecture, R_{5g}, R_{fiber}, δ_{5G}, and δ_{fiber} are set to 1 Mbps, 10 Mbps, 5 ms, and 1 ms, consequently [17].

3.2.2. ***Energy Consumption***

IoT systems consist of various battery-dependent devices that operate autonomously on a limited energy budget. The energy consumption of a processing device is dependent on its operational characteristics, along with the transfer energy cost among the layers, as follows:

$$E = (C_{chip} \times V^2 \times f \times T_{exe}) + E_{comm} \tag{4}$$

where C_{chip}, V, and f represent the chip capacitance, voltage, and frequency, respectively. The communication energy cost is computed as follows:

$$E_{comm-MEC} = (\alpha \times \frac{R_{5G}}{1Mbps} + \beta) \times T_{comm-MEC} \tag{5}$$

where α is the variable power cost per Mbps and is considered equal to 52×10^5, and β represents the fixed power consumption during 5G transmissions, which is equal to 3.86412 [17]. Since the transmission power to the Cloud is fixed due to using fiber network connections, this parameter is considered equal to 3.65 multiplied by the cloud's communication time.

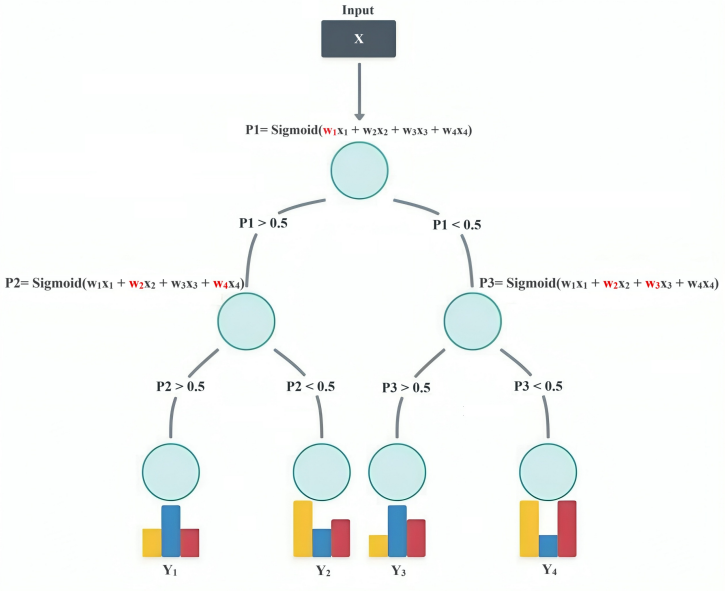


Figure 2: The structure of a sample DDT and its feature prioritizing procedure

3.3. ***Differential Decision Trees***

Differentiable Decision Trees (DDTs) are a powerful innovation in machine learning, designed to combine the interpretability of decision trees with the flexibility of neural networks. DDTs are structured as binary trees with a predefined depth. Each node in the tree serves as the routing or leaf to direct the decision-making flow and make the final decision, consequently. Unlike classical decision trees that commit to a single branch at each decision point, DDTs calculate a weighted probability of going to each side of the split [37, 38]. Fig. 2 shows the structure of a sample DDT and its ability to prioritize features.

As this figure shows, the coordination of the two types of nodes is a dynamic and adaptable structure. Since DDTs update only the nodes involved in an output selection, their learning is path-specific, significantly reducing computational complexity and making them ideal for real-time applications. The standard DDT is further improved to improve performance and adaptability. The soft tree updates the whole structure, including the nodes' probability distributions and weights, after each action to accelerate convergence. The clusTree employs a balanced K-Means to cluster the devices and distribute them evenly across leaf nodes to prevent overloading [39].

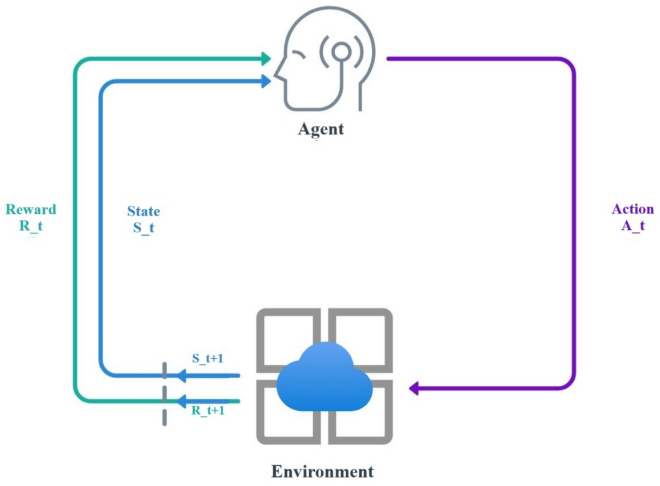


Figure 3: Reinforcement learning paradigm regarding its elements

3.4. ***Reinforcement Learning***

Reinforcement Learning (RL) is a learning paradigm where an agent learns to make decisions by interacting with its environment through trial and error. The core idea of RL revolves around the concepts of state, reward, and punishment. The agent aims to maximize cumulative rewards through repeated interactions, leading to more effective and optimized decision-making over time [40]. Fig. 3 shows the reinforcement learning paradigm considering its elements.

The policy optimization to improve decision-making in RL is performed by algorithms such as policy gradient and proximal policy (PPO). Policy Gradient methods directly optimize the policy by mapping from states to actions. Proximal policy optimization (PPO) improves upon policy gradient methods to provide greater stability and efficiency. It employs the actor-critic (A2C) method to improve the learning process of RL [40, 41]. Fig. 4 shows the actor-critic model in a reinforcement learning environment interaction.

4. Proposed Method

4.1. ***Problem Statement***

Our target problem aims to present a task scheduling scheme in IoT Edge that jointly optimizes delay and energy consumption while considering

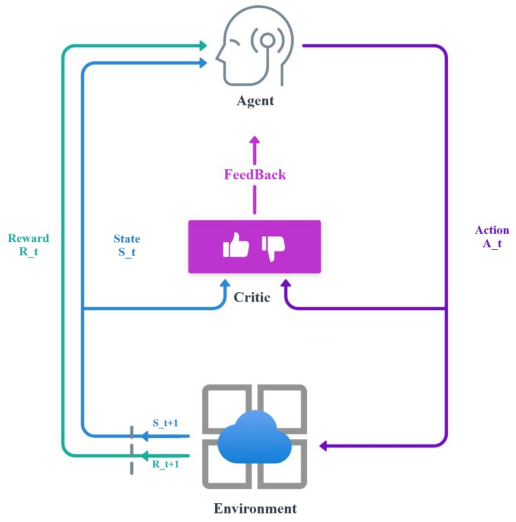


Figure 4: Actor-Critic model in reinforcement learning environment interaction

environmental constraints. Task scheduling in IoT Edge is a real-time process that adheres to system limitations and optimizes key performance metrics. During this process, the scheduler tends to assign each task to a device that is primarily capable of performing it, with consideration given to the type of task and security requirements. Furthermore, it must optimize energy consumption and processing time by selecting devices that execute tasks most efficiently. Meanwhile, considering the continuous changes and the increasing constraints of the environment is required. Ultimately, the system must balance the workload across the network and ensure task allocation does not concentrate on high-performance devices.

The description of this problem could be summarized as follows:

Given:

- A three-layered IoT architecture model consisting of various heterogeneous devices with unique processing capabilities, heterogeneous MEC servers, and a cloud server;
- Applications modeled as the directed acyclic graphs (DAG) of tasks and their interconnections. Each task has parameters including the task type, computational load, and additional constraints related to security or reliability requirements;

- Environmental limitations like the devices' battery levels and each task kind's compatibility influence its ability to process tasks.

Goals:

- Assigning the application's tasks to the appropriate devices, considering their specific type and security requirements;
- Optimizing energy consumption and execution time jointly during the assignment and scheduling processes;
- Considering the continuous environmental changes, including battery limitations, to prevent failures from low energy levels;
- Balancing workload by distributing the tasks to avoid over-reliance on a few high-performance devices;
- Presenting a lightweight and adaptive task scheduling approach suitable for resource-constrained IoT systems.

4.2. Proposed Reinforcement Learning-based Decision Tree Task Scheduling (DARTS)

4.2.1. Structure of the Differential Decision Tree

Our proposed DARTS employs a standard differential decision tree (DDT) structure for task scheduling. This version is employed in DARTS because it preserves interpretability and keeps inference lightweight, while avoiding additional clustering and hyperparameters. DDT is path-specific and only updates the nodes involved in an output selection. This reduces computational complexity significantly and makes them ideal for real-time applications. Moreover, its interpretability inherent in the decision tree structure offers significant advantages in understanding how decisions are made, which is particularly important in IoT systems where transparency is key. Due to our selection duty to choose an optimal device for each task, along with the interpretability of DDTs and the real-time nature of the IoT environment, we choose DDT as the core of our model. Each task is represented by its feature vector, which serves as the input to the model, and the corresponding output is a probability distribution over the available devices. The most suitable device for executing the task is selected based on these assigned probabilities.

The learning process in the proposed tree begins with determining the routing nodes due to the initial weights randomly set at each routing point.

Each leaf node determines an assignment option for the current task, focusing on the various objectives. In the proposed tree, the number of splits for each task depends on its corresponding decision complexity. Thus, the exploration level of the tree depends on resolving appropriate solutions based on the problem's objectives. If the problem constraints are minimal and an appropriate answer exists that meets all criteria, the model resolves the task within that node without exploring additional branches. However, a single probability distribution within the node cannot satisfy all conditions in complex scenarios with competing conditions—such as tasks that vary significantly in type, priority, or required constraints. In such cases, more splitting resolves the problem by specializing the paths for optimal resolution. The proposed model identifies specific input features that are most likely to introduce conflicting requirements and, in response, refines the routing node's learnable weights. This adjustment creates distinct pathways for tasks with unique attributes or constraints, ensuring the system's adaptability to multiple objectives.

As the model learns to distinguish features that demand different routing paths, it optimizes each route to direct tasks toward the most suitable leaf nodes. This iterative process ceases when the model can consistently solve each task type at the designated leaf nodes, efficiently tailoring decisions to meet the objectives without over-complicating the tree structure. This adaptive splitting ensures that each node specializes in specific decision-making needs, creating a responsive and efficient routing process throughout the tree. The details of the employed learning process are presented in the following sections. Alg. 1 summarizes the DDT structure and its settings.

4.2.2. ***MDP of DARTS***

Since task scheduling is a dynamic process at runtime, we apply learning capability to our proposed tree from environmental feedback to make it self-adaptive. In this context, we select RL as an appropriate option that excels by continuously adapting to real-time feedback. IoT systems often face unpredictable events such as fluctuating network loads and device availability. Reinforcement learning handles these challenges effectively, learning adaptive strategies autonomously without requiring explicit programming. This scheme is associated with the constructed DDT of our proposed task scheduling approach to adjust the weights and routing process.

The mathematical framework of the employed learning process for the DDT is described in terms of the MDP elements. The state serves as the

Algorithm 1 Details of the differential decision tree construction in DARTS

Inputs: X: Input feature vector, N_i: Number of input features, N_o: Number of output classes, D: Current depth in the tree, D_m: Maximum depth of the tree.
Outputs: Probability distribution of the leaf (P(Y))

```
- Tree Initialization:
DDT(N_i, N_o, D, D_m)
if (D==D_m) then
    Initialize P∈ R^{N_i}
    else
        Initialize W 𝒩(0,0,1), bias=0
        Left = DDT(N_i, N_o, D + 1, D_m)
        Right = DDT(N_i, N_o, D + 1, D_m)
    end
end
- Routing Process:
if (D == D_m) then
    return P(Y)
    else
        Decision ← σ (X.W)+b
        if (Decision > 0.5) then
            P(Y) ← Decision * Right(X)
            else
                P(Y) ← (1 - Decision)* Left(X)
            end
        end
    end
end
return P(Y)
```

input to our scheduling agents and is represented in two main configurations: task-only and task-device modes. Each offers a different balance of scalability, complexity, and adaptability within the IoT system. In the first mode, only the task's features are provided as inputs to the DDT. This approach enhances scalability, allowing devices to be added or removed without modifying the model. It is particularly suited for dynamic IoT environments that require adaptability and efficient scaling capabilities. The second mode incorporates task and device features as inputs to make the decision-making process more detailed and informed. However, it increases model complexity, slows the learning process, and limits scalability. Changes in the number of devices or their features require model adjustments, making this approach more suitable for stable IoT networks. Additionally, a feature clustering technique called clusTree is applied at routing nodes in this mode, which helps focus the decision process and reduces computational demands during training. Following the scalability and adaptability in dynamic environments, our proposed DARTS employs the task-only input mode. The action involves selecting an appropriate device guided by the DDT's probability distribution

output. This selection is then evaluated based on the reward or penalty it receives, gradually steering the model toward choosing the optimal device for each application's task.

The third element of our proposed MDP is the reward function. Our considered reward is focused on operational efficiency and balanced resource utilization. Operational efficiency emphasizes the system's functional requirements, such as execution time and energy consumption, aligning with the main objectives of task scheduling. Given the nature of policy gradient optimization algorithms, which prioritize actions with higher rewards by assigning them greater selection probabilities, we yield higher rewards for operational efficiency based on the main objectives. Alongside optimizing the objectives, balanced resource utilization addresses the non-functional requirements by stabilizing workload distribution across the network, promoting sustainable system performance. Considering the objectives in the reward tends to favor a few high-performing devices that dramatically decrease the execution delay. This high improvement moderates energy consumption and leads to extreme decisions. Over-reliance on certain devices risks overloading them, and if a device becomes unavailable, the model may lack a suitable fallback. To address this, the reward function is adapted to account for utilization, preventing overuse of some devices and promoting balanced network distribution. The proposed reward function, considering these points, is as follows:

$$Reward = e^{(-1\times(\alpha.e+\beta.t_{eff}))} \times (1-(\lambda.u)) \tag{6}$$

where e, t_{eff}, α, and β represent the energy consumption, effective execution time, and their corresponding weights in augmentation, respectively. Since the two objectives are augmented, they are normalized in the range [0, 1]. The experimental justification of this relation and the appropriate values of the weights to balance the optimization process will be discussed in detail later in the experiment section. Moreover, u and λ represent the utilization and its coefficient, adjusting its influence on the reward. The effective execution time is defined as a deadline-aware adjustment of the execution time, given by:

$$t_{\text{eff}} = t \cdot \exp\left(\frac{t-d}{d}\right) \tag{7}$$

where t is the normalized execution delay and d is the normalized dead-

line. This exponential adjustment ensures that when an application is completed before its deadline (t < d), the resulting negative exponent reduces t_{eff}, thus increasing the reward. If the task is completed precisely on the deadline (t = d), the exponential term becomes one, and t_{eff} is equal to the original execution time, leaving the reward unaffected. When a task is delayed beyond its deadline (t > d), the exponent grows, increasing t_{eff} and consequently reducing the reward and leading to setting a variable to consider in punishment. Moreover, the value of λ is computed based on the diversity and Gini coefficient of the system as follows:

$$\lambda = \omega_v \times \lambda_{Max}.(1 - v) + \omega_g \times \lambda_{Max}.(g) \tag{8}$$

where v and g represent the system's diversity as the ratio of utilized devices to the total available devices and the Gini coefficient as workload distribution across the network, respectively. These parameters are described in the range [0,1]. Values closer to 1 in diversity (v) indicate greater variety in device usage and less balanced workload distribution for the Gini coefficient (g). These metrics are updated dynamically at runtime as the tasks are allocated based on the system's condition. Moreover, λ_{Max} is the threshold that defines the maximum possible impact of the Gini coefficient and diversity on the reward function, helping to ensure balanced utilization throughout the network. The value of this metric should be selected based on the importance of utilization for the deployment, balancing efficiency and resource distribution according to the system's requirements. Last, ω_v and ω_g are the adjusting weights in augmenting the diversity and Gini coefficient parameters. The Gini Coefficient is a statistical measure commonly used to quantify inequality or uneven distribution within a dataset. The Gini Coefficient has broader applications, including workload distribution in networks, where it can measure the balance of resource utilization across devices. A lower Gini Coefficient indicates a more equitable distribution, while a higher value reflects greater disparity. The Gini Coefficient is mathematically defined as follows:

$$g = 1 - \frac{\sum_{i=1}^{n}(n - i + 1).u_i}{\sum_{i=1}^{n} u_i} \tag{9}$$

where n shows the number of processing elements of the system and u_i represents the utilization of each processor.

This proposed reward function leads the model to more appropriate solutions that optimize the main objectives while balancing utilization. Due to the exponential and linear effects of the objectives and utilization in the reward function, the model initially prioritizes operational efficiency in terms of execution delay and energy by identifying fast-executing devices. Over time, the utilization of these devices increases, and the model is pushed to gradually reduce its reliance on them by exploring and adapting to other capable devices in the network that provide more execution delay.

As explained in the reward function, punishment is defined in our proposed DARTS to avoid unwanted decisions. This punishment helps the model understand the system's limitations and adapt to ongoing environmental changes. In this context, punishments for constant constraints and dynamic environmental changes are considered. The former focuses on fixed limitations, such as safety requirements and task compatibility. Safety is defined as a binary flag for each task and an attribute for each device. A safety violation occurs if a safe task is assigned to an unsafe device. This constraint guarantees the appropriateness of resource assignment for safety-critical tasks. As well, the task compatibility refers to the appropriateness of the allocated resource based on the task types. We define four task types while the underlying devices support specific task kinds. A compatibility violation occurs when the task type is not supported by the chosen device. A consistent and large penalty is applied to lead the model to consider these rules. We also define a strict punishment for violating hard deadlines, where any task that exceeds its specified deadline is treated as a failure. This encourages the model to maintain rigid adherence to non-negotiable timing requirements and discourages any lateness. This penalty (P_c) is computed as follows:

$$P_c = P_w \times (F_S + F_T + F_D) \tag{10}$$

where P_w is the constant punishment rate set to -100 times the average reward multiplied by three terms. The F_S, F_T, and F_D are the number of failed tasks due to safety, compatibility, and deadline miss. When reporting batch penalties, we sum or average these indicators over the batch to derive the number of failed tasks. Defaults for device safety flags and supported types of various IoT layers set from Table 2. In this context, by applying this significant fixed punishment, the model learns to avoid actions that violate constant constraints, reinforcing safe and compatible task assignments.

The latter penalty revolves around the dynamic system's condition, such

as battery levels on edge devices that vary over time. The penalty of this situation should increase gradually to prevent the model from reaching a critical status before its occurrence. For instance, considering a fixed penalty for battery level would bias the model against a device only when it reaches the critical level. It leads to an inefficient period where the model repeatedly relies on the device until its penalty reaches a severe penalty value. In this case, we adjust the dynamic penalty (P_d) based on the battery usage as follows:

$$P_d = -P_{init} \times ((b_{start} - b_{end})^{\gamma}) \times ((\frac{100 - b_{end}}{100})^{\zeta}) \tag{11}$$

where P_{init} is the base penalty scaling factor, b_{start} and b_{end} are the initial and final battery level percentage, γ adjusts the penalty based on battery drain rate and ζ represents the penalty's sensitivity to low battery levels. This dynamic penalty is added to the reward function of Eq. 6 as follows:

$$Reward_{Total} = Reward + P_d \tag{12}$$

As the battery level of a device decreases, this penalty incrementally increases the punishment it receives, with penalties intensifying as the device approaches its battery's state-of-charge (SOC). This encourages the model to gradually deprioritize devices with low battery levels and explore options, such as edge devices with higher battery levels or devices from other layers. Through this adaptive approach, the model becomes more resilient, learning to balance performance with the availability of resources dynamically.

- Illustrative Example:

To clarify the effect of the proposed reward and punishment functions, an illustrative example in four conditions is defined. Assume the moving-average reward of a sample execution at the current step is 1. As defined, in this case, the penalty scale is set to $|P_w| = 100 \times 1 = 100$. The total penalty for a decision is then multiplied P_W by the sum of three components: a deadline term (F_D), a safety indicator (F_S), and a type-compatibility indicator F_T. In this example, first consider a soft real-time application that has a 10s deadline and a 2s soft slack. In case of finishing at 11.2s, it has 1.2s of lateness. Thus, its soft lateness factor is $L = (t - d)/s = 0.6$. Moreover, suppose this task is safe and the selected device for its execution is unsafe ($F_S = 1$), while the task type is supported ($F_T = 0$). In this case, the execution penalty is $100 \times (0.6 + 1 + 0) = 160$. This example shows how multiple small violations of the objectives influence the learning.

As in the second case, consider a hard-deadline application that finishes 0.01s later than its assigned deadline. Since any tardiness violates a hard deadline, the F_D is set to 1. If the device is safe ($F_S = 0$) but does not support the task type ($F_T = 1$), the penalty becomes $100(1 + 0 + 1) = 200$. In this case, the full hard-deadline penalty combines with a type mismatch to produce an even stronger punishment than the previous example, which was later.

Third, consider a decision that violates both the safety and the task-type constraints ($F_S = 1$, $F_T = 1$). Since the task is not executed in this case, $F_D = 0$ and the penalty is $100 \times (0 + 1 + 1) = 200$; the battery-depletion term is not applied when constraints are violated. Fourth, if a task finishes on time on a safe, compatible device, we apply only the dynamic battery-depletion term of punishment as P_d from Eq. 11. Using the stated settings, a high state-of-charge (SOC) window 100 to 80 yields $P_d \approx -1.08$, while the same absolute drop at low SOC, 30 to 10, leads to $P_d \approx -98.4$. This case shows that identical energy loss is penalized lightly when the device ends at high charge but heavily when it ends near empty.

4.2.3. ***Learning Process of DARTS***

Our proposed DARTS employs the defined learning model on the constructed DDT to schedule the tasks' applications on edge devices, MEC, and the CC. Our employed learning process is multi-agent and scalable due to IoT systems' dynamic and ever-changing nature. These characteristics introduce challenges like instability and inconsistency in the system due to the distinct actions of agents. To consider this, we have employed the asynchronous advantage actor-critic (A3C) algorithm in our proposed scheduler [42, 43]. This algorithm addresses the mentioned challenges using multiple parallel agents that simultaneously explore different parts of the environment. For consistency, these agents update a shared and global model periodically. This parallelism accelerates the learning process and enriches the learning experience by exposing the model to a broader range of environmental scenarios. This approach is particularly effective for coordinating tasks in IoT systems, where many devices must collaborate on resource allocation and task scheduling.

Our choice to combine the A3C framework with the PPO algorithm provides a robust foundation for stable learning. The primary challenge in online reinforcement learning is that sequentially gathered experiences are highly correlated, which can destabilize training. Furthermore, in such a dynamic and multi-criteria environment, an agent can be prone to destructive policy

updates, where a single large change leads to a sudden, catastrophic drop in performance.

A3C directly mitigates the challenge of correlated experiences by using multiple parallel agents to provide a diverse range of experiences, which decorrelates the data used for updates. While A3C ensures stability through data diversity, reinforcement learning can still suffer from destructively large policy updates. PPO addresses this by using a clipped objective function, which restricts the magnitude of policy changes at each training step, preventing overly aggressive updates that could lead to performance collapses.

Our A3C-based task scheduler has a global agent, a global optimizer, and multiple distributed agents. Distributed agents are considered 32 units experimentally to support robust and adaptable learning. In this context, each independent agent has an actor for task allocation and a critic for evaluating the quality of these decisions. The actor utilizes the constructed DDT to navigate task scheduling, dynamically adjusting task allocations as network conditions evolve. Meanwhile, the critic is a neural network that assesses state values to inform and enhance the actor's decision-making.

Throughout the scheduling process, incoming applications are assigned to distributed agents to define appropriate execution resources and orders. After completion, it updates its parameters based on these accumulated experiences, calculates gradients, and synchronizes these updates with the global model. The global optimizer then integrates these updates, refining the shared model parameters that agents subsequently retrieve, and ensuring all agents work with the latest knowledge before the next task allocation cycle. This collaborative approach enhances the agents' functionality by utilizing collective experience across the network.

At the core of the learning process, we employ Proximal Policy Optimization (PPO) due to its robustness and stable convergence in complex environments. PPO facilitates a steady learning trajectory with primary actor and critic loss components. In this context, the actor loss function is structured as:

$$\begin{cases} P_{action} = P_{path} \times P_{selection} \\ Loss_{actor} = -log(P_{action}) \times Reward_{Total} \end{cases} . \tag{13}$$

where P_{action} represents the logarithmic probability of the chosen action and P_{path} and $P_{selection}$ are the probabilities of reaching the designated leaf node and selecting the device based on the probability distribution at that leaf

node, respectively. The reward modulates this loss term by jointly determining the direction and intensity of the adjustment. Standard rewards keep the loss term positive, promoting desired actions, while punitive rewards shift it strongly negative, discouraging the associated actions. For instance, in fig. 5 in the case where the leaf node addressed as "LR" is targeted, the P_{path} is calculated as 0.7×0.45 and the LR node's $P_{selection}$ is 0.52 that leads to P_{action} equals to 0.1638 as their multiplication.

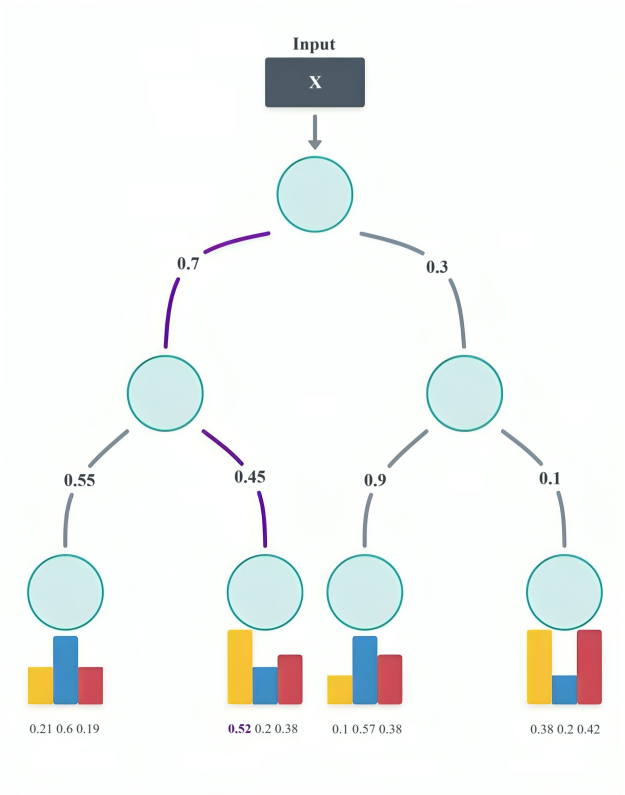


Figure 5: Illustrative Example of calculating the probability of the chosen action based on the path and selection probabilities for leaf node LR (left-right)

To optimize the Loss, we employ the Adam optimizer, which seeks to minimizes the magnitude of loss independent of its sign and focuses on convergence to zero. This approach encourages the model to avoid actions associated with penalties and low rewards while guiding it toward high-reward actions. In this gradient-based optimization, Adam updates the model parameters to reinforce actions yielding high rewards by effectively increasing

their probability in future decisions.

The critic network is modeled as a neural network with a single hidden layer of 256 neurons. It focuses on refining value estimation and calculations of the current state or task. In this context, we use Mean Squared Error (MSE) loss between the critic's predicted values and the actual returns stored in the agent's memory to achieve accurate value estimation. The learning process is stabilized by combining the actor and critic losses by a balancing coefficient as follows:

$$Loss_{Total} = Loss_{actor} + 0.5 \times Loss_{critic} \tag{14}$$

Afterward, the update of this total loss is regarded within a predefined range to stabilize the learning by the PPO clipping technique as follows:

$$Loss_{PPO} = min[r(\theta) \times Loss_{total}, clip(r(\theta), 1-\epsilon, 1+\epsilon) \times Loss_{total}] \tag{15}$$

where $r(\theta)$ and ϵ are the probability ratio between the new and old policies and the clipping parameter to limit changes to the policy, respectively.

Since devices frequently join and leave the network in IoT systems, scalability is a requirement. Thus, our proposed DARTS manages device exit and entry during task scheduling. This adaptability is addressed by manipulating probability distributions based on the changed situation. When a device leaves the network, its probability is redistributed among the remaining devices in a weighted manner, ensuring continuity in task allocation without the need for the model's retraining. This process employs a computationally trivial vector re-normalization at each leaf. When a device joins the network, it gets an initial probability based on its features, aligning it with the system's learned behavior. This initial probability depends on the device's functionality and dynamic features. Importantly, each leaf node's probability assignment is unique, as nodes are tailored to different task types and constraints. For instance, if one node primarily handles safe tasks and another is oriented towards general tasks, a device suited for safe tasks would receive a high probability in the safe node and a lower one in the general node. To adapt the explained probability assignment to the system's current behavior, each leaf node hosts a predictive regression model (either linear regression or a lightweight neural network) that is periodically trained. The regression model leverages periodic updates handled by an asynchronous background process that does not block the real-time scheduler. This design ensures high

Algorithm 2 Details of our proposed task scheduling approach (DARTS) for each upcoming application

Inputs: DAGs of application's tasks(G_{App}), Underlying Resources' characteristics and constraints.
Outputs: Optimal scheduling of Applications' tasks on the resources.
- Initialize a global agent with model parameters and global optimizer, consisting of:
Actor: Differentiable Decision Tree (DDT) for task routing
Critic: Neural network for state-value estimation
- Initialize N distributed agents with the global agent parameters
ready list ← tasks of G_{App} with no data dependency
sorted ready list ← sort (ready list, No. successors)
while *(sorted ready list* $\neq 0$*)* **do**
  for *(*$\tau \in$ *sorted ready list)* **do**
    State(τ) ← τ's features
    Action(State(τ)) ← Appropriate device for τ
    ▷ Guided by actor DDT's probability distribution
    if *(Action(State(*τ*))* $\implies$ *constraints)* **then**
      calculate $Reward_{Total}$ ▷ Eq.12
      else
        $Reward_{Total} = P_c$
        ▷ Punishment based on constant penalty Eq. 10
      end
    end
    Save experience tuple (State(τ), Action(τ), reward(τ))
  end
end
▷ Scheduler updates for each application
- Agent schedulers model update:
Calculate $Loss_{Total}$ (actor and critic losses) ▷ Eq. 14
Apply PPO clipping to $Loss_{Total}$ ▷ Eq. 15
Send gradients to the global optimizer
- Global scheduler model update:
updates global parameters with agent-provided gradients
Synchronize all agents

adaptability without introducing hidden retraining latency. This model takes the device specifications as inputs and outputs the device's probability of being chosen, effectively fitting the new device into the latest learned behavior of the system. The stepwise details of our proposed DARTS are summarized in Alg. 2.

4.3. ***Complexity Discussion and Practical Deployment***

Since our proposed DARTS is a dynamic scheduling scheme, its time complexity analysis is required. DARTS is based on a differentiable decision tree with depth "d" and has a single node that performs the sigmoid operation at each level. Thus, its time complexity is estimated as O(d) as its inference involves traversing a single path from root to leaf with constant-time operations at each level. Moreover, its memory complexity is $2^d * (N + C)$ where N is the input features and C represents the output classes. The exponential

scaling with depth enhances the model's capacity to capture complex decision boundaries but necessitates depth optimization to provide the method's efficiency.

To determine the computational and storage overhead of the proposed model on the resource-constrained edge devices, a practical adoption study is required. Since our proposed DARTS is based on standard DDT, in case of depth d, it has $2^d - 1$ internal nodes and 2^d leaves. Each internal node holds $N+1$ parameters (N feature weights plus a bias), and each leaf emits C logits over candidate devices. Total parameter count estimates as $(2^d - 1)(N + 1) + 2^d \times C$. For example with d=6, N=20 features, and C=50 candidates per leaf, the model has $(64 - 1) \times 21 + 64 \times 50 = 1323 + 3200 = 4523$ parameters. At 32-bit floats, that requires about 18 KB of memory; even with d=8 and larger C, the footprint stays in the tens to low hundreds of KB. Inference traverses a single root-to-leaf path, so as estimated earlier, the runtime overhead is proportional to tree depth and is well-suited to edge CPUs. Moreover, the communication overhead in the proposed DARTS is limited to intra-layer costs, and the inter-layer delays are not considered. It should be noted that our proposed method is flexible and could be compatible with various IoT protocols by adjusting its parameters.

5. Experimental Results

In this section, the efficiency of our proposed task scheduling scheme is evaluated and compared with related studies through several experiments.

5.1. ***Simulation Setup***

To evaluate DARTS, we develop a simulation framework that captures complex dynamics and constraints of real-world IoT systems. This simulator implements a three-layer hierarchical architecture, comprising IoT edge devices, MEC servers for intermediate processing, and a cloud server for handling heavy computational tasks. Each layer is characterized by distinct capabilities and constraints inspired by [17].

The IoT layer emulates devices such as the ESP32-S3, a dual-core microcontroller operating at 1.8–3.6V with clock speeds up to 240 MHz. Although real-world ESP32-based nodes typically operate with a single-digit number of cores, we simulate configurations with 4–16 logical cores to reflect the growing trend toward parallelized task handling in multi-sensor or clustered edge deployments and increase the scheduling complexity of the

problem space. The MEC layer models embedded servers like the NVIDIA Jetson AGX Orin, featuring scalable multi-core ARM Cortex-A78AE CPUs (up to 64 cores) and dynamic voltage-frequency scaling (0.8–1.2V, up to 2.0 GHz), representative of modern AI-enabled edge infrastructure. The cloud layer is modeled using high-performance processors such as the Intel Xeon Platinum 8268 (Cascade Lake), with 24 physical cores per processor and operational frequencies ranging from 2.8 GHz to 3.9 GHz under Turbo Boost. The simulation also incorporates task compatibility and safety constraints at runtime, reflecting real-world deployment challenges. These additions elevate the complexity of scheduling, ensuring a more realistic evaluation of DARTS. A summarized overview of the simulated IoT platform based on its characteristics is provided in Table 2.

Table 2: Characteristics of the simulated IoT platform considering device configurations in three layers

Characteristics	**Architecture Layers**		
	IoT device Layer	*MEC Layer*	*CC Layer*
Voltage	1.8-5V	0.8-1.2V	0.7-1.2V
Frequency	10-160 MHz	0.6-1.5 GHz	2.8-3.9 GHz
Number of CPU Cores	4-16	16-64	Unlimited
Safety Percentage	75%	50%	0%
Task Support Capacity	2 kinds	3 kinds	Unlimited

The applications are implemented as the DAGs of the tasks, each node has five attributes: computational load, data size, safety requirement, task type, and predecessor references. Our simulation framework generates applications of various sizes and shapes, categorized into small, medium, and large-scale graphs with 5 to 100 nodes. To provide inclusive experiments, more than 30000 task graphs of various sizes are generated with the normal distribution. Table 3 shows the statistical description and diversity of the DAGs used in the evaluation of the proposed method. Moreover, to have standard benchmarks, three types of task workflows, including trivial graph shapes, inspired by real-world workflows, and Pegasus Synthetic Workflow, are employed [44]. The characteristics of these graphs are set during the pre-processing phase, the same as the generated ones.

We employed the simulation framework to provide a robust and adaptable environment for evaluating task scheduling strategies in diverse IoT networks called "SchEdge" [45]. This simulator accurately models real-world,

Table 3: Statistical characteristics of the employed application graphs in experiments

Application Graph Scale	**Range**	**Number of Graphs**	**Mean±STD**
Small	5-20	10000	12.501±4.608
Medium	21-40	10000	30.081±6.097
Large	41-100	10000	69.876±17.637

lightweight heterogeneous devices, workload generation, complex task dependencies, and critical operational constraints such as resource compatibility and safety requirements. Implemented in Python and optimized for reinforcement learning-based scheduling, designed with a highly configurable architecture, ensuring adaptability to a wide range of scenarios [45]. Operating on an Intel Core i7 (9th/10th generation) with 8GB RAM, it maintains computational efficiency while ensuring reliable and accurate performance evaluation.

The learning process is built upon an Asynchronous Advantage Actor-Critic (A3C) framework, which in our setup utilizes 32 parallel agents to interact with the environment and gather diverse experiences. Each agent is equipped with a local version of the actor, which is the Differentiable Decision Tree (DDT) with a maximum depth of 3, and a critic, implemented as a neural network with a single hidden layer of 256 neurons for state-value estimation. These local agents send their gradient updates to a central global model. To ensure stable training and prevent destructive policy updates, the loss is calculated and optimized using the Proximal Policy Optimization (PPO) algorithm over 10 epochs per update cycle. Key hyperparameters for this process include a learning rate of 0.002, a PPO clipping epsilon of 0.2, and a discount factor of 0.99, which together create a robust and adaptable learning system suited for the dynamic nature of IoT environments. To ensure the reproducibility of the proposed method, its implementation and source codes are published in: "https://github.com/athena-abdi/Intelligent-Cyber-Physical-Systems-ICPS-/tree/main/Task_Scheduling_in_IoT/DARTS".

5.2. ***Simulation Results***

In this section, to evaluate the effectiveness of our proposed DARTS, four classes of experiments are performed. First, the parameters of the reward function in terms of its shape and coefficients are studied. Then, the capability of the proposed DARTS in multi-objective optimization, meeting the

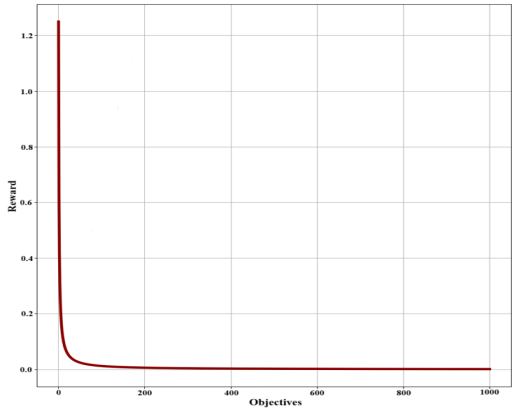
constraints, and scalability handling is investigated. Moreover, the generality of the proposed DARTS in handling various sizes of applications and its complexity analysis are studied. Lastly, the efficiency of the proposed DARTS is compared to related studies in terms of joint optimization of execution time and energy consumption, along with the response time.

5.2.1. ***Parameter Setting***

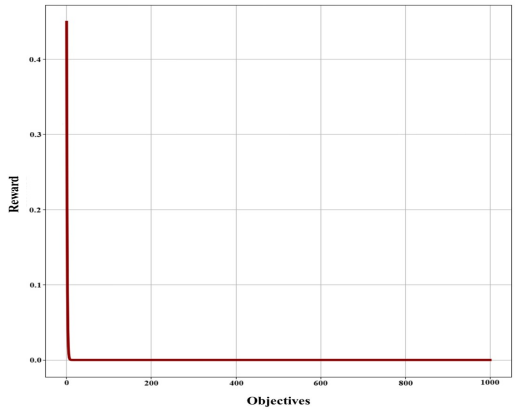
As explained, our proposed reward function of DARTS relates the parameters exponentially and with predefined weights. To evaluate the appropriateness of the shape and the considered weights of this function, several experiments are performed. The nature of the policy gradient algorithms requires the formulation of reward functions where decreasing energy consumption (e) and execution time (t) would result in higher rewards. In this context, various combinations of the objective parameters are studied. Among them, the inverse of linear, exponential, and logarithmic combinations of the objectives represent their effect on the reward function better. Fig. 6 shows the selected reward function under the simplified assumption of equality of energy and execution time. These plots depict how the reward evolves while execution time and energy range from 0 to 1000, highlighting distinct trends and responses of each relation. This provides a visual comparison to evaluate the suitability of various setups for the optimization process.

As this figure shows, the second setup (b) yielded the most favorable results. This choice is justified by its exponential nature, which amplifies the model's sensitivity to variations in energy and time. Unlike linear or logarithmic setups, the exponential combination of objectives is highly responsive to changes, ensuring that small reductions in energy or execution time result in significant reward increases. Furthermore, the exponential decay ensures that the α and β are more effective in the model's learning process. This allows us to balance energy and time more effectively, aligning the reward system closely with the desired behavior.

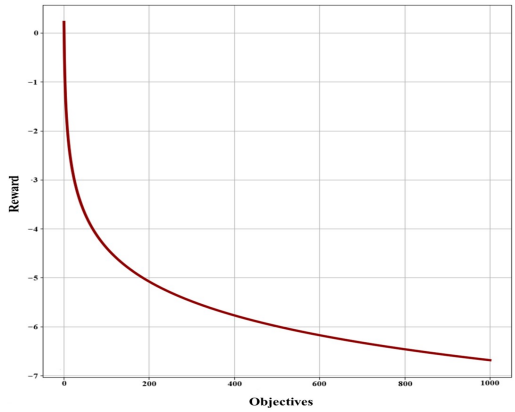
After determining the reward function's shape, the values of α and β should be regulated. These parameters control the relative importance of normalized execution time and energy consumption in the reward function. These coefficients are critical for adjusting the reward system to prioritize time and energy or balancing both. To determine these parameters, various combinations of execution time and energy consumption with low echelons are generated. These parameters are derived from simulated applications



(a) reward=$\dfrac{1}{\alpha.e + \beta.t}$



(b) reward=$e^{-(\alpha.e+\beta.t)}$



(c) reward=$-log(\alpha.e + \beta.t)$

Figure 6: Selected reward functions over energy and execution time variation

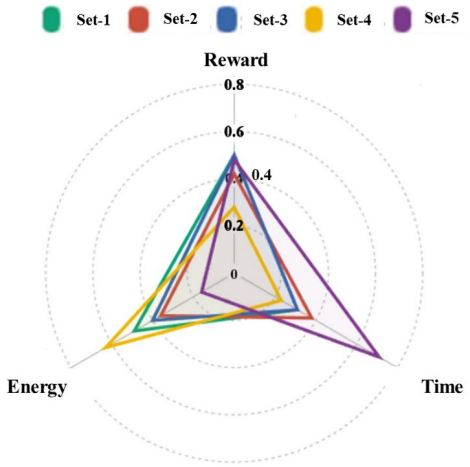
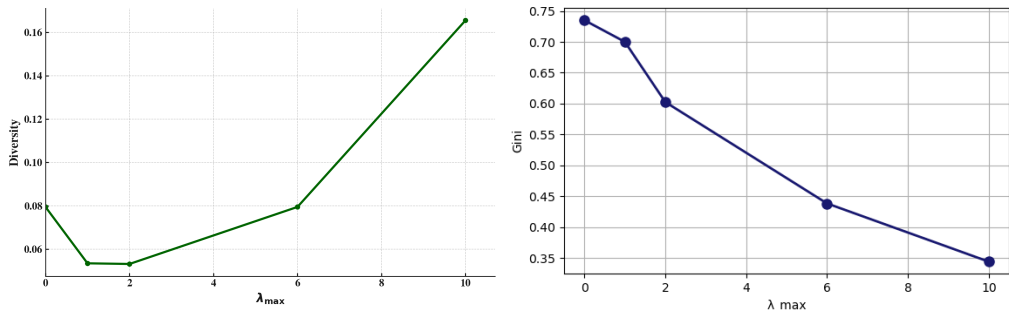


Figure 7: Effect of applying various weights of objective on the reward function, execution time, energy consumption

running on the IoT system, and their values are normalized in the range [0,1]. Afterward, α and β are set in the range [0.1 to 3.5] in 0.1 increments to generate 1225 distinct combinations. This range is chosen to avoid small rewards due to the exponential nature of the reward function. Various permutations of parameters based on the described setting are tested to determine the best coefficients for the objectives in the proposed reward function. The resulting reward values are clustered into five groups representing different behaviors. The average α and β values of each class are computed, and five sets are constructed as follows: set 1 = (0.9120,0.8590), set 2 = (1.75,2.5982), set 3 = (2.9625,2.6486), set 4 = (0.5926,2.62660), and set 5 = (2.6701,0.8955). These five pair sets are applied to the model in a long-term simulation, measuring execution time, energy consumption, and reward. The behavior of each cluster is summarized in Fig. 7.

As this figure shows, the first and third sets represent a nearly balanced performance due to the close values of weights. In these cases, the model tends to minimize energy by focusing on lowering time suited to prioritize efficiency through marginal reductions in time while maintaining low energy consumption. The second set represents a moderately balanced approach, where both time and energy are optimized, but with a slight emphasis on minimizing time. This selection is suitable for general-purpose networks where no specific priority is required. The fourth and fifth sets strongly favor



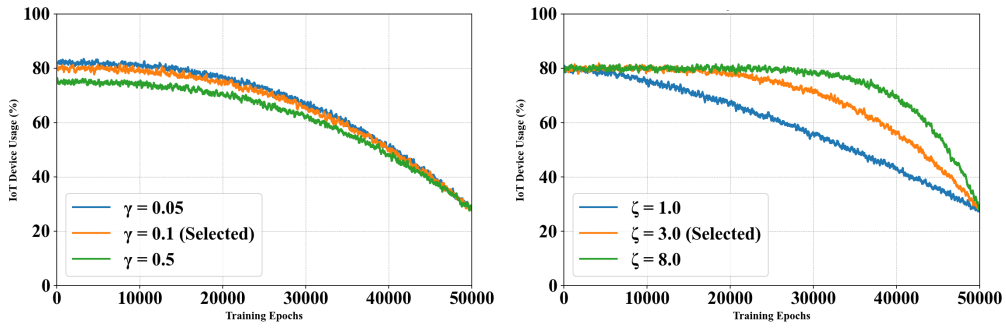
(a) (b)

Figure 8: The effect of λ_{max} on: (a) diversity and (b) Gini coefficient for different values in the range (0,10)

time and energy, respectively. These options fit the time-critical and energy-constrained systems where minimizing one of these objectives is paramount. Our proposed method is designed based on the second set of weights that better compromise the objectives.

Another empirically set parameters of the proposed reward function that aim to promote balanced resource utilization is ω_v, ω_g, and λ_{max} as explained in Eq. 8. Here, ω_v and ω_g are set to 1 in our simulation to favor them simultaneously. λ_{max} is the threshold that defines the maximum possible impact of the Gini coefficient and diversity on the reward function. Figure 8 illustrates the impact of λ_{max} on diversity (8-a) and Gini coefficient (8-b) across different values (0, 1, 2, 6, and 10). It is important to note that the Gini coefficient values are relational, meaning that only active devices—those that have been utilized—are considered in the calculations. Devices with zero usage are excluded to provide a more accurate representation of inequality among used devices. This adjustment is particularly relevant in large-scale networks such as ours, which consists of 151 devices. Given the short evaluation periods and the relatively small intervals over which the Gini coefficient is computed (100 applications per measurement), the metric would otherwise remain consistently high, failing to provide meaningful insights. By refining the calculation to focus only on utilized devices, we ensure that the Gini coefficient effectively captures the distribution of workload across active devices rather than being dominated by the sheer size of the network.

In addition to the reward function, our proposed dynamic penalty in Eq. 11 has two parameters γ and ζ based on the battery drain rate and the



(a) (b)

Figure 9: The effect of various values of dynamic penalty parameters on the device usage: (a) γ represents the drain rate (b) ζ represents sensitivity to low battery levels

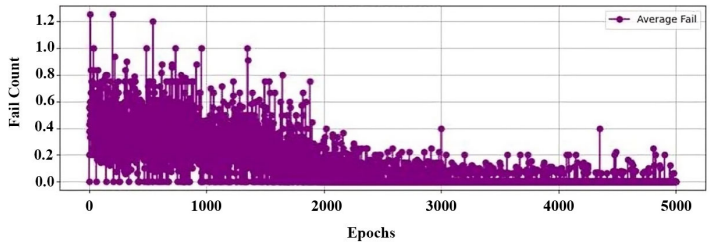
sensitivity of the dynamic penalty to low battery levels. The effect of these parameters on the device usage is analyzed in Fig. 9.

As this figure shows, increasing γ makes the agent more conservative with battery usage, leading to a lower overall allocation to IoT devices. While ζ controls how early the agent reacts to battery depletion. Thus, a lower ζ results in a gradual shift away from IoT devices, while a higher ζ causes a much sharper transition when battery levels become critical. During the simulation of DARTS, we selected $\gamma = 0.1$ and $\zeta = 3$ to ensure a balanced and proactive response to battery drain.

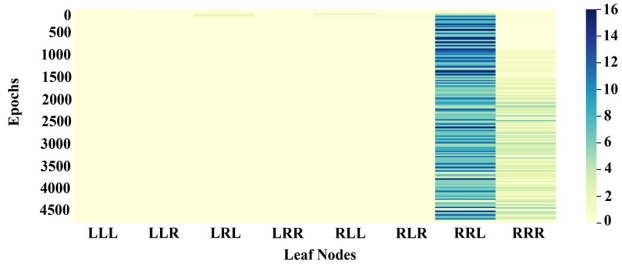
5.2.2. ***Effectiveness of the proposed DARTS in multi-objective optimization***

To evaluate the effectiveness of our proposed task scheduling scheme in exploring a suitable solution while optimizing performance and energy consumption, along with considering the system's utilization, several experiments are performed. First, the optimization process of the method in a single-agent configuration is evaluated. This simulation is configured with the same parameters as the main multi-agent model but operates at a reduced speed to provide a more granular examination of the optimization process. Additionally, to represent the dynamic nature of the IoT environment and evaluate the scalability of the proposed method, a probability of 0.001 at each epoch to either add or remove a device from the network is considered. This setup serves as a focused example of how the model optimizes objectives under fluctuating conditions.

Fig. 10 shows the model's understanding of the environment's fixed constraints through splitting. As illustrated in Fig. 10-a and Fig. 10-b, the first 2000 epochs highlight the model's initial struggle to resolve tasks under constant environmental constraints. The inability to handle conflicting constraints within a single probability distribution results in frequent failures. This is evident in the elevated average fail rate during this period. However, the maps also reveal a pivotal turning point: as the model begins to implement the splitting process, it significantly improves its ability to accommodate conflicting constraints. By refining routing decisions and creating specialized paths for different task types, the model reduces its failure average substantially. This outcome demonstrates the model's capacity to adaptively refine its decision-making structure, ensuring compatibility with fixed environmental constraints while minimizing failures.



(a)



(b)

Figure 10: Fixed environment's constraints optimization (a) failing history of the method in joint optimization (b) Path history map of the traversed leaf nodes (L and R represent the direction of traversal)

After the initial phase of learning and adapting to fixed constraints, the model focuses on optimizing task assignments to maximize rewards. It prior-

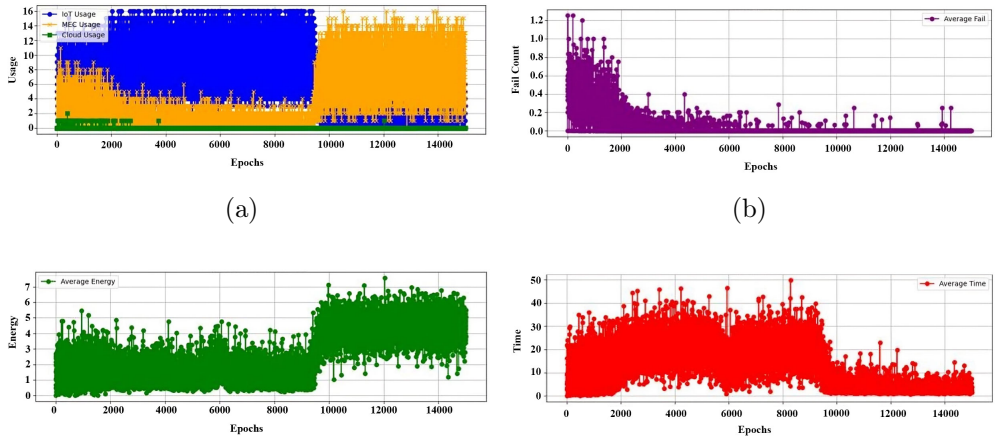


Figure 11: DARTS capability in handling environment changes during task scheduling (a) usage history of the resources (b) failing history of the method in joint optimization (c) Average Energy Usage (d) Average Execution Time

itizes edge devices over those in the CC and MEC layers, as the transmission costs associated with other layers make edge devices more efficient. The initial heavy reliance on edge devices leads to significant battery depletion over time. The designed adaptable battery punishment mechanism becomes instrumental here. As edge devices' battery levels decrease, the penalty activates the model to gradually reduce the usage of these devices, shifting task allocations to alternatives. This adaptive behavior is most evident around the 10000th epoch in our running example when high-performance edge devices are nearly depleted. The model responds by increasingly selecting MEC devices as substitutes. This transition is further reflected in the "Average Energy" and "Average Time" charts presented in Fig. 11.

As this figure shows, the usage of MEC resources increases due to the battery depletion of IoT devices, but this switching does not change the failing rate and optimization process (11-b). Switching to MEC devices increases average energy consumption due to higher transmission and operational costs, while reduced processing time is observed due to the superior computational capabilities of MEC devices (11-c & d). This trade-off demonstrates the model's capacity to adjust task scheduling to changing conditions while balancing performance and resource constraints. It should be noted that in this running example, we adjust battery depletion parameters to ac-

celerate this process and observe the model's adaptive responses in fewer epochs. Even after 10,000 epochs, the model maintains a low failure rate due to its probabilistic device selection strategy. Rather than always selecting the highest-probability device, it selects devices proportionally to their assigned probabilities. This prevents being greedy, encourages exploration, and helps it adapt to changing conditions. However, to reduce this, increasing punishment mechanisms or selecting the highest probability device during the test phase can be used for loss-intolerant networks. This pattern continues to appear in later figures.

After resolving the constraints and regulating the changes, the model stabilizes and balances resource distribution by the balanced resource utilization part of the reward function. This balancing coincides with the introduction of new high-performance edge devices, which, due to the probability regression models, receive higher probabilities in relevant leaf nodes. The model utilizes this opportunity to explore and identify options within the same range of standards and efficiency as its previous choices. The failing history diagram demonstrates that the model successfully identifies and incorporates these new devices while maintaining optimal scheduling. Fig. 12 shows the concurrent scalability management in conjunction with the model's tendency toward utilization and its cyclic adaptation to network dynamics. This process introduces calculated risks, as evidenced by fluctuations in both energy consumption and execution time until epoch 35000 of Fig. 12. These changes indicate the adaptive balancing mechanism at work, ensuring the model continues to meet its multi-objective goals while responding dynamically to scalability and utilization challenges. After epoch 35000 of Fig. 12. The events observed in the previous phase are repeated, this time with the model's attention shifting to the MEC layer as the usage of new edge devices decreases. This cyclical process reflects the model's ability to dynamically adapt its policy to the changing network environment, leveraging available resources to maintain efficiency while responding to device exits and entries. In this simulation, spanning 50,000 epochs, the environment underwent 47 changes—26 device removals and 21 device entries—demonstrating the dynamic nature of the IoT environment.

As this figure shows, higher utilization across the network introduces greater variability in both time and energy metrics. These metrics deviate further from their optimal values as the model balances the workload, a natural trade-off of ensuring broader resource distribution and sustainability within the network. This cycle of adaptation is expected to recur with on-

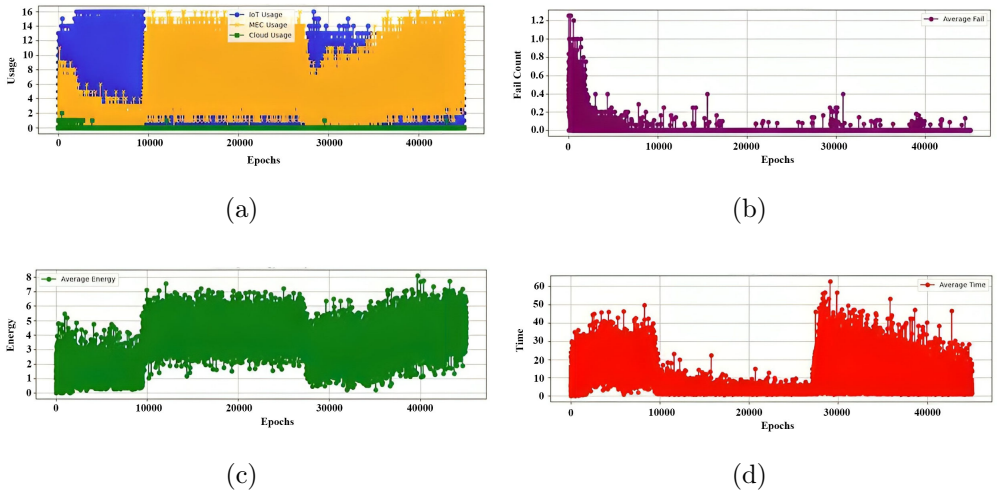


Figure 12: DARTS capability in meeting scalability and utilization during task scheduling (a) usage history of the resources (b) failing history in joint optimization (c) Average Energy Usage (d) Average Execution Time

going changes in the network, as the model continually refines its policy to optimize performance across available devices. While extreme and continuous changes can increase decision-making complexity and introduce ambiguity, the model demonstrates the ability to stabilize over time, maintaining a balance between efficiency and adaptability. This robust adaptive mechanism ensures the network operates effectively, even under unpredictable and evolving conditions.

As it has been explained, our proposed DARTS has load-balancing capability. Utilization is added as a prohibitive term to adjust the reward decision to optimize the execution time and energy consumption. The effect of utilization is regulated by a coefficient based on the diversity and Gini coefficient of the system called λ_{max} in Eq.8. By increasing this coefficient, the task allocation process tends to provide more distribution in the cost of unexpected splitting (shown in Path History Heatmap Fig. 13-a). However, this heightened focus on balancing resource distribution also introduces new challenges in the later stages of learning. This leads to fluctuations in the "Average Fail", "Time", and "Energy consumption" plots of Fig. 13-b to 13-d. To show the effect of this parameter and exaggerate the load-balancing term, we increase it from 6 to 10. In this case, due to the model's focus on utilization, it allocates tasks to nodes that have rarely or never been visited

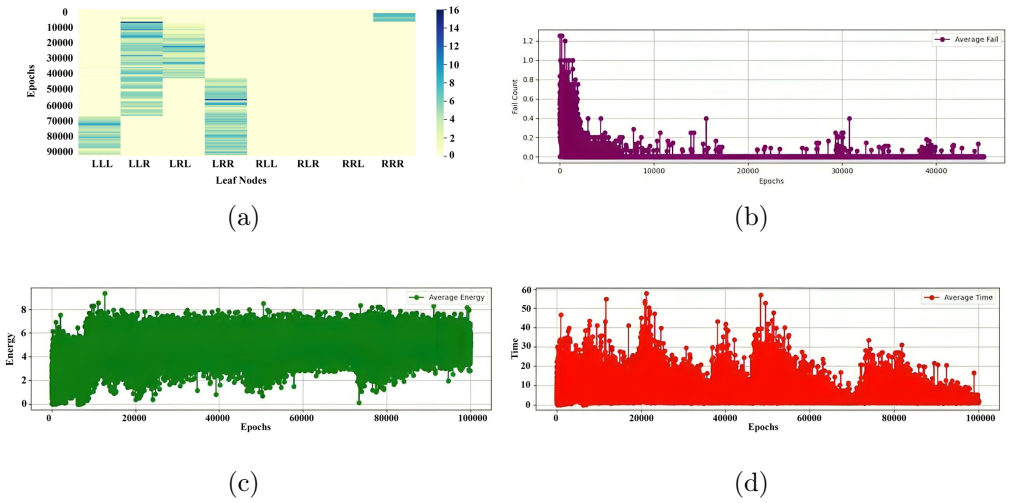
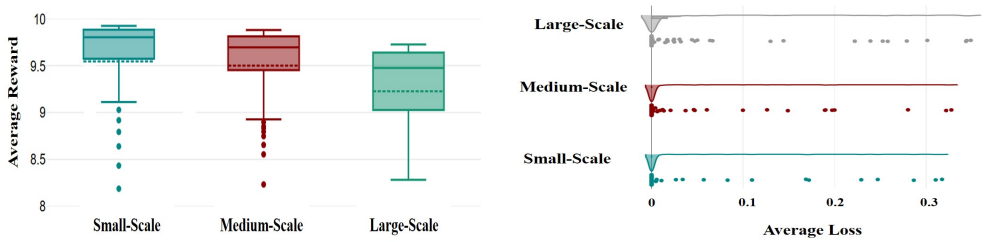


Figure 13: DARTS capability in load balancing during task scheduling (a) Path history in tree traverse (b) Average fail history (c) Average energy usage (d) Average execution time

during the primary learning process, such as those beyond explored paths of LLR and LRL. As these new nodes contain outdated or less-refined policies, the model initially struggles but quickly adapts. Exploring methods or phases could be introduced to address this challenge, ensuring that the model visits and updates underutilized nodes periodically during earlier learning stages. This would reduce the unfamiliarity with these nodes, minimizing the temporary inefficiencies caused by delayed exploration. This progression will allow tree learning to progress to a higher level, where each leaf node provides highly optimized answers. The network will achieve exceptional utilization as all paths are paved more equally.

As has been explained, our proposed DARTS is a multi-agent method capable of handling various sizes of applications. To show the generality of DARTS, more than 15000 various-sized application graphs ranging from 5 to 100 tasks are generated. Afterward, the efficiency of the proposed method in scheduling these applications is studied. The learning process is completed in these cases, and the optimized solution is derived. Fig. 14 shows the efficiency of the DARTS learning for various scales of application graphs in terms of the average accumulated reward and average loss.

As this figure shows, learning our proposed DARTS for various application sizes in terms of the average reward and average loss is mostly the same. This



(a) (b)

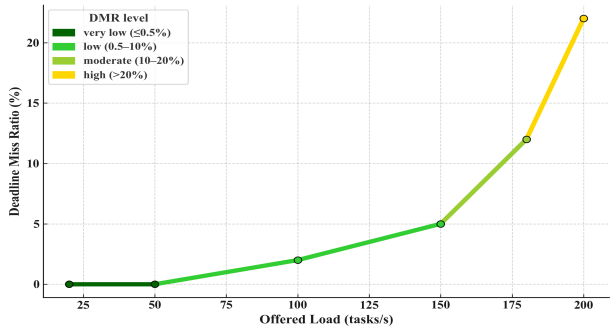
Figure 14: Efficiency of the learning process of DARTS in various sizes of application graphs considering (a) average reward, (b) average loss

shows the independence of the proposed method from the application's sizes and its capabilities in performing appropriate task scheduling. In order to compare the average losses, about 100 sample application graphs of various sizes are studied.

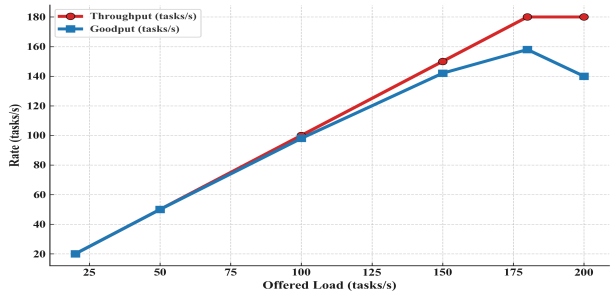
5.2.3. ***Evaluation of Quality of Service (QoS) Metrics in the Proposed DARTS***

To demonstrate the efficiency of the proposed DARTS, its deadline-sensitive behavior in QoS metrics is evaluated. To this aim, throughput, goodput, and deadline-miss ratio (DMR) are considered. These metrics determine the number of tasks completed per second, the number of tasks completed in time per second, and the ratio of tasks with missing deadlines to the ones that arrived, respectively. These parameters are evaluated by generating various loads that have 20, 50, 100, 150, 80, and 200 tasks per second. Fig. 15 shows the mentioned QoS parameters for different derived loads.

As this figure shows, DMR increases monotonically with load offered, as expected under deadline constraints. Moreover, DARTS attains the highest goodput up to saturation—about 140 tasks/s at 150—reflecting a few deadline violations. In our three-tier platform, the cloud server is modeled with virtually unbounded compute (unlimited cores), whereas MEC servers represent finite, near-edge capacity. Thus, the cloud provides ultimate processing capability, and the scheduler can offload spillover work to it to maintain throughput. This comes at the cost of additional backhaul latency and potential broker queuing, so the benefit appears primarily when the system is compute-limited rather than network-limited.



(a)



(b)

Figure 15: Efficiency of the proposed method in QoS parameters for various loads of network in terms of (a) Deadline Miss Ratio (b) Throughput and Goodput

5.2.4. ***Comparison to Related Methods***

To demonstrate the efficiency of our proposed DARTS, we compare it to related methods. In this context, relevant heuristic, meta-heuristic, and learning-based schemes are selected. The considered meta-heuristic methods are based on Particle Swarm Optimization (PSO) [31], Whale Optimization [46], and NSGA-II [32], to explore and identify optimal solutions. Moreover, a greedy search-based heuristic method [17], a deep reinforcement learning (DRL) approach inspired by [34], a deep deterministic policy gradient (DDPG)-based scheme inspired by [35], and a Q-learning scheme considered in [33] are also incorporated. To ensure the fairness of the comparisons and enhance the credibility of the results, the specific configurations of these methods are mentioned in Table 4.

To ensure a fair and robust comparison, all experiments were conducted on a unique infrastructure with an AMD Ryzen 5700 CPU and 16 GB of

Table 4: Implementation characteristics of the compared related methods (T_m: Execution time Coefficient, E_m: Energy Consumption Coefficient, It: Iteration Number, Pop: Population Size, μ: Mutation Rate, α: Learning Rate, M: Memory Size, B: Batch Size, γ: Discount Factor)

Method	**Type**	**Parameters**
TEMS [17]	Heuristic	T_m: 1.2, E_m:2.3
WOA [46]	Meta-heuristic	It: 100, No. Whales: 30
PSO [31]	Meta-heuristic	It: 100, No. Particles: 30
NSGA-II [32]	Meta-heuristic	It: 100, Pop: 30, μ: 0.6
DRL [34]	Learning	α: 0.0009 γ: 0.9, M: 1000 B: 64
DDPG [35]	Learning	α: 0.001 γ: 0.99, M: 1000 B: 64
DQN [33]	Learning	α: 0.001 γ: 0.99, M: 1000 B: 64 ϵ: 1, ϵ_{min}: 0.01, ϵ_{decay}: 0.999

RAM. Moreover, specific applications in the form of task graphs are employed to guarantee that the performance differences are attributable to the algorithms themselves and not variations in the input data. The comparisons are performed on 50000 applications of various sizes, and the assumed IoT platform consists of 100 edge devices, 50 MEC, and a cloud server. It should be noted that the compared methods do not consider constraints such as diverse task types, the differentiation between safe and unsafe tasks, battery depletion, and utilization, such as our proposed DARTS. Thus, these characteristics are omitted during the comparison, and the capability of methods in optimization of execution time and energy consumption as the main objectives are studied. Fig. 16 shows the comparison result in terms of derived execution time, energy consumption, and average response time.

Based on this comparison, our proposed DARTS outperforms other related methods in terms of compromising the objectives. The solutions of other methods generally tend to optimize energy consumption, but fail to balance the trade-off between these objectives. However, the proposed DARTS optimized both objectives more efficiently. Moreover, due to the real-time characteristics of IoT applications, the response time of the scheduling mechanism should be as short as possible. DARTS enhances the response time by about 40.87% compared to related learning-based schemes. This acheivement motivates the proposed method's suitability for real-time IoT scheduling. To clarify the performance gain of our proposed method, these comparisons are repeated ten times, and the statistical variance of each parameter in terms of

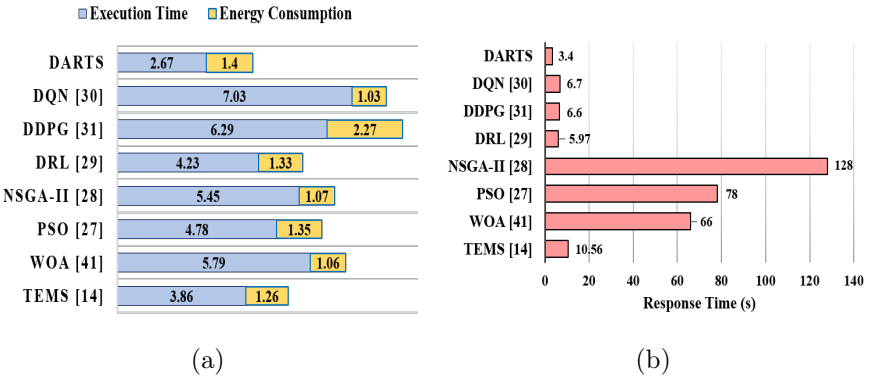


Figure 16: Comparison of DARTS to related methods in terms of (a) optimized execution time and energy consumption, (b) average response time

its mean and standard deviation is reported. Moreover, the results for various application graphs derived synthetically from SchEdge simulator [45] or the standard ones derived from [44] are presented separately. Table 5 shows the details of the comparison results.

Based on this table, our proposed DARTS outperforms learning, meta-heuristic, and heuristic-based schemes in synthetically generated application graphs by approximately 43.64% in joint optimization of execution time and energy consumption, respectively. In the case of application graphs derived from [44], the values are lower due to the smaller scale of the DAGs. Here, our proposed DARTS surpasses related studies by about 26.83% in execution time and energy consumption, jointly. In the mentioned improvement analysis, the weights of execution time and energy consumption are considered equal during the joint optimization. Moreover, since the considered heuristic method [17] presents a determined solution, its variation over multiple runs is zero. In addition, the QoS of the proposed method is compared to selected related studies. Fig. 17 shows the results of this comparison for DMR, throughput, and goodput metrics. It should be noted that this comparison is performed for various system loads ranging from 25 to 200 tasks per second.

As this figure shows, methods with larger response time (Fig. 16-b) demonstrate higher DMR at the same load. For instance, at the load point of 150 tasks/s, both DARTS (5%) and DRL (8.7%) remain single-digit DMR, while DDPG is low-teens (12%) and DQN is higher at about 19%. In this case, meta-heuristic methods miss deadlines more frequently as queues build un-

Table 5: Detailed comparison results of DARTS to related studies considering the statistical variance and different datasets ($\mathcal{T}$ and $\mathcal{E}$ represent execution time and energy consumption)

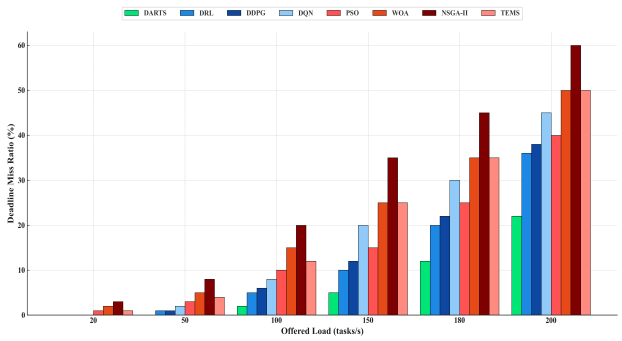
Method	**Synthetic DAGs**		**Standard DAGs [44]**	
	$\mathcal{T}$	$\mathcal{E}$	$\mathcal{T}$	$\mathcal{E}$
DARTS	2.67±0.21	1.40±0.11	2.23±0.16	0.8±0.06
DRL	4.23±0.42	1.33±0.13	3.19±0.26	0.75±0.06
DDPG	6.29±0.63	2.27±0.23	2.88±0.26	1.16±0.1
DQN	7.03±0.7	1.03±0.1	3.64±0.29	0.69±0.06
PSO	4.78±0.24	1.35±0.07	3.02±0.12	1.07±0.04
WOA	5.79±0.79	1.06±0.35	3.50±0.14	0.74±0.12
NSGA-II	5.45±1.33	1.07±0.56	4.01±1.2	0.63±0.23
TEMS	3.86 ±0	1.26±0	3.11±0	0.61±0

Table 6: QoS of DARTS and compared scheduling schemes at load 150 tasks/s in terms of deadline-miss ratio, throughput, and goodput (mean ± std)

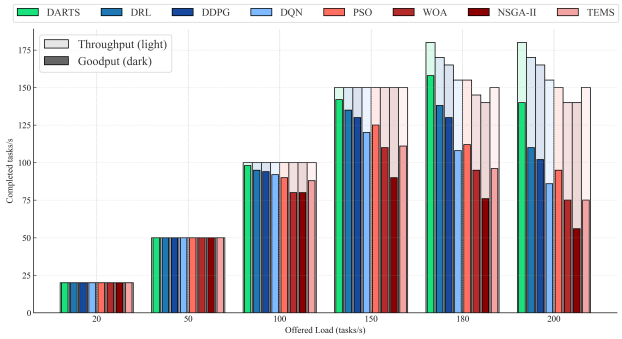
Method	**DMR (%)**	**Throughput (tasks/s)**	**Goodput (tasks/s)**
DARTS	5.04 ± 0.60	150	142.04 ± 0.50
DRL	8.68 ± 1.41	150	134.86 ± 3.93
DDPG	12.06 ± 1.57	150	130.13 ± 2.73
DQN	19.19 ± 1.59	150	119.42 ± 1.73
PSO	14.87 ± 1.29	150	125.36 ± 3.43
WOA	23.71 ± 2.04	145	110.26 ± 3.66
NSGA-II	35.23 ± 2.00	140	90.34 ± 3.51
TEMS	24.74 ± 1.89	150	111.34 ± 3.95

der volatility. Moreover, our proposed DARTS completes all tasks and meets their deadline, leading to the highest throughput and goodput compared to related studies in various loads. As well, in throughput and goodput, DARTS has the best performance. Focused on one load of 150 tasks/s, Table 6 shows the average values and statistical standard deviation for DMR and goodput of the compared scheduling schemes.

As it has been mentioned, our proposed DARTS can adapt itself to the dynamic and changing environments. Since the real-world IoT systems are volatile due to continual device entry and exit, this capability makes the proposed DARTS more appropriate than previous studies that are mainly static and decide based on the initial configuration of the system, regardless



(a)



(b)

Figure 17: QoS comparison of DARTS to related studies in terms of (a) Deadline Miss Ratio, (b) Throughput, and Goodput

of its further variations. In this context, we compare DARTS with related studies in a baseline scenario under various device churn rates of 5%, 15%, and 30% over 10,000 epochs on our assumed three-tier platform and under the described workload pattern with 150 tasks per second. These churn events alternate; every leave is followed by a join and vice versa, so the average pool size remains stable while the configuration changes frequently. For each churn level, we analyze 100 epoch windows starting immediately after each event. To show the scalability and dynamic adaptability of DARTS, we first compare it with DRL and NSGA-II in a baseline scenario under device churn. Fig. 18 reports the resulting deadline hit ratio for all three methods at different churn levels. NSGA-II computes a single mapping before the run and has no native support for reacting when devices leave or join. The DRL baseline likewise has no built-in feature for handling device churn and simply

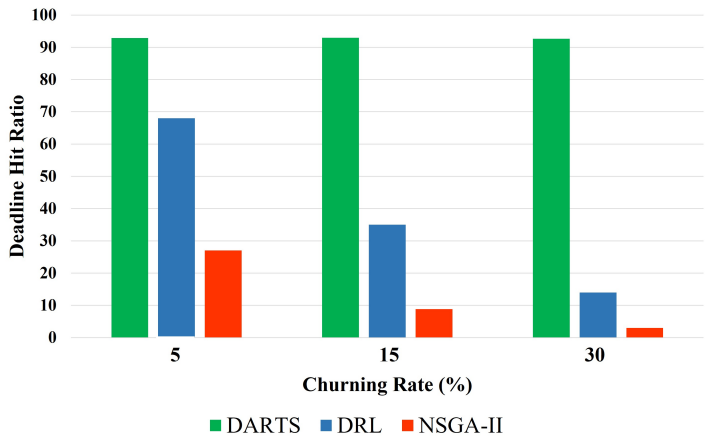


Figure 18: Comparison of DARTS to Static DRL and NSGA-II in terms of Deadline Hit Ratio

applies the policy that was learned for the initial configuration. DARTS, in contrast, is designed to handle device entry and exit through probability renormalization and regression-based priors for joining devices and continues to adapt during the run. As churn increases, the deadline miss ratio of DRL and NSGA-II rises sharply, while DARTS remains in the single-digit range. This experiment, therefore, measures the pure behaviour of each method under device churn and shows how vulnerable the traditional schedulers are when they operate without native churn handling.

As this figure shows, DRL provides a prominent deadline hit ratio in various device churning rates due to its adaptability to dynamic device environments. NSGA-II computes a single mapping before the run and has no native support for reacting when devices leave or join. The DRL baseline likewise has no built-in feature for handling device churn and simply applies the policy that was learned for the initial configuration. DARTS, in contrast, is designed to handle device entry and exit through probability renormalization and regression-based priors for joining devices and continues to adapt during the run. As churn increases, the deadline hit ratio of DRL and NSGA-II rises sharply, while DARTS remains stable. Afterward, we reuse the same platform, workload, and churn process as in Fig. 18 and introduce minimal, architecture-consistent modifications to DRL and NSGA-II so that they can react to joins and leaves, with all metrics averaged across

the corresponding post-event windows. The methods respond to churn according to their architectural constraints. DARTS encodes only tasks in its state and action space, with no explicit device modeling. When a device departs, it re-normalizes the leaf probabilities; when a new device appears, it seeds logits using a per-leaf regression prior. This allows immediate usability through a single tree traversal followed by lightweight validation. DRL is constrained by a fixed decision head and cannot dynamically add or remove actions. Under the alternation protocol, the action corresponding to the most recently departed device is reused for the next joining device, and the new device inherits its logit. In this way, DRL can use newly arrived devices, but it must relearn their quality and cannot redistribute probability mass as flexibly as DARTS. NSGA-II, as an offline optimizer, lacks native support for live churn. To adapt it for this scenario, we simulate an online variant that re-solves the entire placement after each churn event before deploying the updated mapping. This approach maintains feasibility and near-baseline steady-state cost but introduces substantial solve-time latency per event. We compare these churn-aware variants against DARTS and extend the analysis to reallocation latency, time, and energy overheads, and DMR as shown in Table 7. Reallocation latency is the time from detecting a churn event to the first valid placement that respects compatibility and safety constraints for the next affected task, covering queueing, policy inference, validation, and any necessary resampling. Time and energy overheads are expressed relative to the matched zero percent windows, and DMR represents the ratio of missed deadlines over various applications on average. To respect fairness across simulations, all random choices are held identical using a fixed seed, the same devices are removed and added in corresponding runs, and joining devices follow the described specifications.

As this table shows, our proposed DARTS is adaptable with varying device churn rates with limited reallocation overhead. The adaptability of NSGA-II and DRL methods is limited at a high cost of time. DARTS reallocates within seconds and shows little sensitivity to churn, since a single inference together with queueing and validation is enough. DRL reacts more slowly, and its time and energy overheads still grow with churn. Reusing the slot of a departed device lets it tap new devices, but its adaptation remains slower and less flexible than DARTS, and as the best edge options drain, it increasingly shifts work to the MEC and Cloud tiers. NSGA-II has a large and nearly constant per-event latency because it resolves the mapping before any placement. In the post-event windows, the overhead under DARTS re-

Table 7: Adaptability of DARTS in various device churn rates compared to related scheduling schemes

Method	Churning Rate	Reallocation Latency(s)	Energy Overhead(%)	Time Overhead(%)	DMR(%)
DARTS	5%	3.50	0.50	3	7.11
	15%	3.70	1.70	6	7.05
	30%	4.10	3	10	7.34
NSGA-II	5%	128.75	0.80	1.10	64
	15%	126.22	1.70	1.90	73
	30%	129.35	2.60	3.10	86
DRL	5%	5.90	7	2	10
	15%	6.80	24	9	17
	30%	9.50	42	28	27

mains minimal. Fresh edge capacity improves execution time early in the run, but battery depletion later shifts load upward and cancels much of the gain. DRL shows clearly increasing overheads as churn rises, while NSGA-II maintains a small steady state time and energy overhead once the re-optimization finishes. Quality of service follows the same ordering. DARTS keeps the deadline miss ratio low and can even improve at the highest churn, but the battery-driven shift to higher layers limits the full benefit of this effect. DRL's miss rate rises with churn because it cannot assign well-informed logits to newly joined devices and instead has to relearn their usefulness from scratch after slot reuse, which makes its adaptation slow and error-prone under frequent joins and leaves. NSGA-II ends up highest because deadlines expire while the optimizer is running.

5.2.5. ***Ablation Study***

5.2.5.1. Analysis the DART's Stability:

To empirically validate the stability of the learning model of DARTS, we conducted an ablation study comparing our combined A3C+PPO approach against models using only PPO, only A3C, and a standard single-agent REINFORCE baseline. Table 8 shows the results of this comparison for the mentioned cases in terms of convergence result, number of training epochs, and final average reward.

As this table shows, the single-agent REINFORCE baseline exhibited high variance and failed to converge within 10,000 epochs, settling at a final average reward of -2.33. While adding PPO in a single-agent setup stabilized the learning process, it was limited by correlated experiences, converging slowly at epoch 8,900 to a suboptimal reward of -0.33. Conversely, the A3C-only model demonstrated rapid learning due to decorrelated experi-

ences, converging by epoch 4,200 to a reward of -0.067; however, it was highly unstable and suffered from severe policy collapses. Our proposed A3C+PPO model balances out these approaches, achieving the fastest convergence at approximately epoch 1400 while maintaining a stable, high positive reward without any policy collapses. These results empirically confirm that both A3C and PPO components are essential for effective learning that are employed in our proposed DARTS.

Table 8: Stability analysis of various learning setups compared to DARTS

Learning Setup	**#Training epochs**	**Average Reward**	**Convergence**
Single-agent REINFORCE	10000	-2.33	Failed
Single-agent PPO	8900	-0.33	Done (slow)
A3C	4200	-0.067	Done (unstable)
A3C + PPO (DARTS)	1400	0.04	Done (fast and stable)

5.2.5.1. Analysis the DART's DDT Depth

To empirically validate the appropriateness of the DARTS's DDT depth, we have evaluated it with various depths of DDT. As it has been explained, the depth of the differentiable decision tree in DARTS determines the number of hierarchical binary partitions that can be expressed based on the task and device feature space. In this context, the minimum useful depth corresponds to the number of independent conflicting decision axes that must be separated during routing. In this experiment, have varied the DDT depth of DARTS from 2 to 8 and evaluate its performance in terms of executon time and energy consumption of the derived solution, inference time, FLOPS per decision, number of parameters, and number of convergence epochs. Table 9 presents the results of this evaluation.

As this table shows, in DARTS, the useful depth corresponds to the number of independent conflicting decision axes that must be separated during routing is three. In depth 2, the model is not able to find the appropriate solution considering the joint optimization of target objectives due to its limited learning capability. However, in low-utilization scenarios, this depth is also appropriate for our designed DARTS. In case of inadequate larger depth, increase expressiveness marginally at the cost of higher parameter count and inference time. This deepness leads to limiting the generalization capability of the model while increasing its overfitting probability, as in this

Table 9: Analysis of DARTS with various depths of DDT

DDT Depth	Execution Time	Energy Consumption	Inference Time(μs)	FLOPS/ decision	#Params	#Convergence Epochs
2	5.97	2.1	0.0167	334	631	3800
3	4.2	4.3	0.0250	501	1271	3900
4	4.6	4.1	0.0334	668	2551	4200
6	9.1	1.6	0.0501	1002	10231	5100
8	7.5	1.9	0.0668	1336	40951	5900

experiment, the deep models are incapable of optimizing execution time and energy consumption simultaneously. Based on this experiment, the most suitable depth of DDT for DARTS is 3, as it provides appropriate optimization cost in terms of execution time and energy consumption of the derived solution. Moreover, the final solution is resolved in an appropriate inference time and with a lower number of parameters and operations and making it a good candidate for real-time and resource-constrained applications. As a data-driven recommendation, the minimum useful depth corresponds to the number of independent conflicting decision axes that must be separated during routing. With sufficiently expressive node weights—i.e., adequate width per decision node—each split can isolate one non-linear or categorical conflict without requiring unnecessary additional levels. In DARTS, the dominant source of conflict arises from task-type granularity and device-level support constraints, alongside additional routing factors such as deadline slack and safety mode that contribute to conflicting groups. Consequently, selecting depths of two or three for it already provides enough structural capacity for typical task graphs. However, depth 3 shows better performance in high device churn scenarios. It should be noted that this experiment is performed on several application graphs of various sizes.

6. Conclusion and Future Work

In this paper, an explainable and multi-agent task scheduling approach through decision tree reinforcement learning (DARTS) is presented. DARTS aims to optimize execution time and energy consumption as the main objectives of IoT systems. As well, it meets environmental constraints such as task compatibility and safety requirements, along with providing load balancing among processing elements. Moreover, DARTS provides dynamic

device assignment capability and scalability of the system through probabilistic reasoning and modular updates in the designed differentiable decision tree scheduling approach. In this context, DARTS employs proximal policy optimization (PPO) within an asynchronous advantage actor-critic (A3C) framework in the lightweight structure of a differential decision tree to ensure stable convergence and efficient distributed learning. The main objectives and constraints are reflected in the proposed reward and punishment functions, and their relations are tuned experimentally.

The interpretable and lightweight structure of DARTS, based on differentiable decision trees, enables transparent decision-making and efficient integration into IoT systems. Moreover, it provides real-time solutions that operate efficiently under dynamic and evolving conditions to address the inherent challenges of task scheduling in IoT systems. The model's ability to handle conflicting constraints through adaptive splitting and its capacity for scalability, demonstrated by the seamless addition and removal of devices, highlight its robustness. Through the designed reward system, efficient punishment mechanisms, and parameter tuning process, we ensured that the model aligns with diverse network requirements, prioritizing efficiency while accommodating user-specific preferences. The punishment system proved particularly effective in helping the model internalize the constant environmental rules as well as adapting to dynamic changes like battery depletion. This allowed the model to make informed decisions while maintaining compliance with operational constraints.

Simulation results validated the effectiveness of our proposed DARTS, showing its ability to achieve multi-objective optimization and adapt to ongoing changes, including scenarios where nearly 32% of the devices were replaced or removed and 24% of devices experienced battery depletion. Despite these dynamic changes, the model maintained momentum and adapted its policy in a direction that aligned with the defined objectives, ensuring it stayed within the boundaries of environmental constraints while optimizing performance. Furthermore, it has been observed that this approach can seamlessly handle various application sizes, ranging from 5 to 100, without encountering any issues. The modular design of the reward system, with its flexibility to prioritize energy, time, utilization, or a balanced approach, further strengthens its adaptability across various IoT scenarios. This includes applications ranging from energy-critical networks to time-sensitive and utilization-focused deployments. The iterative improvement of task scheduling policies, driven by a carefully constructed reward function and

regular updates to global knowledge, underscores the strength of this reinforcement learning-based approach.

Compared to related studies, our proposed method outperforms heuristic and metaheuristic algorithms by about 31% in terms of joint optimization of execution time and energy consumption. Furthermore, compared to learning-based methods, our approach achieves about 47% improvement. Moreover, it outperforms DRL methods with its significantly lower computational complexity, making it an ideal choice for dynamic and real-time IoT environments. In terms of inference time, our proposed DARTS outperforms the DRL-based scheme by about 40.87%. Moreover, its improved response time over metaheuristic and heuristic is prominent due to their search-based policy. Although DARTS demonstrates the appropriate combination of performance efficiency and computational simplicity, there are still limitations in it that bound its current scope. First, in the proposed DARTS, a closed-form rule for choosing the DDT depth is not provided. The depth is selected empirically as a function of objective complexity and the size/heterogeneity of the device action space. Moreover, achieving stable policies under churn typically requires more training epochs than some DRL baselines, even though inference is lightweight. Also, the proposed method does not capture all device-level phenomena, such as thermal throttling, memory pressure, long-term device aging, and performance drift. Last, while edge–MEC–cloud communication is modeled, the protocol/broker effects, such as message queing telemetry transpoty/constrained application protocol (MQTT/CoAP), QoS retransmissions, or bursty path delays in a broker-in-the-loop setup, are not evaluated.

As future trends, these directions are proposed:

- Depth selection via data-driven growth/pruning or constrained search that ties depth to device heterogeneity and objective granularity;
- Hierarchical extension that augments device selection with core/DVFS decisions supporting;
- Enhancing system realism by incorporating thermal and memory constraints, explicit device-aging processes, and broker-in-the-loop experiments to quantify protocol-level delay and throughput under load;
- Exploring the forest-based variants and various types of DDTs, such as soft and clustrees, to increase policy capacity and robustness while preserving interpretability and edge-friendly inference.

References

[1] L. Fotia, F. Delicato, G. Fortino, Trust in edge-based internet of things architectures: state of the art and research challenges, ACM Computing Surveys 55 (9) (2023) 1–34.

[2] A. Sadeghi-Niaraki, Internet of thing (iot) review of review: Bibliometric overview since its foundation, Future Generation Computer Systems 143 (2023) 361–377.

[3] K. Pramilarani, P. V. Kumari, Cost based random forest classifier for intrusion detection system in internet of things, Applied Soft Computing 151 (2024) 111125.

[4] E. Al-Masri, A. Souri, H. Mohamed, W. Yang, J. Olmsted, O. Kotevska, Energy-efficient cooperative resource allocation and task scheduling for internet of things environments, Internet of Things 23 (2023) 100832.

[5] A. Bandopadhyay, V. Mishra, S. Swain, K. Chatterjee, S. Dey, S. Mallik, A. Al-Rasheed, M. Abbas, B. O. Soufiene, Edgematch: A smart approach for scheduling iot-edge tasks with multiple criteria using game theory, IEEE Access (2024).

[6] Q. Lu, G. Li, H. Ye, Edgesim++: A realistic, versatile, and easily customizable edge computing simulator, IEEE Internet of Things Journal (2024).

[7] T. Bu, Z. Huang, K. Zhang, Y. Wang, H. Song, J. Zhou, Z. Ren, S. Liu, Task scheduling in the internet of things: challenges, solutions, and future trends, Cluster Computing 27 (2024) 1017–1046.

[8] V. Patsias, P. Amanatidis, D. Karampatzakis, T. Lagkas, K. Michalakopoulou, A. Nikitas, Task allocation methods and optimization techniques in edge computing: A systematic review of the literature, Future Internet 15 (2023) 254.

[9] M. Karami, A. Abdi, H. R. Zarandi, A cross-layer aging-aware task scheduling approach for multiprocessor embedded systems, Microelectronics Reliability 85 (2018) 190–197.

[10] A. Abdi, A. Salimi-badr, Dyuns: Dynamic and uncertainty-aware task scheduling for multiprocessor embedded systems, Sustainable Computing: Informatics and Systems 43 (2024) 101009.

[11] M. R. Raju, S. K. Mothku, Delay and energy aware task scheduling mechanism for fog-enabled iot applications: A reinforcement learning approach, Computer Networks 224 (2023) 109603.

[12] A. Alam, P. Shah, R. Trestian, K. Ali, G. Mapp, Energy efficiency optimisation of joint computational task offloading and resource allocation using particle swarm optimisation approach in vehicular edge networks, Sensors 24 (2024) 3001.

[13] A. Abdi, A. Salimi-Badr, Enf-s: An evolutionary-neuro-fuzzy multi-objective task scheduler for heterogeneous multi-core processors, IEEE Transactions on Sustainable Computing 8 (3) (2023) 479–491.

[14] Z. Wang, M. Goudarzi, M. Gong, R. Buyya, Deep reinforcement learning-based scheduling for optimizing system load and response time in edge and fog computing environments, Future Generation Computer Systems 152 (2024) 55–69.

[15] N. Khaledian, M. Voelp, S. Azizi, M. H. Shirvani, Ai-based & heuristic workflow scheduling in cloud and fog computing: a systematic review, Cluster Computing (2024) 1–34.

[16] Z. Lin, C. Li, L. Tian, B. Zhang, A scheduling algorithm based on reinforcement learning for heterogeneous environments, Applied Soft Computing 130 (2022) 109707.

[17] J. C. Dos Anjos, J. L. Gross, K. J. Matteussi, G. V. González, V. R. Leithardt, C. F. Geyer, An algorithm to minimize energy consumption and elapsed time for iot workloads in a hybrid architecture, alamors 21 (2021) 2914.

[18] W. Zhang, Z. Zhang, S. Zeadally, H.-C. Chao, V. C. Leung, Energy-efficient workload allocation and computation resource configuration in distributed cloud/edge computing systems with stochastic workloads, IEEE Journal on Selected Areas in Communications 38 (2020) 1118–1132.

[19] A. Abdi, A. Girault, H. R. Zarandi, Erpot: A quad-criteria scheduling heuristic to optimize execution time, reliability, power consumption and temperature in multicores, IEEE Transactions on Parallel and Distributed Systems 30 (10) (2019) 2193–2210.

[20] A. Mseddi, W. Jaafar, H. Elbiaze, W. Ajib, Joint container placement and task provisioning in dynamic fog computing, IEEE Internet of Things Journal 6 (2019) 10028–10040.

[21] Y. Wu, J. Wu, L. Chen, J. Yan, Y. Luo, Efficient task scheduling for servers with dynamic states in vehicular edge computing, Computer Communications 150 (2020) 245–253.

[22] M. K. Hasan, E. Sundararajan, S. Islam, F. R. A. Ahmed, N. B. M. Babiker, A. I. Alzahrani, N. Alalwan, M. A. Khan, et al., A novel segmented random search based batch scheduling algorithm in fog computing, Computers in Human Behavior 158 (2024) 108269.

[23] J. Zhao, Y. Xia, X. Sun, T. Long, Q. Peng, S. Guo, F. Meng, Y. Dong, Q. Xia, Ets-ddpg: an energy-efficient and qos-guaranteed edge task scheduling approach based on deep reinforcement learning, Wireless Networks (2024) 1–14.

[24] X. Zhou, W. Liang, K. Yan, W. Li, I. Kevin, K. Wang, J. Ma, Q. Jin, Edge-enabled two-stage scheduling based on deep reinforcement learning for internet of everything, IEEE Internet of Things Journal 10 (2022) 3295–3304.

[25] Q. He, G. Cui, X. Zhang, F. Chen, S. Deng, H. Jin, Y. Li, Y. Yang, A game-theoretical approach for user allocation in edge computing environment, IEEE Transactions on Parallel and Distributed Systems 31 (2019) 515–529.

[26] W. Y. B. Lim, J. S. Ng, Z. Xiong, J. Jin, Y. Zhang, D. Niyato, C. Leung, C. Miao, Decentralized edge intelligence: A dynamic resource allocation framework for hierarchical federated learning, IEEE Transactions on Parallel and Distributed Systems 33 (2021) 536–550.

[27] H. Xiao, J. Zhao, Q. Pei, J. Feng, L. Liu, W. Shi, Vehicle selection and resource optimization for federated learning in vehicular edge comput-

ing, IEEE Transactions on Intelligent Transportation Systems 23 (2021) 11073–11087.

[28] T. Basaklar, A. A. Goksoy, A. Krishnakumar, S. Gumussoy, U. Y. Ogras, Dtrl: Decision tree-based multi-objective reinforcement learning for runtime task scheduling in domain-specific system-on-chips, ACM Transactions on Embedded Computing Systems 22 (5s) (2023) 1–22.

[29] G. Gokhale, S. S. Karimi Madahi, B. Claessens, C. Develder, Distill2explain: Differentiable decision trees for explainable reinforcement learning in energy application controllers, in: Proceedings of the 15th ACM International Conference on Future and Sustainable Energy Systems, 2024, pp. 55–64.

[30] G. K. Walia, M. Kumar, Computational offloading and resource allocation for iot applications using decision tree based reinforcement learning, Ad Hoc Networks 170 (2025) 103751.

[31] N. Khaledian, K. Khamforoosh, R. Akraminejad, L. Abualigah, D. Javaheri, An energy-efficient and deadline-aware workflow scheduling algorithm in the fog and cloud environment, computing 106 (2024) 109–137.

[32] S. Karami, S. Azizi, F. Ahmadizar, A bi-objective workflow scheduling in virtualized fog-cloud computing using nsga-ii with semi-greedy initialization, Applied Soft Computing 151 (2024) 111142.

[33] S. Swarup, E. M. Shakshuki, A. Yasar, Energy efficient task scheduling in fog environment using deep reinforcement learning approach, Procedia Computer Science 191 (2021) 65–75.

[34] Z. Wang, M. Goudarzi, M. Gong, R. Buyya, Deep reinforcement learning-based scheduling for optimizing system load and response time in edge and fog computing environments, Future Generation Computer Systems 152 (2024) 55–69.

[35] P. Choppara, S. S. Mangalampalli, Resource adaptive automated task scheduling using deep deterministic policy gradient in fog computing, IEEE Access 13 (2025) 25969–25994.

[36] Y. B. Zikria, R. Ali, M. K. Afzal, S. W. Kim, Next-generation internet of things (iot): Opportunities, challenges, and solutions, Sensors 21 (2021) 1174.

[37] O. Fivel, M. Klein, O. Maimon, Soft decision trees, in: Machine Learning for Data Science Handbook: Data Mining and Knowledge Discovery Handbook, Springer, 2023, pp. 143–170.

[38] N. Frosst, G. Hinton, Distilling a neural network into a soft decision tree, arXiv preprint arXiv:1711.09784 (2017).

[39] G. Nanfack, P. Temple, B. Frénay, Constraint enforcement on decision trees: A survey, ACM Computing Surveys (CSUR) 54 (10s) (2022) 1–36.

[40] R. S. Sutton, Reinforcement learning: An introduction, A Bradford Book (2018).

[41] A. Del Rio, D. Jimenez, J. Serrano, Comparative analysis of a3c and ppo algorithms in reinforcement learning: A survey on general environments, IEEE Access (2024).

[42] Z. Zhang, F. Zhang, Z. Xiong, K. Zhang, D. Chen, Lsia3cs: Deep-reinforcement-learning-based cloud–edge collaborative task scheduling in large-scale iiot, IEEE Internet of Things Journal 11 (2024) 23917–23930.

[43] S. S. Mangalampalli, G. R. Karri, S. N. Mohanty, S. Ali, M. Ijaz Khan, S. Abdullaev, S. A. AlQahtani, Multi-objective prioritized task scheduler using improved asynchronous advantage actor critic (a3c) algorithm in multi cloud environment, IEEE Access 12 (2024) 11354–11377.

[44] J. Beránek, S. Böhm, V. Cima, Task graphs for benchmarking schedulers (2019).

[45] A. Hamedi, A. Ghaedi, A. Soltanbeigi, A. Abdi, Schedge: A dynamic, multi-agent, and scalable scheduling simulator for iot edge, in: 2025 33rd International Conference on Electrical Engineering (ICEE), 2025.

[46] M. Ghobaei-Arani, A. Shahidinejad, A cost-efficient iot service placement approach using whale optimization algorithm in fog computing environment, Expert Systems with Applications 200 (2022) 117012.