Multi-Objective and Energy Efficient Reinforcement Learning for Edge AI Applications

By

Toygun Basaklar

A dissertation submitted in partial fulfillment of
the requirements for the degree of

Doctor of Philosophy

(Electrical and Computer Engineering)

at the

UNIVERSITY OF WISCONSIN–MADISON

2024

Date of final oral examination: 05/29/2024

The dissertation is approved by the following members of the Final Oral Committee:
Umit Y. Ogras, Associate Professor, Electrical and Computer Engineering, UW-Madison
Younghyun Kim, Associate Professor, Electrical and Computer Engineering, UW-Madison
Qiaomin Xie, Assistant Professor, Industrial and Systems Engineering, UW-Madison
Josiah Hanna, Assistant Professor, Computer Science, UW-Madison
Suat Gumussoy, Senior Research Scientist, Siemens

Dedicated to my *father, Can*, may his memory forever be a comfort and a reminder for success,
mother, Dilek, the kindest human being who shaped me into who I am,
brother, Berk, the problem solver, the musician, the dad, the engineer,
sister, Bike, the best strong, independent woman, the cool mom,
nieces, Belis and Mina, the joy and light of our family,
friends, my lifelong support system.

ACKNOWLEDGMENTS

I extend my deepest appreciation to my Ph.D. advisor, Prof. Umit Y. Ogras, for his unwavering support and guidance. The completion of this dissertation would not have been possible without his mentorship. His technical expertise and enthusiasm for research sparked many insightful discussions and enabled me to develop as an independent researcher. I deeply appreciate his endless sympathy and understanding over the years.

I sincerely thank Dr. Suat Gumussoy, who essentially served as a co-advisor during my Ph.D. His enthusiasm for research often made me question my own efforts, but it also motivated me to strive for excellence. Without his support and guidance, completing this thesis would have been incredibly difficult.

I am immensely thankful to Prof. Younghyun Kim, Prof. Qiaomin Xie, and Prof. Josiah Hanna for taking the time to serve on my Ph.D. defense committee. All of their suggestions and comments have been invaluable in improving the quality of this dissertation.

I also wish to thank my previous advisor during my Master's studies in Turkey, Prof. Ziya Ider, for laying the foundational skills and knowledge that have been invaluable in pursuing my Ph.D.

I am also grateful for all of my friends and colleagues at eLab: Dr. Samet Arda, Prof. Ganapati Bhat, Prof. Sumit Mandal, Dr. Anish Krishnakumar, Dr. Yigit Tuncel, Dr. Sizhe An, Alper Göksoy, Conrad Holt, Nuriye Yıldırım, Aditya Ukarande, Mingcong Cao, Alish Kanani, Lukas Pfromm, Jiahao Lin, and Xinmiao Xiong. Their help and support and the conversation made this journey a lot easier.

I deeply appreciate my lifelong support system, my friends: the US branch – Yigit, Ezgi, Mehmet, Matt, Melina, Luke, Stephanie, Kaan, Yavuz, Hande, Daniel, Aybüke, Gökhan, Gizem, Levent, Baran; the Canada branch – Emre, Zeynep; the Europe branch – Doğacan, Dilan, Tuğberk, Ezgi, Burkay, Muhammet; and the Turkey branch – Emir, Aslı, Onur, Barış, and Naz. Although they are all over the world, the deep connection and support we share have played a crucial role in my life, making my Ph.D. journey enjoyable.

This would not have been possible without the support of my parents, Can and Dilek, my brother Berk, and my family. Your love, support, and encouragement have been of immense support in my journey.

Finally, I would like to acknowledge and thank the Defense Advanced Research Projects Agency and the National Science Foundation for funding the research presented in this dissertation.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Reinforcement learning (RL) has demonstrated significant potential across a wide array of applications, including entertainment, robotics, finance, healthcare, autonomous vehicles, smart systems, and industrial automation. While RL is traditionally effective at optimizing tasks with a single objective, it often struggles with the complexities of real-world problems that involve multiple conflicting objectives. These scenarios require the simultaneous balancing of multiple, often conflicting, objectives, a task for which traditional single-objective RL methods are unsuited as they focus on optimizing a singular cumulative reward.

This shortcoming has driven the development of Multi-Objective Reinforcement Learning (MORL), specifically engineered to manage multiple objectives effectively. MORL must address several key challenges: i) Balancing and trading off between conflicting objectives is complex and requires sophisticated techniques to identify optimal trade-offs and ensure diverse, representative solutions across the entire preference space, ii) Many existing MORL approaches require repetitive training for different preferences, which is impractical in dynamic environments and computationally expensive, iii) Applying MORL in real-world scenarios like energy management for wearable devices, environmental monitoring, or task scheduling in computing platforms introduces complexities due to the need for optimal, feasible, and efficient solutions in resource-constrained environments. These applications require energy-efficient algorithms and reliable, timely responses despite limited resources and harsh conditions.

This dissertation aims to tackle these challenges and advance reinforcement learning to enable its application to real-world problems. It focuses on developing multi-objective, energy-aware, and hardware-friendly algorithms that can be effectively applied in resource-constrained environments. The core of the research is the development of novel MORL frameworks that enhance decision-making capabilities across diverse applications, including energy management for wearable devices, environment monitoring, and runtime task scheduling in domain-specific system-on-chips. The dissertation introduces several innovative algorithms. The

Preference-Driven Multi-Objective Reinforcement Learning (PD-MORL) framework is a significant contribution, designed to handle varying preferences dynamically without retraining the model. This framework is especially effective in environments where objectives change frequently. Further, the thesis presents energy-aware MORL algorithms for energy management in edge devices that optimize energy usage dynamically based on the available and harvested energy. Another contribution is the Decision Tree-based Multi-Objective Reinforcement Learning (DTRL) algorithm, which optimizes task scheduling to enhance the performance of system-on-chips under multiple criteria. This algorithm is designed to be hardware-friendly to make near-optimal decisions considering competing objectives within nanoseconds to be on par with the task execution times in specialized PEs. Experimental evaluations demonstrate that the proposed algorithms not only outperform existing techniques in terms of efficiency and effectiveness but also do so with fewer computational resources, making them suitable for real-world applications. The research bridges the gap between theoretical advancements and practical applications, offering a significant step forward in the application of reinforcement learning to multi-objective problems.

1 INTRODUCTION

Reinforcement learning (RL) has emerged as a powerful and versatile paradigm within the broader field of machine learning. At its core, RL involves an agent that learns to make decisions by interacting with an environment, receiving feedback in the form of rewards or penalties, and using this feedback to improve its future actions. This trial-and-error approach allows RL to tackle complex decision-making problems that are difficult to solve with traditional methods.

RL has demonstrated significant potential across a wide array of applications, including but not limited to:

- **Board/Video Games:** RL has achieved groundbreaking success in mastering complex games includeing AlphaGo [8], which defeated world champions in Go, Dota 2 [9] and Starcraft II [10], which achieved human-level performance. RL techniques enable the development of agents that can learn optimal strategies and adapt to different game scenarios through extensive self-play and exploration.

- **Robotics:** In robotics, RL is extensively used to enable autonomous agents to learn tasks such as navigation, manipulation, and locomotion. Robots equipped with RL algorithms can learn to perform complex tasks like assembling products, navigating through uncertain environments, and even interacting with humans in a socially acceptable manner [11].

- **Smart Systems and IoT:** RL is applied in smart grids, energy management systems, and Internet of Things (IoT) devices to optimize resource usage and

improve efficiency [12, 13].

- **Autonomous Vehicles:** In the domain of autonomous driving, RL is used to develop systems that can learn to drive by themselves. These systems learn to make real-time steering, acceleration, and braking decisions, handling various driving scenarios and road conditions [14, 15]. This capability is crucial for ensuring the safety and efficiency of autonomous vehicles.

- **Healthcare:** RL is increasingly used in healthcare for personalized treatment planning, drug discovery, and robotic surgery [16, 17]. It helps in developing adaptive treatment strategies that can respond to the changing health status of patients, thereby improving outcomes. RL algorithms can optimize the sequencing of treatments and adapt to the patient's response over time.

- **Finance:** In the financial sector, RL is used for portfolio management, algorithmic trading, and risk management [18, 19]. By learning from historical data and market trends, RL algorithms can make informed decisions to maximize returns and minimize risks. These systems can adapt to changing market conditions and develop strategies that balance risk and reward.

- **Industrial Automation:** RL is applied in manufacturing processes for optimizing production lines, predictive maintenance, and quality control [20]. It enables machines to adapt to varying conditions and demands, improving productivity and reducing downtime. RL-based systems can learn to optimize the allocation of resources and scheduling of tasks to enhance efficiency.

- **Chip Design and Placement:** In the field of chip design, RL has been used to optimize the placement of components on a chip, which is a complex problem involving numerous variables and constraints. RL techniques can significantly reduce the time and effort required to design efficient and high-performance chips [21, 22].

While RL has proven effective in optimizing single-objective tasks, many real-world problems are inherently multi-objective. These problems require balancing multiple, often conflicting, objectives simultaneously. For example, (i) in robotics, an agent may need to optimize both speed and energy efficiency; (ii) in smart grids, the objectives might include minimizing the cost while maximizing reliability and sustainability; and (iii) in healthcare, treatment strategies must balance efficacy and side effects. For RL to be applicable to real-world problems, it is crucial to develop algorithms that address multi-objective functionality, energy awareness, and hardware friendliness. Energy awareness is critical, especially for applications involving resource-constrained environments like wearable devices and edge computing. Energy-efficient RL algorithms can significantly extend the operational life of battery-powered devices and reduce overall energy consumption, which is essential for sustainability and cost-effectiveness. Hardware friendliness ensures that RL algorithms can be efficiently implemented on various hardware platforms, including specialized processing elements in embedded systems and domain-specific systems-on-chips (DSSoCs). This involves designing algorithms that are computationally efficient and can operate within the constraints of the application. Multi-objective functionality allows RL algorithms to simultaneously

consider and optimize multiple objectives, providing solutions that are more aligned with real-world needs. Traditional single-objective RL approaches are inadequate for multi-faceted problems, as they are designed to optimize a single cumulative reward. This limitation has driven the development of Multi-Objective Reinforcement Learning (MORL), which aims to handle multiple objectives effectively [23]. However, MORL faces several challenges: balancing conflicting objectives requires sophisticated techniques, existing approaches often need repetitive and costly training, and real-world applications demand optimal, feasible, and energy-efficient solutions in resource-constrained environments.

1.1 Challenges in MORL

1.1.1 Balancing Trade-offs Between Multiple Objectives

In MORL, in contrast to single-objective environments, performance is measured using multiple objectives. Consequently, there are multiple Pareto-optimal solutions as a function of the preference between objectives [24]. MORL requires balancing and trading off between conflicting objectives, which is non-trivial. Identifying the optimal trade-offs and ensuring that the solutions are diverse and representative of the entire preference space is a significant challenge. This requires sophisticated techniques to navigate the multi-objective landscape and to discover Pareto optimal solutions that appropriately balance the different objectives according to the specified preferences.

1.1.2 Scalability and Efficiency

Many existing MORL approaches necessitate repetitive training for different preference settings, which is impractical for dynamic environments where objectives and constraints can frequently change. Moreover, the computational complexity of finding a diverse and dense set of Pareto optimal solutions increases exponentially with the problem's dimensionality and the size of the action space. For instance, training agents to learn all possible outcomes and interactions in video games demands hundreds of TPU or GPU days, incurring costs of tens or hundreds of thousands of dollars from a typical cloud service provider. Such extensive training is infeasible in many application domains due to time and resource constraints. Therefore, scalable and efficient algorithms are required to handle the large state and action spaces typical of real-world applications.

1.1.3 Real-World Applicability

Applying MORL in real-world scenarios, such as energy management for wearable devices, environmental monitoring systems, or task scheduling in computing platforms, introduces additional complexities. These applications require solutions that are not only optimal but also feasible and efficient in resource-constrained environments. For instance, wearable devices have limited battery life and computational power, necessitating energy-efficient algorithms. Similarly, environmental monitoring systems, such as those used for wildfire detection, require reliable and timely responses while operating under harsh conditions with limited resources.

1.2 Contributions

This dissertation aims to tackle these challenges and advance reinforcement learning to enable its application to real-world problems by developing multi-objective, energy-aware, and hardware-friendly algorithms that can be effectively applied in resource-constrained environments. It bridges the gap between theoretical advancements and practical applications, paving the way for more intelligent and adaptive systems in various domains.

In summary, this dissertation makes the following contributions:

- **PD-MORL: Preference-Driven Multi-Objective Reinforcement Learning Algorithm** [25]: It proposes a novel MORL algorithm that trains a *single universal network to cover the entire preference space scalable to continuous robotic tasks*. The proposed approach, Preference-Driven MORL (PD-MORL), utilizes the preferences as guidance to update the network parameters. It also employs a novel parallelization approach to increase sample efficiency. We show that PD-MORL achieves up to 25% larger hypervolume for challenging continuous control tasks and uses an order of magnitude fewer trainable parameters compared to prior approaches.

- **A Comprehensive Multi-Objective Energy Management Approach for Wearable Devices with Dynamic Energy Demands**: It learns the trade-off between meeting the application's energy demand and maintaining the battery energy level. We deployed our framework on a wearable device prototype using TensorFlow Lite for Micro, leveraging its small (less than 120 KB) memory

footprint. Evaluations show that tinyMAN-MO operates within 10% of the Pareto-optimal solutions with only 1.98 ms execution time and 23.17 μJ energy consumption overhead.

- **GEM-RL: Generalized Energy Management of Wearable Devices using Reinforcement Learning** [26]: GEM-RL learns the trade-off between utilization and the battery energy level of the target device under dynamic EH patterns and battery conditions. It also uses a lightweight approximate dynamic programming (ADP) technique that utilizes the trained MORL agent to optimize the utilization of the device over a longer period. Thorough experiments show that, on average, GEM-RL achieves Pareto front solutions within 5.4% of the offline Oracle for a given day. For a 7-day horizon, it achieves utility up to 4% within the offline Oracle and up to 50% higher utility compared to baseline EM approaches. The hardware implementation on a wearable device shows negligible execution time (1.98 ms) and energy consumption (23.17 μJ) overhead.

- **A Self-Sustained CPS Design for Reliable Wildfire Monitoring** [27]: It presents the first self-sustained cyber-physical system that dynamically co-optimizes the wildfire detection accuracy and active time of sensors. The proposed approach employs reinforcement learning to train a policy that controls the sensor operations as a function of the environment (i.e., current sensor readings), harvested energy, and battery level. The proposed cyber-physical system is evaluated extensively using real-life temperature, wind, and solar energy harvesting datasets and an open-source wildfire simulator.

In long-term (5 years) evaluations, the proposed framework achieves 89% uptime, which is 46% higher than a carefully tuned heuristic approach. At the same time, it averages a 2-minute initial response time, which is at least 2.5× faster than the same heuristic approach. Furthermore, the policy network consumes 0.6 mJ per day on the TI CC2652R microcontroller using TensorFlow Lite for Micro, which is negligible compared to the daily sensor suite energy consumption.

- **DTRL: Decision Tree-based Multi-Objective Reinforcement Learning for Runtime Task Scheduling in Domain-Specific System-on Chips** [28]: DTRL trains a single global differentiable decision tree (DDT) policy that covers the entire objective space quantified by a preference vector. Extensive experimental evaluations demonstrate that DTRL captures the trade-off between execution time and power consumption, thereby generating a Pareto set of solutions using a single policy. Furthermore, comparison with state-of-the-art heuristic–, optimization–, and machine learning-based schedulers shows that DTRL achieves up to 9× higher performance and up to 3.08× reduction in energy consumption. The trained DDT policy achieves 120 ns inference latency on Xilinx Zynq ZCU102 FPGA at 1.2 GHz, resulting in negligible runtime overheads. Evaluation on the same hardware shows that DTRL achieves up to 16% higher performance than a state-of-the-art heuristic scheduler.

The rest of the dissertation is organized as follows: Chapter 2 introduces the PD-MORL framework. Chapter 3 and 4 present multi-objective energy management approaches for wearable devices. Chapter 5 presents a self-sustained CPS

design for reliable wildfire monitoring using reinforcement learning. Chapter 6 DTRL algorithm for runtime task scheduling in domain-specific systems-on-chips (DSSoCs). Finally, Chapter 7 concludes the dissertation with directions for future work.

2 PD-MORL: PREFERENCE-DRIVEN MULTI-OBJECTIVE REINFORCEMENT LEARNING ALGORITHM

2.1 Motivation and Contributions

Existing approaches for multi-objective optimization generally transform the multi-dimensional objective space into a single dimension by statically assigning weights (preferences) to each objective [29]. Then, they use standard RL algorithms to obtain a policy optimized for the given preferences. These approaches suffer when the objectives have widely varying magnitudes since setting the preference weights requires application domain expertise. More importantly, they can find only a single solution for a given set of goals and constraints. Thus, they need to repeat the training progress when the constraints or goals change. However, repetitive retraining is impractical since the constraints and design can change frequently depending on the application domain. Therefore, obtaining a set of Pareto front solutions that covers the entire preference space with a single training is critical [4, 1, 30].

This work presents a novel multi-objective reinforcement learning algorithm *using a single policy network that covers the entire preference space scalable to continuous robotic tasks*. At its core, it uses a multi-objective version of Q-Learning, where we approximate the Q-values with a neural network. This network takes the states and preferences as inputs during training. Making the preferences input parameters allows the trained model to produce the optimal policy for any user-specified preference at run-time. Since the user-specified preferences effectively drive the

policy decisions, it is called preference-driven (PD) MORL. For each episode during training, we randomly sample a preference vector ($\boldsymbol{\omega} \in \Omega : \sum_{i=0}^{L} \omega_i = 1$) from a uniform distribution. Since the number of collected transitions by interacting with the environment for some preferences may be underrepresented, we utilize hindsight experience replay buffer (HER) [31]. As a key insight, we observe that the preference vectors have similar directional angles to the corresponding vectorized Q-values for a given state. Using the insight, we utilize the cosine similarity between the preference vector and vectorized Q-values in the Bellman's optimality operator to guide the training. However, not every Pareto front perfectly aligned with the preference vectors. To mitigate this adverse effect, we fit a multi-dimensional interpolator to project the original preference vectors ($\boldsymbol{\omega} \in \Omega$) to normalized solution space to align preferences with the multi-objective solutions. The projected preference vectors are used in our novel preference-driven optimality operator to obtain the target Q-values. Additionally, to increase the sample efficiency of the algorithm, we divide the preference space into sub-spaces and assign a child process to these sub-spaces. Each child process is responsible for its own preference sub-space to collect transitions. This parallelization provides efficient exploration during training, assuring that there is no bias towards any preference sub-space.

PD-MORL can be employed in any off-policy RL algorithm. We develop a multi-objective version of the double deep Q-network algorithm with hindsight experience replay buffer (MO-DDQN-HER) [32] for problems with discrete action spaces and evaluate PD-MORL's performance on two commonly used MORL benchmarks: Deep Sea Treasure [23] and Fruit Tree Navigation Task [1]. We

specifically choose these two benchmarks to make a fair comparison with the prior approach [1] that also aims to achieve a unified policy network. Additionally, we develop a multi-objective version of the Twin Delayed Deep Deterministic policy gradient algorithm with hindsight experience replay buffer (MO-TD3-HER) [33] for problems with continuous action spaces. Using MO-TD3-HER, we evaluate PD-MORL on the multi-objective continuous control tasks such as MO-Walker2d-v2, MO-HalfCheetah-v2, MO-Ant-v2, MO-Swimmer-v2, MO-Hopper-v2 that are presented by Xu et al. [4]. With the combination of the use of the cosine similarity term, the HER, and the parallel exploration, PD-MORL achieves up to 78% larger hypervolume for simple benchmarks and 25% larger hypervolume for continuous control tasks and uses an order of magnitude fewer trainable parameters compared to prior approaches while achieving broad and dense Pareto front solutions. We emphasize that it achieves these results with a single policy network.

2.2 Related Work

Existing MORL approaches can be classified as single-policy, multi-policy, and meta-policy approaches. The main difference among them is the number of policies learned during training. Single-policy approaches transform a multi-objective problem into a single-objective problem by combining the rewards into a single scalar reward using a scalarization function. Then, they use standard RL approaches to maximize the scalar reward [34]. Most of the previous studies find the optimal policy for *a given preference* between the objectives using scalarization [35].

Additionally, recent work takes an orthogonal approach and encodes preferences as constraints instead of scalarization [30]. These approaches have two primary drawbacks: they require domain-specific knowledge, and preferences must be set beforehand.

Multi-policy approaches aim to obtain a set of policies that approximates the Pareto front of optimal solutions. The most widely used approach is to repeatedly perform a single-policy algorithm over various preferences [36, 7, 37]. However, this approach suffers from a large number of objectives and dense Pareto solutions for complex control problems. In contrast, [38] suggests a manifold-based policy search MORL approach which assumes policies to be sampled from a manifold. This manifold is defined as a parametric distribution over the policy parameter space. Their approach updates the manifold according to an indicator function such that sample policies yield an improved Pareto front. Parisi et al. [39] extend this method with hypervolume and non-dominance count indicator functions using importance sampling to increase the sample efficiency of the algorithm. However, the number of parameters to model the manifold grows quadratically [40] as the number of policy parameters increases. Therefore, these approaches are not scalable for complex problems such as continuous control robotics tasks to achieve a dense Pareto front where deep neural networks with at least thousands of parameters are needed.

[3] and [1] use a multi-objective Q-learning approach that simultaneously learns a set of policies over multiple preferences. These studies use a single network that takes preferences as inputs and uses vectorized value function updates in contrast

to standard scalarized value function updates. An orthogonal approach by [40] is the meta-policy approach that frames MORL as a meta-learning problem using a task distribution given by a distribution over the preference space. The authors first train a meta-policy to approximate the Pareto front implicitly. Then, they obtain the Pareto optimal solution of a given preference by only fine-tuning the meta-policy with a few gradient updates. A more recent study proposes an efficient evolutionary learning algorithm to update a population of policies simultaneously in each run to improve the approximation of the Pareto optimal solutions [4]. All non-dominated policies for each generation of policies are stored as an external Pareto front archive during the training. Finally, it outputs this Pareto front archive as approximated Pareto front solutions. However, this approach obtains a policy network for each solution on the Pareto front. Moreover, there is no specific knowledge of correspondence between preference vectors and solutions in this Pareto front.

Distinct from the prior work, PD-MORL utilizes the relation between the Q-values and the preferences and proposes a novel update rule, including the cosine similarity between the Q-values and preferences. Additionally, PD-MORL increases its sample efficiency by using a novel parallelization approach with HER and provides efficient exploration. The combination of the cosine similarity term, the HER, and the parallel exploration achieve up to 25% larger hypervolume for challenging continuous control tasks compared to prior approaches using an order of magnitude fewer trainable parameters.

2.3 Background

MORL requires learning several tasks with different rewards simultaneously. Each objective has an associated reward signal, transforming the reward from a scalar to a vector, $\mathbf{r} = [r_1, r_2, ..., r_L]^T$, where L is the number of objectives. A multi-objective problem can be formulated as a multi-objective Markov decision process (MOMDP) defined by the tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathbf{r}, \Omega, f_\omega \rangle$, where $\mathcal{S}$, $\mathcal{A}$, $\mathcal{P}(s'|s, a)$, $\mathbf{r}$, and Ω represents state space, action space, transition distribution, reward vector, and preference space, respectively. The function $f_\omega(\mathbf{r}) = \omega^T \mathbf{r}$ yields a scalarized reward using a preference of $\omega \in \Omega$. If ω is taken as a fixed vector, the MOMDP boils down to a standard MDP, which can be solved using standard RL techniques. However, if we consider all possible returns from a MOMDP and all possible preferences in Ω, a set of non-dominated policies called the Pareto front can be obtained. A policy π is Pareto optimal if there is no other policy π' that improves its expected return for an objective without degrading the expected return of any other objective. For complex problems, such as continuous control tasks, obtaining an optimal Pareto front using an RL setting is an NP-hard problem [4]. Hence, the main goal of the MORL algorithms is to obtain an approximation of the Pareto front. The quality of the approximated Pareto front is typically measured by two metrics: (i) hypervolume and (ii) sparsity [23].

Definition 1 (Hypervolume Indicator). Let P be a Pareto front approximation in an L-dimensional objective space and contains N solutions. Let $\mathbf{r}_0 \in \mathbb{R}^L$ be the

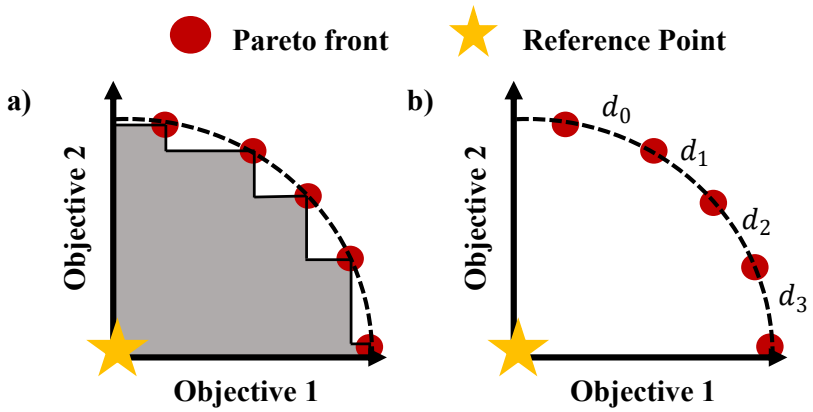


Figure 2.1: (a) The hypervolume between a reference point and solutions in Pareto front is shown with the shaded area. (b) The sparsity is the average square distance ($d_i, i = \{0, \ldots, 3\}$) between consecutive solutions in the Pareto front.

reference point. Then, the hypervolume indicator is defined as:

$$I_H(P) := \Lambda(H(P, \mathbf{r}_0)) \tag{2.1}$$

where $H(P, \mathbf{r}_0) = \{\mathbf{z} \in \mathbb{R}^L | \exists\, 1 \leqslant i \leqslant N : \mathbf{r}_0 \preceq \mathbf{z} \preceq P_i\}$ with P_i being the i^{th} solution in P and $\preceq$ is the relation operator of multi-objective dominance. $\Lambda(\cdot)$ denotes the Lebesgue measure with $\Lambda(H(P, \mathbf{r}_0)) = \int_{\mathbb{R}^L} \mathbb{1}_{H(P,\mathbf{r}_0)}(\mathbf{z})d\mathbf{z}$ and $\mathbb{1}_{H(P,\mathbf{r}_0)}$ being the characteristic function of $H(P, \mathbf{r}_0)$.

Definition 2 (Sparsity). For the same P, sparsity is defined as:

$$Sp(P) := \frac{1}{N-1} \sum_{j=1}^{L} \sum_{i=1}^{N-1} (P_{i_j} - P_{i+1_j})^2 \tag{2.2}$$

where P_i is the i^{th} solution in P and P_{i_j} is the sorted set for the j^{th} objective.

Improvement in any objective manifests as an increase in the hypervolume, as

illustrated in Figure 2.1. However, the hypervolume alone is insufficient to assess the quality of the Pareto front. For instance, hypervolume may increase due to an increase in only one of the objectives which indicates that the algorithm does not improve other objectives. The sparsity metric is introduced to assess whether the Pareto front solutions are dense or sparse. A larger hypervolume indicates that a more desired Pareto front approximation is achieved. Lower sparsity values imply that a dense set of solutions is achieved.

2.4 PD-MORL: Preference Driven MORL Algorithm

This section introduces the proposed preference-driven MORL algorithm. We first provide a theoretical analysis of PD-MORL based on the multi-objective version of Q-learning [41] by following the framework provided by Yang et al. [1]. The proofs for this theoretical analysis are available in Appendix A. We then introduce multi-objective double deep Q-networks with a preference-driven optimality operator for problems with discrete action space. Finally, we extend the Twin Delayed Deep Deterministic Policy Gradient (TD3) algorithm [42] to a multi-objective version utilizing our proposed approach for problems with continuous action space.

2.4.1 Theoretical Analysis of PD-MORL

In standard Q-learning, the value space is defined as $\mathcal{Q} \in \mathbb{R}^{\mathcal{S}\times\mathcal{A}}$, containing all bounded functions $Q(s,a)$ which are the estimates of the total expected rewards when the agent is at state s, taking action a. We extend it to a multi-objective value

space by defining the value space as $\mathcal{Q} \in \mathbb{R}^{L^{\mathcal{S}\times\mathcal{A}}}$, containing all bounded functions $\mathbf{Q}(s, a, \boldsymbol{\omega})$ which are the estimates of expected total rewards under preference $\boldsymbol{\omega} \in \mathbb{R}^{L}_{\geqslant 0} : \sum_{i=0}^{L} \omega_i = 1$. We then define a metric in this value space as:

$$d(\mathbf{Q}, \mathbf{Q}') := \sup_{s\in\mathcal{S}, a\in\mathcal{A}, \boldsymbol{\omega}\in\Omega} |\boldsymbol{\omega}^T(\mathbf{Q}(s, a, \boldsymbol{\omega}) - \mathbf{Q}'(s, a, \boldsymbol{\omega}))|. \tag{2.3}$$

This metric gives the distance between $\mathbf{Q}$ and $\mathbf{Q}'$ as the supremum norm of the scalarized distance between these two vectors where the identity of indiscernables ($d(\mathbf{Q}, \mathbf{Q}') = 0 \Leftrightarrow \mathbf{Q} = \mathbf{Q}'$) does not hold and thus, makes metric d a pseudo-metric. Further details for the axioms of this pseudo-metric are given in Appendix A. In the following Theorem, we show that the metric space equipped with this metric is complete since the limit of the sequence of operators is required to be in this space.

Theorem 2.1 (Multi-objective Metric Space $(\mathcal{Q}, d)$ is Complete)**.** *The metric space* $(\mathcal{Q}, d)$ *is complete and every Cauchy sequence* $\mathbf{Q}_n(s, a, \boldsymbol{\omega})$ *is convergent in metric space* $(\mathcal{Q}, d)$ $\forall\, s, a, \boldsymbol{\omega} \in \mathcal{S}, \mathcal{A}, \Omega$.

Given a policy π and sampled transition τ, we can define multi-objective Bellman's evaluation operator $\mathcal{T}_\pi$ using the metric space $(\mathcal{Q}, d)$ as:

$$(\mathcal{T}_\pi \mathbf{Q})(s, a, \boldsymbol{\omega}) := \mathbf{r}(s, a) + \gamma \mathbb{E}_{\mathcal{T}\sim(\mathcal{P},\pi)} \mathbf{Q}(s', a', \boldsymbol{\omega}) \tag{2.4}$$

where $(s', a', \boldsymbol{\omega})$ denotes the next state-action-preference pair and $\gamma \in (0, 1)$ is the discount factor.

We define a *preference-driven optimality operator* $\mathcal{T}$ by adding a cosine similarity

term between preference vectors and state-action values to the multi-objective Bellman's optimality operator as:

$$(\mathcal{T}\mathbf{Q})(s, a, \boldsymbol{\omega}) := \mathbf{r}(s, a) + \gamma \mathbb{E}_{s' \sim \mathcal{P}(\cdot|s,a)} \mathbf{Q}(s', \sup_{a' \in \mathcal{A}} (S_c(\boldsymbol{\omega}, \mathbf{Q}(s', a', \boldsymbol{\omega})) \cdot (\boldsymbol{\omega}^T \mathbf{Q}(s', a', \boldsymbol{\omega}))), \boldsymbol{\omega}) \tag{2.5}$$

where $S_c(\boldsymbol{\omega}, \mathbf{Q}(s', a', \boldsymbol{\omega}))$ denotes the cosine similarity between preference vector and Q-value. This term enables our optimality operator to choose actions that align the preferences with the Q-values and maximize the target value, as elaborated in Section 2.4.2 under the preference alignment subtitle.

Theorem 2.2 (Multi-objective Bellman's Evaluation Operator is Contraction)**.** *Let* $(\mathcal{Q}, d)$ *be a complete metric space* (*as in Theorem 2.1*)*. Let* $\mathbf{Q}$ *and* $\mathbf{Q}'$ *be any two multi-objective Q-value functions in this space. The multi-objective Bellman's evaluation operator is a contraction and* $d(\mathcal{T}_\pi \mathbf{Q}, \mathcal{T}_\pi \mathbf{Q}') \leqslant \gamma d(\mathbf{Q}, \mathbf{Q}')$ *holds for the Lipschitz constant* γ (*the discount factor*)*.*

Theorem 2.3 (Preference-driven Multi-objective Bellman's Optimality Operator is Contraction)**.** *Let* $(\mathcal{Q}, d)$ *be a complete metric space. Let* $\mathbf{Q}$ *and* $\mathbf{Q}'$ *be any two multi-objective Q-value functions in this space. The preference-driven multi-objective Bellman's optimality operator is a contraction and* $d(\mathcal{T}\mathbf{Q}, \mathcal{T}\mathbf{Q}') \leqslant \gamma d(\mathbf{Q}, \mathbf{Q}')$ *holds for the Lipschitz constant* γ (*the discount factor*)*.*

Theorem 2.2 and Theorem 2.3 state that multi-objective evaluation and optimality operators are contractions. They ensure that we can apply our optimality operator in Equation 2.5 iteratively to obtain the optimal multi-objective value function given by Theorem 2.4 and Theorem 2.5 below, respectively.

Theorem 2.4 (Preference-driven Multi-objective Optimality Operator Converges to a Fixed-Point)**.** *Let* $(\mathcal{Q}, \mathrm{d})$ *be a complete metric space which is proved above and let* $\mathcal{T} : \mathcal{Q} \to \mathcal{Q}$ *be a contraction on* $\mathcal{Q}$ *with modulus* γ*. Then,* $\mathcal{T}$ *has a unique fixed point* $\mathbf{Q}^* \in \mathcal{Q}$ *such that* $\mathcal{T}(\mathbf{Q}) = \mathbf{Q}^*$*.*

Theorem 2.5 (Optimal Fixed Point of Optimality Operator)**.** *Let* $\mathbf{Q}^* \in \mathcal{Q}$ *be the optimal multi-objective value function, such that it takes multi-objective Q-value corresponding to the supremum of expected discounted rewards under a policy* π *then* $\mathbf{Q}^* = \mathcal{T}(\mathbf{Q}^*)$*.*

These two theorems state that applying our multi-objective optimality operator T iteratively on any multi-objective Q-value function $\mathbf{Q}$ converges to the optimal $\mathbf{Q}^*$ under metric d and the Bellman equation holds at the same fixed-point. Hence, we use this optimality operator to obtain target values when optimizing the loss function in Algorithm 1.

2.4.2 Preference Driven MO-DDQN-HER and MO-TD3-HER

The main objective of this work is to obtain a single policy network to approximate the Pareto front that covers the entire preference space. For this purpose, we extend double deep Q-network (DDQN) [43] to a multi-objective version (MO-DDQN) with the preference-driven optimality operator to obtain a single parameterized function representing the $\mathcal{Q} \in \mathbb{R}^{\mathrm{L}^{\mathcal{S}\times\mathcal{A}}}$ with parameters θ. This network takes $s, \boldsymbol{\omega}$ as input and outputs $|\mathcal{A}| \times \mathrm{L}$ Q-values, as described in Algorithm 1. To pull $\mathbf{Q}$ towards

$T(\mathbf{Q})$, MO-DDQN minimizes the following loss function at each step k:

$$L_k(\theta) = \mathbb{E}_{(s,a,\mathbf{r},s',\boldsymbol{\omega})\sim\mathcal{D}}\left[\left(\mathbf{y} - \mathbf{Q}(s,a,\boldsymbol{\omega};\theta)\right)^2\right] \tag{2.6}$$

where $\mathcal{D}$ is the experience replay buffer that stores transitions $(s, a, \mathbf{r}, s', \boldsymbol{\omega})$ for every time step, $\mathbf{y} = \mathbf{r} + \gamma\mathbf{Q}(s', \sup_{a'\in\mathcal{A}}(S_c(\boldsymbol{\omega}, \mathbf{Q}(s', a', \boldsymbol{\omega}))\cdot(\boldsymbol{\omega}^T\mathbf{Q}(s', a', \boldsymbol{\omega})));\theta')$ denotes the preference-driven target value which is obtained using target Q-network's parameters θ'. Here, $S_c(\boldsymbol{\omega}, \mathbf{Q}(s', a', \boldsymbol{\omega}))$ denotes the cosine similarity between the preference vector and the Q-values. Without the cosine similarity term, the supremum operator yields the action that only maximizes $\boldsymbol{\omega}^T\mathbf{Q}(s', a', \boldsymbol{\omega})$. This may pull the target Q-values in the wrong direction, especially when the scales of the objectives are in different orders of magnitude. For instance, let us assume a multi-objective problem with percent energy efficiency ($\in [0, 1]$) being the first objective and the latency ($\in [1, 10]$) in milliseconds being the second objective. Let us also assume that the preference vector ($\boldsymbol{\omega} = \{0.9, 0.1\}$) favors the energy efficiency objective and there are two separate actions (a_1, a_2) that yields Q-values of $Q_1 = \{0.9,\ 1\}, Q_2 = \{0.1,\ 10\}$ respectively. The supremum operator chooses action, a_2 since $\boldsymbol{\omega}^T Q_2 = 1.09$ is higher than $\boldsymbol{\omega}^T Q_1 = 0.91$. Since the scales of these two objectives are in different orders of magnitudes, $\sup_{a'\in\mathcal{A}}(\boldsymbol{\omega}^T\mathbf{Q}(s', a', \boldsymbol{\omega}))$ always chooses the action that favors the latency objective. This behavior negatively affects the training since the target Q-values are pulled in the wrong direction. On the contrary, our novel cosine similarity term enables the optimality operator to choose actions that align the preferences with the Q-values and maximize the target value at the same time. The supremum operator now chooses action, a_1

since $S_c \cdot \boldsymbol{\omega}^T Q_1 = 0.75 \times 0.91 = 0.68$ is higher than $S_c \cdot \boldsymbol{\omega}^T Q_2 = 0.12 \times 1.09 = 0.13$. With this addition, the algorithm disregards the large $\boldsymbol{\omega}^T \mathbf{Q}(s', a', \boldsymbol{\omega})$ where the preference vector and the Q-values are not aligned since the $S_c(\boldsymbol{\omega}, \mathbf{Q}(s', a', \boldsymbol{\omega}))$ takes small values.

For each episode during training, we randomly sample a preference vector $(\boldsymbol{\omega} \in \Omega : \sum_{i=0}^{L} \omega_i = 1)$ from a uniform distribution. For example, assume the randomly sampled preference vector $(\boldsymbol{\omega})$ favors energy efficiency rather than speed in multi-objective continuous control tasks. Increasing energy efficiency (i.e., lower speed) will increase the number of transitions before reaching a terminal condition since it decreases the risk of reaching a terminal state (e.g., fall). Hence, transitions that favor speed instead of energy efficiency may be underrepresented in the experience replay buffer in this example. This behavior may create a bias towards overrepresented preferences, and the network cannot learn to cover the entire preference space. To overcome this issue, we employ hindsight experience replay buffer (HER) [31], where every transition is also stored with N_ω randomly sampled preferences $(\boldsymbol{\omega}' \in \Omega : \sum_{i=0}^{L} \omega'_i = 1)$ different than the original preference of the transition.

To increase the sample efficiency of our algorithm, we also divide the preference space into C_p sub-spaces $(\tilde{\Omega})$, where C_p denotes the number of child processes. Each child process is responsible for its own preference sub-space. The agent, hence the network, is shared among child processes and the main process. In each child process, for each episode, we randomly sample a preference vector from child's preference sub-space $(\boldsymbol{\omega} \in \tilde{\Omega} : \sum_{i=0}^{L} \omega_i = 1)$. These C_p child processes run in

Algorithm 1: Preference Driven MO-DDQN-HER

Input: Minibatch size N_m, Number of time steps N, Discount factor γ, Target network update coefficient τ, Multi-dimensional interpolator $I(\boldsymbol{\omega})$.
Initialize: Replay buffer $\mathcal{D}$, Current $\mathbf{Q}_\theta$ and target network $\mathbf{Q}_{\theta'} \leftarrow \mathbf{Q}_\theta$ with parameters θ and θ'.
for $n = 0$: N **do**
    `// Child Process`
    Initialize $t = 0$ and $done = False$.
    Reset the environment to randomly initialized state s_0.
    Sample a preference vector $\boldsymbol{\omega}$ from the subspace $\tilde{\Omega}$.
    while $done = False$ **do**
        Observe state s_t and select an action a_t ϵ-greedily:
        $$a_t = \begin{cases} a \in A & w.p. \quad \epsilon \\ \max_{a \in \mathcal{A}} \boldsymbol{\omega}\mathbf{Q}(s_t, a, \boldsymbol{\omega}; \theta), & w.p. \quad 1-\epsilon \end{cases}$$
        Observe $\mathbf{r}$, s', and $done$.
        Transfer $(s_t, a_t, \mathbf{r}_t, s', \boldsymbol{\omega}, done)$ to main process.
    `// Main Process`
    Sample N_ω preferences $\boldsymbol{\omega}'$
    for $j = 1$: N_ω **do**
        Store transition $(s_t, a_t, \mathbf{r}_t, s', \boldsymbol{\omega}'_j, done)$ in $\mathcal{D}$
    Sample N_m transitions from $\mathcal{D}$.
    $\boldsymbol{\omega}_p \leftarrow I(\boldsymbol{\omega})$
    $\mathbf{y} \leftarrow \mathbf{r} + \gamma \mathbf{Q}(s', \sup_{a' \in \mathcal{A}}(S_c(\boldsymbol{\omega}_p, \mathbf{Q}(s', a', \boldsymbol{\omega})) \cdot (\boldsymbol{\omega}^T \mathbf{Q}(s', a', \boldsymbol{\omega})); \theta')$
    $L_k(\theta) = \mathbb{E}_{(s,a,\mathbf{r},s',\boldsymbol{\omega}) \sim \mathcal{D}} \left[\left(\mathbf{y} - \mathbf{Q}(s, a, \boldsymbol{\omega}; \theta) \right)^2 \right]$
    Update θ_k by applying SGD to $L_k(\theta)$.
    Update target network parameters $\theta'_k \leftarrow \tau\theta_k + (1-\tau)\theta'_k$

parallel to collect transitions. These transitions are stored inside the HER in the main process. Network architectures and implementation details are available Appendix B. This parallelization provides efficient exploration during training.

Preference alignment: A solution in the Pareto front may not align perfectly with its preference vector. This may result in a bias in our updating scheme due to the cosine similarity term we introduced. To mitigate this adverse effect, we fit a

multi-dimensional interpolator to project the original preference vectors ($\boldsymbol{\omega} \in \Omega$) to normalized solution space to align preferences with the multi-objective solutions. We identify the key preferences and obtain solutions for each of these preferences. We set key preference points as $\boldsymbol{\omega}_j = 1 : j = i, \boldsymbol{\omega}_j = 0 : j \neq i \; \forall \; i, j \in \{0, \ldots, L\}$ where i^{th} element corresponds to the objective we try to maximize. A preference vector of $\boldsymbol{\omega} = \{\frac{1}{L}, \ldots, \frac{1}{L}\}$ is also added to this key preference set. For example, for a two-dimensional objective space, the key preference set becomes $\{[1,\ 0], [0.5,\ 0.5], [0,\ 1]\}$. We first obtain solutions for each preference in this key preference set by training the agent with a fixed preference vector and without using HER (see Figure 2.2(a)). We then use these solutions to obtain a normalized solution space, as shown in Figure 2.2(b). Here, normalization is the process of obtaining unit vectors for these solutions. For example, the normalized vector for a solution f is described as $\hat{f} = \frac{f}{|f|}$. Then, we use the normalized solution space and key preference point to fit a multi-dimensional interpolator $I(\boldsymbol{\omega})$ to project the original preference vectors ($\boldsymbol{\omega} \in \Omega$) to the normalized solution space. As a result, we obtain projected preference vectors ($\boldsymbol{\omega}_p$) as illustrated in Figure 2.2(c). These projected preference vectors are incorporated in the cosine similarity term of the preference-driven optimality operator. This practical modification extends to theoretical results as the proof for Theorem 2.3 is also valid for $\boldsymbol{\omega}_p$. The interpolator is also updated during training as PD-MORL may find new non-dominated solutions for identified key preferences.

Extension to continuous action space: Finally, we extend the TD3 algorithm to a multi-objective version using PD-MORL for problems with continuous action space. In this case, the target values are no longer computed using the optimality

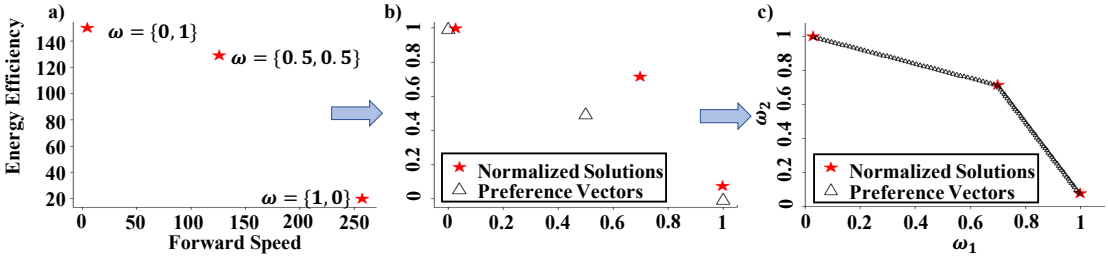


Figure 2.2: Overview of the interpolation scheme for the MO-Swimmer-v2 problem. (a) Obtained solutions for key preference points. (b) Normalized solution space and corresponding preference vectors. (c) Interpolated preference space.

operator. Instead, the actions for the target Q-value are determined by the target actor network. The actor network is updated to obtain a policy that maximizes the Q-values generated by the critic network. This update rule eventually boils down to obtaining a policy that maximizes the expected discounted rewards since, by definition, this expectation is the Q-value itself. To incorporate the relation between preference vectors and Q-values for the multi-objective version of TD3, we include a directional angle term $g(\boldsymbol{\omega}, \mathbf{Q}(s, a, \boldsymbol{\omega}; \theta))$ to both actor's and critic's loss function. This directional angle term $g(\boldsymbol{\omega}, \mathbf{Q}(s, a, \boldsymbol{\omega}; \theta)) = \cos^{-1}(\frac{\boldsymbol{\omega}^T \mathbf{Q}(s,a,\boldsymbol{\omega};\theta)}{||\boldsymbol{\omega}_p|| \, ||\mathbf{Q}(s,a,\boldsymbol{\omega};\theta)||})$ denotes the angle between the preference vector $(\boldsymbol{\omega})$ and multi-objective Q-value $\mathbf{Q}(s, a, \boldsymbol{\omega}; \theta)$ and provides an alignment between preferences and the Q-values. Algorithm, network architectures, and implementation details are available in Appendix B.1.5 and B.3.

2.5 Experiments

This section extensively evaluates the proposed PD-MORL technique using commonly used MORL benchmarks with discrete state-action spaces (Section 2.5.1) and

complex MORL environments with continuous state-action spaces (Section 2.5.2). Detailed descriptions of these benchmarks are provided in Appendix B.2. Our results are compared to state-of-the-art techniques in terms of hypervolume, sparsity, and the Pareto front plots.

2.5.1 MORL Benchmarks with Discrete State-Action Spaces

This section illustrates the efficacy of PD-MORL in *obtaining a single universal network that covers the entire preference space* in discrete state-action environments. We use the following commonly used benchmarks to (i) make a fair comparison with prior work [1], and (ii) show that PD-MORL scales to more than two objectives:

Deep Sea Treasure (DST) has discrete state ($\mathcal{S} \subseteq \mathbb{N}^2$) and action spaces ($\mathcal{A} \subseteq \mathbb{N}^4$). It has two competing objectives: time penalty and treasure value. The treasure values increase as their distance from the starting point $s_0 = (0, 0)$ increases. Agent gets a -1 time penalty for every time step.

Fruit Tree Navigation (FTN) has discrete state ($\mathcal{S} \subseteq \mathbb{N}^2$) and action spaces ($\mathcal{A} \subseteq \mathbb{N}^2$) with six objectives: different nutrition facts of the fruits on the tree: {Protein, Carbs, Fats, Vitamins, Minerals, Water}. The goal of the agent is to find a path on the tree to collect the fruit that maximizes the nutrition facts for a given preference.

Figure 2.3 illustrates the policies achieved by PD-MORL for three preference vectors capture corner cases. Although our policy is *not specific* to any preference, the submarine achieves optimal treasure values for the given preference. For example, it finds the best trade-off within the shortest time one step) when we only value the time penalty ($w = \{0, 1\}$). As the importance of the treasure value increases

(w ={0.5,0.5}, $w = \{1, 0\}$), it spends optimal amount of time (8 steps and 19 steps, respectively) to find a deeper treasure. Figure 2.4 provides further insights into the progression of the policies found by PD-MORL. Let the first objective be the treasure value and the second objective be the time penalty in this figure. At the beginning of the training, each child process starts collecting transitions within their own preference sub-space $\tilde{\Omega}$. Since the agent acts ϵ-greedily at early phases and cannot reach the higher treasure values due to terminal condition (time limit), we observe that most of the solutions are stuck in the first-subspace ($\tilde{\Omega}_1$) The transitions collected from child processes are stored in the hindsight experience replay buffer with different preferences other than their original preferences. This buffer enables extra exploration for the agent and leads the agent to different sub-spaces, as illustrated in the middle figure. As the training progresses, HER and our novel optimality operator with the angle term guide the search to expand the Pareto front and cover the entire preference space with a single network. Among existing algorithms that learns a unified policy [1, 3], the Envelope algorithm [1] is the superior and a more recent approach. Hence, we compare and evaluate PD-MORL and the Envelope algorithm on DST and FTN by obtaining a set of preference

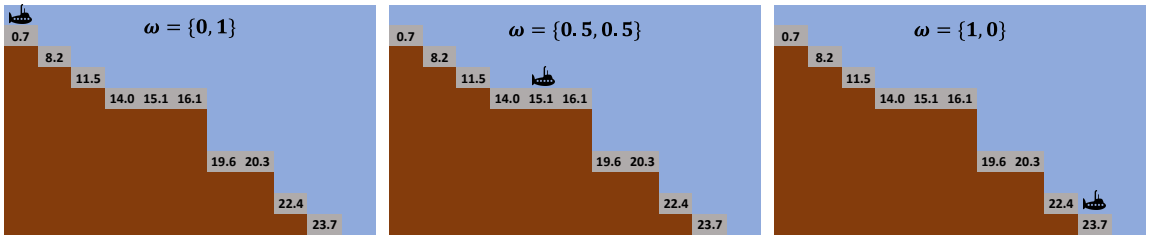


Figure 2.3: Optimal policies obtained by the trained agent for three corner preferences.

vectors that covers the entire preference space.

PD-MORL achieves both 6.1% larger hypervolume and 56% lower sparsity than the Envelope algorithm for DST problem, as shown in Table 2.1. Similarly, PD-MORL generates up to 78% larger hypervolume than the Envelope algorithm for FTN. Since FTN is a binary tree, the distance between Pareto solutions is the same, and hence, the sparsity metric is the same between the two approaches. Our experiments show that the improvement of PD-MORL compared to the Envelope algorithm increases with the problem complexity (i.e., increasing tree depth), as shown in Table 2.1. PD-MORL provides efficient and robust exploration using HER and our novel optimality operator together. Further comparisons against the Envelope algorithm [1] are provided in Appendix B.4.

2.5.2 MORL Benchmarks with Continuous Control Tasks

This section evaluates PD-MORL on popular multi-objective continuous control tasks based on MuJoCo physics engine [4, 44]. The environment details of these

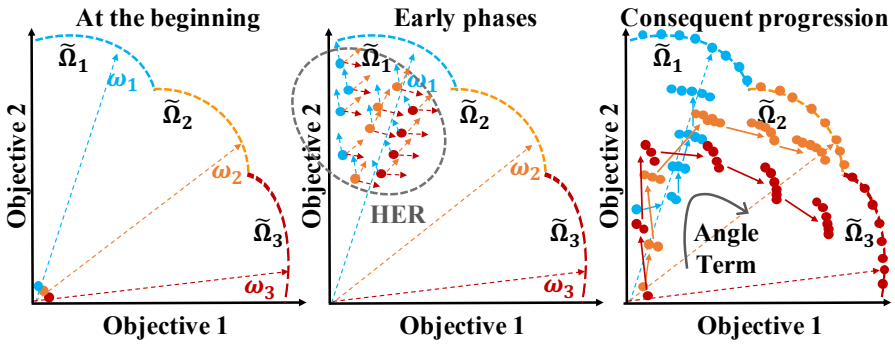


Figure 2.4: Policy progression during training.

Table 2.1: Comparison of our approach with prior work [1] using two simple MORL benchmarks in terms of hypervolume and sparsity metrics. The reference point for hypervolume calculation is set to (0,-19) and (0,0) for DST and FTN, respectively.

	Deep Sea Treasure		**Fruit Tree Navigation (d=6)**		**Fruit Tree Navigation (d=7)**	
	Hypervolume	**Sparsity**	**Hypervolume**	**Sparsity**	**Hypervolume**	**Sparsity**
Envelope [1]	227.89	2.62	8427.51	N/A	6395.27	N/A
PD-MORL	241.73	1.14	9299.15	N/A	11419.58	N/A

Table 2.2: Environment details of continuous control benchmarks.

	State Space	**Action Space**
MO-Walker2d-v2	$\mathcal{S} \subseteq \mathbb{R}^{17}$	$\mathcal{A} \subseteq \mathbb{R}^{6}$
MO-HalfCheetah-v2	$\mathcal{S} \subseteq \mathbb{R}^{17}$	$\mathcal{A} \subseteq \mathbb{R}^{6}$
MO-Ant-v2	$\mathcal{S} \subseteq \mathbb{R}^{27}$	$\mathcal{A} \subseteq \mathbb{R}^{8}$
MO-Swimmer-v2	$\mathcal{S} \subseteq \mathbb{R}^{8}$	$\mathcal{A} \subseteq \mathbb{R}^{2}$
MO-Hopper-v2	$\mathcal{S} \subseteq \mathbb{R}^{11}$	$\mathcal{A} \subseteq \mathbb{R}^{3}$

benchmarks are given in Table 2.2. The action and state spaces take continuous values and consist of multiple dimensions, making these benchmarks challenging to solve. The goal is to tune the amount of torque applied on the hinges/rotors for a given preference ω while satisfying multiple objectives (e.g., forward speed vs. energy efficiency, forward speed vs. jumping height, etc.). We compare the performance of PD-MORL against two state-of-the-art approaches [4, 40] that use these continuous control benchmarks. We emphasize that *these approaches must learn a different policy network for every solution in the Pareto front. In contrast, PD-MORL learns a single universal network that covers the entire preference space. The Envelope algorithm [1] is not included in these comparisons since it was not evaluated with continuous action spaces, and PD-MORL already outperforms it on simpler problems.* Training details are reported in Appendix B.3.

Table 2.3: Performance comparison of the proposed approach and state-of-the-art algorithms on the continuous control benchmarks in terms of hypervolume and sparsity metrics. Reference point for hypervolume calculation is set to (0,0) point. HV*: Hypervolume

	MO-Walker2d-v2		**MO-HalfCheetah-v2**		**MO-Ant-v2**		**MO-Swimmer-v2**		**MO-Hopper-v2**	
	HV*	**Sparsity**	**HV***	**Sparsity**	**HV***	**Sparsity**	**HV***	**Sparsity**	**HV***	**Sparsity**
PG-MORL [4]	4.82×10^6	0.04×10^4	5.77×10^6	$\mathbf{0.44 \times 10^3}$	6.35×10^6	$\mathbf{0.37 \times 10^4}$	2.57×10^4	9.9	$\mathbf{2.02 \times 10^7}$	0.5×10^4
META [40]	2.10×10^6	2.10×10^4	5.18×10^6	2.13×10^3	2.40×10^4	1.56×10^4	1.23×10^4	24.4	1.25×10^7	4.84×10^4
PD-MORL (Ours)	$\mathbf{5.41 \times 10^6}$	$\mathbf{0.03 \times 10^4}$	$\mathbf{5.89 \times 10^6}$	0.49×10^3	$\mathbf{7.48 \times 10^6}$	0.78×10^4	$\mathbf{3.21 \times 10^4}$	**5.7**	1.88×10^7	$\mathbf{0.3 \times 10^4}$

We first compare PD-MORL to prior work using hypervolume and sparsity metrics. Since META [40] and PG-MORL [4] report the average of six runs, we also ran each benchmark six times with PD-MORL. The average metrics are reported in Table 2.3, while the standard deviations and results of individual runs are given in Appendix B.4 Table B.5.

The desired Pareto front approximation should have high hypervolume and low sparsity metrics. PD-MORL outperforms the current state-of-the-art techniques both in terms of hypervolume and sparsity on every benchmark except MO-Hopper-v2, as summarized in Table 2.3. We emphasize that our *PD-MORL technique trains only a single network, while the other methods use customized policies network for each Pareto point.* For example, we achieve 12% higher hypervolume and 25% better sparsity than the most competitive prior work PG-MORL [4] on the Walker environment. They reported 263 Pareto front solutions for this environment. This corresponds to, in total, 2.8×10^6 trainable parameters since they have different policy network for each solution. In contrast, PD-MORL achieves better or comparable result using only 3.4×10^5 trainable parameters. This may enable the deployment of PD-MORL in a real-life scenario where there are dynamic

changes in the design constraints and goals. We also note that PG-MORL achieves better sparsity metric for MO-HalfCheetah-v2 and MO-Ant-v2 environments. However, PG-MORL also achieves a smaller hypervolume compared to PD-MORL. This suggests that to determine the quality of a Pareto front, these metrics are rather limited and should be further investigated by the existing literature. Therefore, we also plot Pareto front plots for all algorithms in Figure 2.5(a)-(e) to have a better understanding of the quality of the Pareto front. Figure 2.5 shows that the proposed PD-MORL technique achieves a broader and denser Pareto front than the state-of-the-art approaches despite using a single unified network. PD-MORL mostly relies on the interpolation procedure before the actual training. The key solutions obtained during this period determine the initial convergence of the algorithm. However, for MO-Hopper-v2, the obtained key solutions for preference vectors $\{1, 0\}, \{0.5, 0.5\}, \{0, 1\}$ are $\{1778, 4971\}, \{1227, 1310\}, \{3533, 3165\}$ respectively. These key solutions are not a good representative of the Pareto front shown in Figure 2.5(e). To have representative key solutions is a limitation of PD-MORL; however, it can be solved with hyperparameter tuning.

Finally, Figure 2.5(f) plots the progression of the hypervolume of the Pareto front for the environments with similar scales. The plots show that PD-MORL effectively pushes the hypervolume by discovering new Pareto solutions with the help of efficient and robust exploration. We ensure this by using HER, our novel directional angle term, and dividing the preference space into sub-spaces to collect transitions in parallel. The progression of the Pareto front, sparsity, and hypervolume of all benchmarks are provided in Appendix B.4.

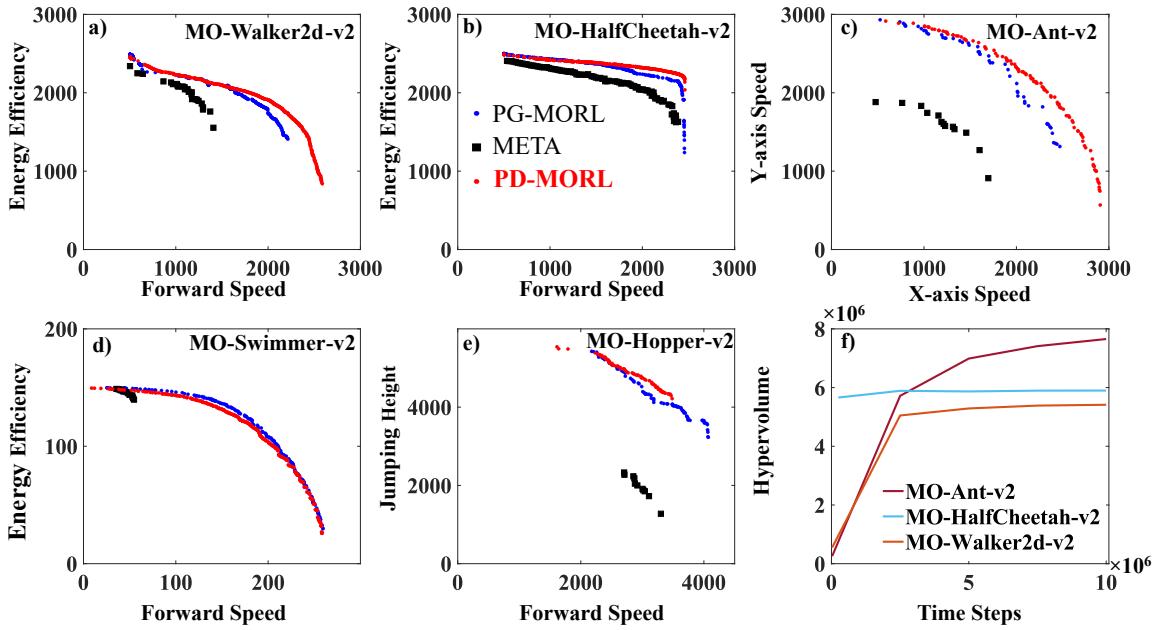


Figure 2.5: Pareto front comparison for continuous control tasks (a)-(e). Results for META and PG-MORL are obtained from [4] and code base of PG-MORL. (f) Progression of hypervolume of the Pareto front. Results for environments that have similar scales are given.

3 A COMPREHENSIVE MULTI-OBJECTIVE ENERGY MANAGEMENT APPROACH FOR WEARABLE DEVICES WITH DYNAMIC ENERGY DEMANDS

3.1 Motivation and Contributions

The emergence of small form-factor and low-cost wearable Internet of Things (IoT) devices enabled many novel edge-computing applications ranging from remote health monitoring to smart livestock monitoring systems [45, 46, 47, 48, 49, 50, 51]. To be practical, the devices that run these applications must operate within a tight energy budget (~μJ) and computational power due to limited battery capacity and small form factor [52, 53]. The small battery capacity limits the battery lifetime and requires frequent recharging, deteriorating the user experience. To mitigate this effect, energy harvesting (EH) from ambient sources, such as light, motion, electromagnetic waves, and body heat, has emerged as a promising solution to power these devices [54, 55].

Energy-neutral operation (ENO) can be achieved if the total energy consumed over a given period equals the energy harvested in the same period. One cannot rely on energy harvesting alone since the application performance and utilization of the device can tank in low EH conditions [56]. Therefore, energy management techniques are required to ensure ENO (i.e., the device maintains a certain battery level and minimum performance level) despite the variations in the harvested energy. These techniques use the available energy judiciously to maximize the

application performance while minimizing manual recharge interventions to tackle this challenge [57]. They must satisfy the following conditions to be deployed on a resource-constrained device:

1. Incur low execution time and power consumption overheads,
2. Have a small memory footprint,
3. Be responsive to dynamic changes in the environment,
4. Learn to adopt environmental and workload changes.

While most current energy management algorithms can satisfy criteria (1) and (2), they fail to meet the other two requirements. Specifically, they rely on fixed, non-adaptive predictive models of future harvested energy (3), and they do not incorporate an online/offline learning functionality to adopt dynamic changes in the environment (4). To the best of our knowledge, no energy management framework in the literature fulfills these four criteria.

While conventional optimization approaches may address the energy management problem by formulating it as a constrained optimization problem and utilizing solvers like CPLEX [58] and Gurobi [59], these methods face practical limitations when applied to resource-constrained wearable edge devices. These solvers typically require significant computational resources and impose lengthy execution times, rendering them unsuitable for online deployment scenarios where real-time decision-making is crucial [60]. As a powerful and practical alternative, we propose a reinforcement learning (RL) based energy management framework, tinyMAN, for

resource-constrained wearable edge devices. By leveraging RL, tinyMAN can adapt to dynamic changes in energy harvesting conditions and workload demands in real-time, allowing it to make informed decisions and maximize device utilization. The advantage of RL lies in its ability to learn optimal energy management strategies through trial and error, iteratively improving its performance over time. Instead of relying on pre-defined models of future energy harvesting values, tinyMAN learns to implicitly capture the complex dynamics of energy harvesting patterns and adapt its behavior accordingly. This adaptive nature of RL makes it particularly well-suited for environments with uncertain and fluctuating energy availability, such as wearable edge devices deployed in real-world settings. Furthermore, RL-based approaches, such as tinyMAN, offer scalability and flexibility, enabling them to accommodate diverse application requirements and environmental conditions. By learning directly from past experiences, tinyMAN can adapt to varying operating conditions and evolving user preferences, ensuring robust and efficient energy management in dynamic wearable edge computing environments.

At its core, tinyMAN takes the battery level and the previous harvested energy values as inputs (states) and maximizes the device's utility by judiciously allocating the harvested energy throughout the day (action). The utility is defined by an arbitrary concave function of allocated energy (e.g., $u = \log(x)$ or $u = \sqrt{x}$). To train tinyMAN, we employ the Twin Delayed Deep Deterministic Policy Gradient (TD3) Algorithm, a state-of-the-art RL algorithm for continuous action spaces [42]. Hence, the energy allocation values that tinyMAN yields can take continuous values according to the current energy availability. To this end, we first develop an

environment for the RL agent to interact with. This environment uses the light and motion EH modalities and *American Time Use Survey* [61] data from 4772 different users to model the dynamic changes in the harvested energy and battery. Over time, by interacting with the environment, the agent learns to manage the harvested energy on the device according to the battery energy level and the harvested energy. According to our evaluations, tinyMAN achieves up to 17% higher device utility than prior approaches while making 80% less battery constraint violations, such as not satisfying ENO or depleting battery prematurely.

The tinyMAN framework assumes the application running on the device is characterized by a fixed utility function with no energy demand by the application. This assumption is valid for continuous monitoring applications, such as heart rate monitoring, where the application operates within the allocated energy budget to produce utility. However, in actual use cases, the application can demand energy, and as a result, the utility function can be dynamic (i.e., a function of the application's demand). For example, a wearable device for activity recognition typically has higher demand during the day than at night. In light of this, we propose tinyMAN-MO, the multi-objective extension of the tinyMAN framework that learns the trade-off between meeting the application's demand and the battery energy level of the target device under varying energy harvesting conditions. To train tinyMAN-MO, we employ MO-TD3 [25], a multi-objective version of the TD3 algorithm, to obtain a single policy network that covers the *Pareto-front* of the objective space. It transforms the two-dimensional objective space into a single dimension by assigning weights (preferences) to each objective [29]. As a result, tinyMAN-MO

takes user-specified preference as a run-time input and yields the optimal energy allocations for the current energy harvesting and application demand conditions. Finally, we deploy both frameworks on a wearable device prototype and show that their execution time, energy consumption, and memory usage overhead are negligible compared to the daily energy harvesting capacity of any user. tinyMAN-MO operates within 10% of the Pareto-optimal solutions calculated using an offline solver with only 1.98 ms execution time and 23.17 µJ energy consumption overhead. *In summary, the major contributions of this work are as follows:*

- tinyMAN, a *prediction-free* RL based energy manager for resource-constrained wearable edge IoT devices,
- Evaluations that show tinyMAN achieves 17% higher device utilization than the state-of-the-art approaches while enabling ENO and 80% fewer battery constraint violations,
- tinyMAN-MO, the *multi-objective* extension of tinyMAN for energy management under varying application demands,
- Evaluations that show tinyMAN-MO operates within 10% of the Pareto-optimal solutions calculated using an offline solver by learning the tradeoff between meeting application demand and saving energy,
- Hardware implementations of both frameworks that demonstrate deployment on wearable devices with less than 100 KB memory footprint and 30 µJ energy consumption per inference.

3.2 Related Work

Energy-harvesting wearable IoT devices aim for energy-neutral operation to achieve self-sustainability. Recent research on energy harvesting and management focuses on optimizing the allocation of harvested energy to achieve self-sustainability and maximize device utilization [54, 62, 63, 64]. Kansal et al. [54], ensure ENO if the total energy consumed in a given period equals the harvested energy in the same period. The authors propose a linear programming approach to maximize the duty cycle of a sensor node and a lightweight heuristic to help solve the linear programming with ease. Although their approach is lightweight, it does not consider the application requirements when deciding the duty cycle of the nodes. Bhat et al. [62] address this issue by introducing a generalized utility function that takes into account the application characteristics and a lightweight framework based on a relaxed convex optimization that maximizes the utility while achieving self-sustainability. Their approach first determines the initial energy allocations for a given period based on the predicted EH values. Then it makes corrections to these allocations based on the difference between the predicted and actual EH values. However, the relaxation of one of the constraints in the optimization problem may lead to sub-optimal solutions. Tuncel et al. [57] propose a similar approach by employing a rollout technique without relaxing any constraints in the optimization problem to overcome the variations of harvested energy and user activities. Their approach also first decides initial energy allocations based on the expected EH values and makes adjustments to these allocations using the rollout algorithm at runtime. Hussein et al. [63] propose AdaEM, a two-stage approach to adapt to

uncertainties in EH values and user activities. In the first stage, they employ a supervised learning method to learn the distribution and uncertainty of EH values for a particular user based on their activities and location. The second stage involves solving a dynamic, robust optimization problem using the output from the first stage. However, the precision of activity prediction significantly impacts the quality of decisions in the second stage, while the first stage is typically tailored to a specific user or user group, potentially leading to suboptimal outcomes. Yamin and Bhat [65] propose a similar approach to AdaEM. It proposes a machine-learning approach to predict future solar energy availability for multiple intervals in the future, along with uncertainty bounds. Their proposed energy management algorithm then uses predicted EH values and uncertainty bounds to optimize energy allocation. Furthermore, both approaches rely on *predictive models* for future energy harvesting (EH) values. *Consequently, their effectiveness is closely tied to the accuracy of these predictions*.

Prediction-free approaches diverge from the reliance on forecasts of harvested energy, unlike the prediction-based approaches outlined above [64]. These meth-

Table 3.1: Comparison of the proposed tinyMAN framework with prior approaches

Approaches	Generalized Reward	Prediction Free	Deployable	Application Demand
[54]	✗	✗	✓	✗
[62]	✓	✗	✓	✗
[57]	✓	✗	✓	✗
[63]	✓	✗	✓	✗
[64]	✗	✓	✗	✗
tinyMAN	✓	✓	✓	✗
tinyMAN-MO	✓	✓	✓	✓

ods refrain from training or learning models to predict future energy harvesting values for optimizing energy allocations. RLMan is a recent prediction-free energy management approach based on reinforcement learning [64]. It aims to maximize the packet generation rate while avoiding power failures. Although it shows significant improvements in average packet rate, the reward function in RLMan focuses on maximizing the packet rate in a point-to-point communication system, which does not generalize to other performance metrics and ignores application requirements. In addition, the authors do not discuss the deployability of their framework on edge devices. In complement to the previous studies, we present tinyMAN, a prediction-free energy manager that uses a generalized reward function and is easily deployable on resource-constrained edge devices, as shown in Table 3.1. Both tinyMAN and the prior approaches consider a stationary utility function where the shape of this function does not change over time. However, many applications, such as activity trackers, demand energy during the hour of operation and thus, have utility functions that vary over time. To address this limitation, we propose tinyMAN-MO, a multi-objective extension that aims to optimize both meeting the varying energy demands of the application and maintaining a non-zero battery level.

3.3 Background

This section first introduces the battery energy dynamics and constraints to formulate the optimization problem. It also explains how various EH patterns are

obtained. Then, it describes the Twin Delayed Deep Deterministic Policy Gradient (TD3) algorithm used to train the tinyMAN RL agent.

3.3.1 Problem Formulation

Figure 3.1 illustrates a wearable device scenario in which our proposed tinyMAN framework is utilized. The proposed tinyMAN framework is deployed in an environment that consists of a wearable device, a flexible energy harvester, and a flexible battery, as depicted in Figure 3.1. This environment supports any arbitrary wearable applications facilitated by a target wearable device equipped with processing, sensing, and communication capabilities. This work uses a prototype wearable device as the target platform to deploy tinyMAN. This device incorporates a low-power MCU with communication capabilities, non-volatile RAM, and various sensors, including IMU, humidity, temperature, and light sensors. Additionally, it integrates components to utilize harvested energy effectively. A comprehensive list of the components used in the prototype wearable device is given in Table 3.2. In

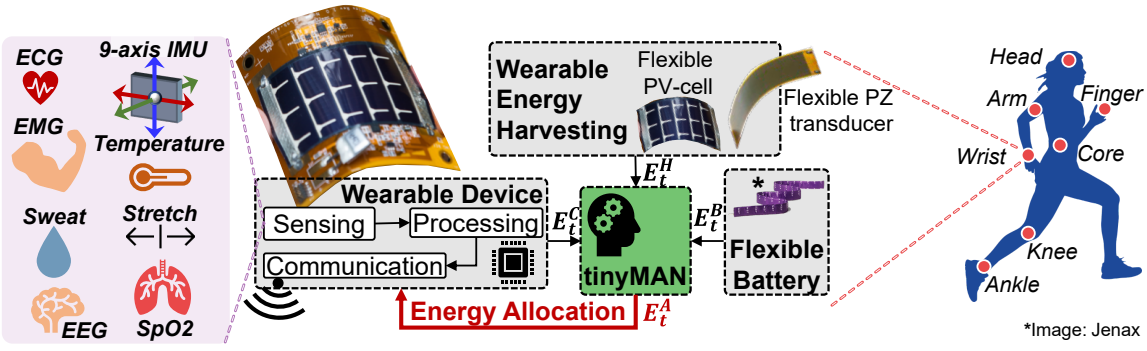


Figure 3.1: Illustration of the wearable device environment our proposed tinyMAN agent interacts with.

the following, we define the battery energy dynamics, the relevant constraints, and the utility function of the device and explain the EH source model.

Battery dynamics and constraints: tinyMAN maximizes the utilization of a target device under ENO and battery constraints by finding the optimum energy allocations. The assumed target device houses a flexible, small form-factor LiPo battery with a capacity of 12 mAh and can charge the battery through energy harvesting. Therefore, the battery energy dynamics in the environment is a function of:

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

Our energy management framework uses an episodic setting where each episode corresponds to a single day ($T = 24$ hours), and each step t in an episode lasts an hour. Using these definitions, we write the battery energy dynamics as follows:

$$E_{t+1}^B = E_t^B + \beta E_t^H - \alpha E_t^A,\ t \in T \tag{3.1}$$

where β corresponds to the efficiency of the harvester and α corresponds to the percent utilization of the allocated energy.

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

$$E_{cap}^B \geqslant E_t^B \geqslant E_{min}^B,\ \ t \in T \tag{3.2}$$

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

$$E_T^B \approx E_{target} \tag{3.3}$$

For achieving ENO, we set $E_{target} = E_0^B$ such that the battery energy level at the end of the episode is equal to the battery energy level at the beginning of the same episode. We enforce these constraints using the reward function as explained in Section 3.4.1.

Utility: The utility is a metric that represents the useful output produced by the device, such as accuracy or throughput, depending on the target application running on the device. For example, for a state-of-the-art heart rate monitoring application, the utility can be defined by the sampling frequency. The proposed tinyMAN framework supports any arbitrary utility function.

For the current work, we define the utility according to the minimum energy consumption of the device in an hour, which is calculated using the components listed in Table 3.2. The sum of the idle currents of these components amounts to 54.6 μA, resulting in an idle energy consumption of $E_{min}^A = 0.64$ J when the device operates at a VDD of 3.3V. Thus, if the allocated energy falls below this minimum threshold ($E_t^A < E_{min}^A$), the utility is zero (or negative), indicating that the device does not produce any useful output. However, it is important to note that the utility function can take on any shape based on the specific requirements of the application. In our study, we employed a logarithmic utility function with a diminishing return rate, as elaborated in Section 3.4.1.

Table 3.2: Components used in the prototype wearable device.

Component	**VDD**	$\mathbf{I}_{idle}$	**Part #**
Microcontroller	1.8-3.8V	0.9 μA	CC2652R
IMU	1.7-3.6V	8 μA	MPU9250
Nonvolatile Ram	1.6-3.6V	10 μA	MB85AS4MT
Humid. & Temp. Sensor	2.7-5.5V	0.1 μA	HDC1000
Ambient Light Sensor	1.6-3.6V	0.3 μA	OPT3001
Boost Converter for EH	2.5-5.2V	0.3 μA	BQ25504
LDO linear regulator	2.0-5.5V	35 μA	TLV702

EH Source: The EH source uses the dataset presented in [50] to generate EH scenarios according to different user patterns. This dataset uses light and motion energy modalities as ambient energy sources. It combines power consumption measurements with the activity and location information of 4772 users from the American Time Use Survey dataset [61] to generate varying 24-hour EH patterns per user. We divide the EH dataset [50] into four clusters according to the users' EH patterns throughout the day. The hourly distributions of these four clusters are illustrated in Figure 3.2. These distributions are based on the mean and the standard deviation of EH patterns in the same cluster. Therefore, the EH source generates a harvested energy value at every hour according to the distributions in the dataset as the day progresses.

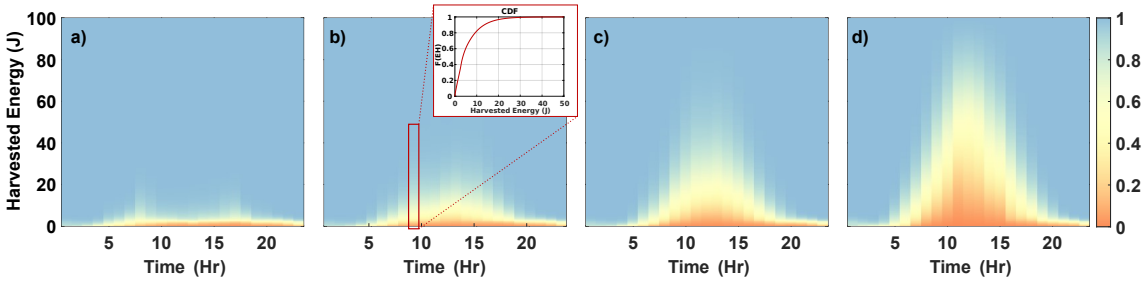


Figure 3.2: Hourly Cumulative distribution function of the harvested energy for a) Cluster 1, b) Cluster 2, c) Cluster 3, and d) Cluster 4. The inset graph shows the CDF of a specific hour, with the y-axis showing the CDF with respect to EH and the x-axis showing the EH values.

3.3.2 Twin Delayed Deep Deterministic Policy Gradient (TD3) Algorithm

The objective of an RL agent is to maximize the cumulative reward by interacting with the environment. According to the state s of the environment and the current policy π, the agent chooses an action a. Based on this action, the environment returns the next state s' and reward r. In this study, we adopt the Twin Delayed Deep Deterministic Policy Gradient [42] (TD3) algorithm, which extends the deep deterministic policy gradient (DDPG) [66] method due to its superior performance over state-of-the-art reinforcement learning (RL) approaches and its reduced sensitivity to hyperparameter tuning. The DDPG method consists of an actor that employs policy gradient methods to obtain the optimal policy and a critic that uses a deep Q network (DQN) to assess the actor's actions. In DDPG, the actor is trained to learn the correlation between state and action, whereas the critic is trained to establish the relationship between state-action pairs and the expected cumulative returns (Q-values). However, one limitation of DDPG is that the critic is prone to

overestimating the target Q-value. This can lead to challenges with policy stability and convergence to local optima due to the approximation errors in the Q-value that are used to enhance the policy [42]. To address the overestimation error, TD3 [42] offers three significant improvements over DDPG: (i) clipped double Q-learning, (ii) target policy smoothing, and (iii) delayed policy updates.

Clipped double Q-learning: TD3 adopts two critic networks instead of one, which enables it to use the lower Q-value of the two to generate the targets in Bellman's optimality equation. Both critic networks are updated by TD3 using the following loss:

$$L_{critic}(\theta_i) = \mathbb{E}_{(s_t,a_t,r_t,s_{t+1})\sim\mathcal{D}}\left[\left(y - Q(s_t, a_t; \theta_i)\right)^2\right] \tag{3.4}$$

$$y = r + \gamma \arg_Q \min_{i=1,2} Q(s_{t+1}, \tilde{a}; \theta'_i) \tag{3.5}$$

where $\mathcal{D}$ is the experience replay buffer that stores transitions (s_t, a_t, r_t, s_{t+1}) for every time step in the environment, y is the target value, and γ is the discount factor.

Target policy smoothing: When calculating target values, the target actor-network generates the action $\tilde{a}$. However, the DDPG approach can lead to target values with high variance, even for similar actions, because deterministic policies are susceptible to overfitting to sharp peaks in the value estimate. To mitigate this issue, the TD3 algorithm introduces some noise ϵ to the action instead of using the action

given directly by the target actor-network, as follows:

$$\tilde{a} = \pi(s_{t+1}, \phi') + \epsilon : \epsilon \sim clip(\mathcal{N}(0, \sigma), -c, c) \tag{3.6}$$

The policy of the target actor-network for state s_{t+1} is denoted by $\pi(s_{t+1}, \phi')$, and the range of $-c$ to $+c$ denotes that any added noise is bounded to ensure that the target action stays close to the actual action. This regularization technique helps to decrease the variance in the target values.

Delayed policy updates: Deterministic policy gradient methods update the parameters of the actor-network by maximizing the Q-values, which are obtained utilizing the actions created by the actor-network as follows:

$$L_{actor}(\phi) = \mathbb{E}_{(s_t, a_t, r_t, s_{t+1})\sim\mathcal{D}}\Big[Q(s_t, a_t; \theta_1)|_{a_t=\pi(s_t;\phi)}\Big] \tag{3.7}$$

This update rule can result in divergence during the agent's training, particularly when a suboptimal policy is overestimated. As a result, the agent may update on states with high error, leading to instability and a shift towards suboptimal policies. To address this issue, TD3 adopts a less frequent update schedule for the actor-network compared to the value network. This approach of infrequent policy updates generates value estimates with lower variance and therefore facilitates the development of better policies.

3.4 Proposed Energy Manager – tinyMAN

This section provides the environment dynamics and introduces the tinyMAN RL framework.

3.4.1 Environment Dynamics

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

State Space: The state is a 6-tuple that consists of:

- **Current battery energy** ($\frac{E_t^B}{E_{max}^B} \in [0,1]$)**:** The energy level of the battery at the beginning of the current step t divided by the battery capacity.
- **Harvested energy in the previous time step** ($\frac{E_{t-1}^H}{E_{max}^B} \in [0,1]$)**:** Harvested energy during the previous step $t-1$ divided by the battery capacity.
- **Time** ($t \in \mathbb{Z}$)**:** The current time step t, corresponding to the current hour of the day.
- **Initial battery energy level** ($\frac{E_0^B}{E_{max}^B} \in [0,1]$)**:** The energy level of the battery at the beginning of the episode $(t{=}0)$ divided by the battery capacity.
- **Cumulative E^H** ($\sum_{\tau=0}^{t-1} \frac{E_\tau^H}{E_{max}^B} \in \mathbb{R}$)**:** Cumulative harvested energy in the previous time steps divided by the battery capacity.
- **Cumulative E^A** ($\sum_{\tau=0}^{t-1} \frac{E_\tau^A}{E_{max}^B}$)**:** Cumulative energy allocations in the previous time steps divided by the battery capacity.

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

Reward function: Our objective is to maximize the utility of the device under certain constraints on the battery energy level. Although tinyMAN can support any utility function, for a fair comparison with previous literature [62, 57], we utilize a logarithmic utility function in this work:

$$u(E_t^A) = \ln\left(\frac{E_t^A}{E_{min}^A}\right) \tag{3.8}$$

Given that we evaluate performance based on average utility, using a different utility function would introduce bias and hinder meaningful comparisons across studies.

In an RL setting, the reward function plays a pivotal role in imposing constraints on the battery. There are two constraints that can be imposed on the reward function: (*i*) emergency reservoir energy constraint (Equation 3.2) and (*ii*) ENO constraint (Equation 3.3). Considering the objective and the constraints on the battery, we define the reward function as a piecewise function based on both the battery energy

level and the time of day:

$$r_t = \begin{cases} u(E_t^A) & E_t^B \geqslant E_{min}^B \text{ and } t \neq T \\ -|E_t^B - E_{min}^B| & E_t^B < E_{min}^B \text{ and } t \neq T \\ -|E_t^B - E_{target}| & E_t^B \leqslant 0 \text{ and } t \neq T \\ -|E_t^B - E_{target}| & t = T \end{cases} \tag{3.9}$$

Under the first condition, where the battery energy level complies with the constraints and the time t is not equal to T (the end of an episode/day), the objective is to maximize utility. If the battery energy level falls below the emergency reservoir threshold and t is not equal to T, we enforce the emergency reservoir constraint by penalizing the agent using the term $-|E_t^B - E_{min}^B|$. In the event of battery depletion, and t is not equal to T, we heavily penalize the agent using the term $-|E_t^B - E_{target}|$. Finally, if the battery energy level remains within the constraints, t reaches T, we apply the ENO constraint by penalizing the agent using the term $-|E_t^B - E_{target}|$. An episode terminates if time T is reached or the battery is completely drained.

According to the environment dynamics explained in this section, we develop our environment in Python and register it as an OpenAI's Gym [67] environment.

3.4.2 The Proposed tinyMan RL Framework

Algorithm 2 summarizes the training of the tinyMAN agent for a given cluster of users. First, we initialize an empty replay buffer $\mathcal{D}$ and critic and actor networks with random weights (line 5). We also initialize the target critic and actor networks

Algorithm 2: tinyMAN - RL based Energy Manager

1 **Input:** Number of total time steps N, Minibatch size $\mathcal{B}$, Discount factor γ, Policy update delay p_{delay},
2 The standard deviation for smoothing noise added to target policy σ,
3 The standard deviation for Gaussian exploration noise added to the policy σ',
4 The clipping factor of the target policy smoothing noise c.
5 **Initialize:** Replay buffer $\mathcal{D}$, Critic networks $\mathbf{Q}_{\theta_1}$,$\mathbf{Q}_{\theta_2}$ and actor-network π_ϕ with parameters θ_1, θ_2, ϕ.
6 Target networks $\mathbf{Q}_{\theta_1'} \leftarrow \mathbf{Q}_{\theta_1}$, $\mathbf{Q}_{\theta_2'} \leftarrow \mathbf{Q}_{\theta_2}$, $\pi_{\phi'} \leftarrow \pi_\phi$
7 $done \leftarrow$ True
8 Parameter update count, $p_n = 0$.
9 **while** *Total Number of Steps* $<$ N **do**
10 **if** $done =$ True **then**
11 # End of an episode, start a new one
12 Reset the environment to a randomly chosen initial battery energy E_0^B and an EH pattern.
13 $done \leftarrow$ False
14 Choose an action with exploration noise $a_t \sim \pi(s_t;\phi) + \epsilon : \epsilon \sim \mathcal{N}(0,\sigma')$
15 Collect samples $\{s_t, a_t, r_t, s_t', done\}$ by interacting with the environment and store them in $\mathcal{D}$.
16 Sample random $\mathcal{B}$ transitions from $\mathcal{D}$.
17 $\tilde{a}_t \leftarrow \pi(s_{t+1},\phi') + \epsilon : \epsilon \sim clip(\mathcal{N}(0,\sigma), -c, c)$
18 $y \leftarrow r + \gamma \arg_Q \min_{i=1,2} Q_{\theta_i'}(s_{t+1}, \tilde{a}_t)$
19 $Loss_{critic}(\theta_i) = \mathbb{E}\Big[\Big(y - Q_{\theta_i}(s_t,a_t)\Big)^2\Big]$
20 Update critic network parameters.
21 $p_n = p_n + 1$
22 **if** $p_n \bmod p_{delay}$ **then**
23 $Loss_{actor}(\phi) = \mathbb{E}\Big[\, Q_{\theta_1}(s_t,a_t)|_{a_t=\pi(s_t;\phi)}\Big]$
24 Update actor parameters and target network parameters.

with the same parameters (line 6). The networks consist of fully connected layers with three hidden layers and 64 neurons in each layer. At the start of each episode n, we randomly choose an EH pattern and an initial battery energy level (line 12). The EH pattern is generated using the hourly distributions depicted in Figure 3.2 and is

unique for each episode, allowing tinyMAN to learn the EH patterns of users in that cluster. The agent then interacts with the environment using its actor-network (π_ϕ) (line 14). The resulting transitions $(s_t, a_t, r_t, s_{t+1}, done)$ are then stored in the experience replay buffer $\mathcal{D}$ (line 15). The algorithm then samples a minibatch ($\mathcal{B}$) of transitions from $\mathcal{D}$ (line 16). Using this minibatch of samples, the smoothed target action values are determined based on the noise ϵ and the clipping factor c (line 17). The target value y is computed using the target critic-networks and target action values (line 18). Finally, the actor and critic networks are updated using the loss functions described in Section 3.3.2 (lines 19-24). The training concludes once the total number of time steps reaches N.

The EH dataset [50] is divided into four clusters according to the users' EH patterns throughout the day. The agent is trained separately on each cluster. We implement tinyMAN in Python using Adam optimizer with a learning rate of 1E-4. Since the proposed tinyMAN framework is deployed on a wearable device, the characteristics of the target device, such as battery capacity, minimum battery energy level (E^B_{min}), and minimum energy allocation (E^A_{min}) should be identified before the training. These characteristics do not change over time during the training. We set the emergency reservoir energy as $E^B_{min} = 10$ J, roughly corresponding to 5 minutes of active time for the components listed in Table 3.2. This parameter and others, as listed in Table 3.3, can be tailored according to the requirements of another device or application. The hyperparameters listed in Table 3.3 are selected based on extensive experimentation and parameter tuning centered around the original values and discussion presented in the original paper for TD3 algorithm [42].

Table 3.3: Definition of the hyperparameters for tinyMAN and their values

Parameter	Description	Value	Parameter	Description	Value
N	Total number of time steps	1×10^6	σ	Target policy smoothing noise std.	0.2
$\mathcal{B}$	Minibatch size	100	c	Clipping factor of target policy noise	0.5
$\mathbb{D}$	Replay buffer size	1×10^5	N_{Layer}	Number of hidden layers	3
γ	Discount factor	0.995	N_{Neuron}	Number of hidden neurons	64
p_{delay}	Policy update delay	2	lr	Learning rate	1×10^{-4}
σ'	Exploration noise std.	0.3	E^B_{min}	Emergency reservoir energy	10

3.5 Addressing Varying Demand by the Application – tinyMAN-MO

So far, the proposed tinyMAN framework has a single objective; to maximize the utility under ENO constraints. It also considers a stationary utility function, i.e., the shape of the utility function does not change over the horizon. This assumption holds for many continuous monitoring applications, such as heart or respiration rate monitoring, where the application operates within the allocated energy budget without making explicit demands. In contrast, many other applications, such as sleep trackers, pedometers, or activity trackers, can demand energy and have utility functions that vary over time. For instance, a workout tracker demands high energy and produces utility when the user is active (i.e., working out or playing sports). Therefore, meeting the varying demands is critical for many applications. Motivated by these observations, we present tinyMAN-MO, the multi-objective extension of tinyMAN, to co-optimize, meeting the application's varying demand and maintaining a non-zero battery level.

3.5.1 Multi-objective Problem Formulation

Introducing application demand as a new constraint adds complexity, as it contradicts the existing ENO constraints presented in Section 3.3.1. For instance, if the total energy demand of the application surpasses the harvested energy within the time horizon T, achieving energy neutrality becomes unfeasible. Therefore, we reformulate the problem into a multi-objective problem with two conflicting objectives: meeting the application's demand and sustaining a non-zero battery energy level. For this, we need to define the application's energy demand in our work to utilize it in the objective function.

Energy Demand: This study focuses on an activity tracker application as a representative case study. However, it is important to note that the approach proposed here extends beyond this particular application and applies to various applications. Activity trackers typically experience higher energy demands during user activity. To model this behavior, we generate variable demand curves based on individual user activity data sourced from the American Time Use Survey dataset [50]. This dataset consists of 10 activity labels [57], which we categorize into three levels: *Active* task, *Daily* task, *Not active* task. For instance, activities like "exercise" fall under *Active* tasks while "personal care" and "work" fall under *Daily* task, and "sleep" and "eat" falls under *Not active* task. Using this categorization, we derive daily activity level patterns for 4772 users at a 1-minute resolution, as depicted for 50 randomly selected users in Figure 3.3-a) using a lasagna plot. As anticipated, periods of low activity, such as the early and late hours of the day and lunchtime, are predominantly categorized as *Not active*. Following this analysis, we refine

our modeling approach by incorporating energy consumption data from three commercial products:

i. *Not active*: 50 mJ/min – Oura Ring [68]

ii. *Daily*: 140 mJ/min – Amazon Halo [69]

iii. *Active*: 480 mJ/min – Whoop Strap [70]

we convert the activity level information into the application's hourly energy demand. For instance, Figure 3.3-b) displays the computed hourly demand for a randomly selected user from Figure 3.3-a). This user exhibits peak activity around midday, resulting in elevated demand during that period. Conversely, the resting

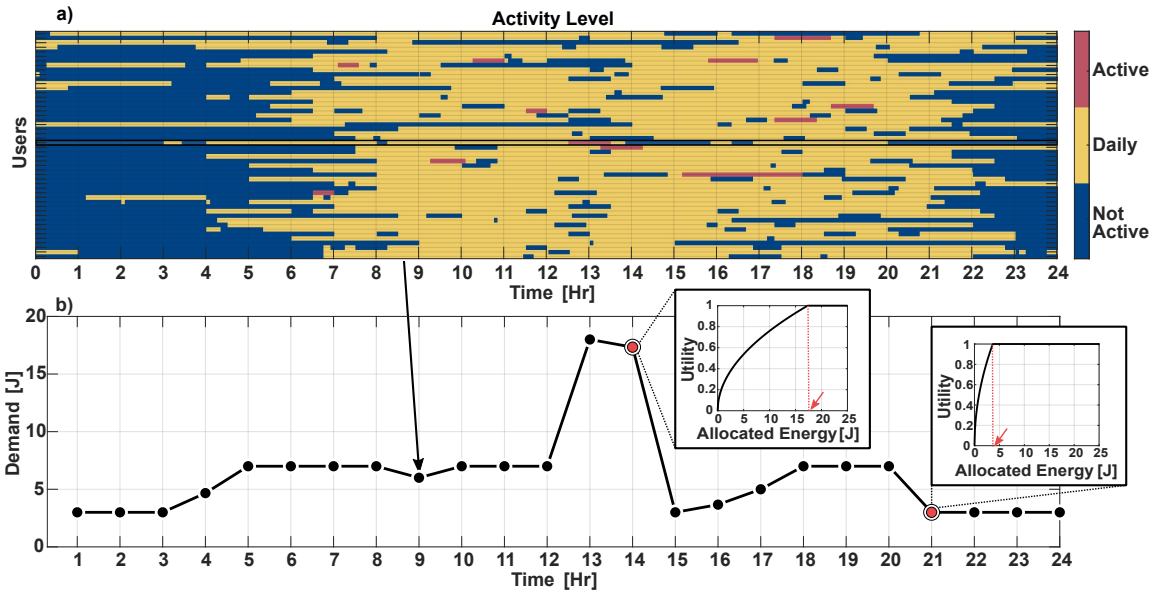


Figure 3.3: a) Activity levels of randomly selected 50 users in the dataset. Each row is a user, each box in the grid corresponds to 60 minutes, and colors correspond to activity levels. b) The demand for one of the users is highlighted by the black box. The insets show the different utility curves due to different demands.

and eating phases following the active period correspond to decreased application demand. Using this dataset, we calculate the hourly energy demand for each user and integrate it into our revised utility function for the multi-objective problem, as outlined in the following.

Utility: We revise the utility function in Section 3.3.1 to incorporate the demand:

$$u(E_t^A, D_t) = \begin{cases} \sqrt{\frac{E_t^A}{D_t}} & E_t^A < D_t \\ 1 & E_t^A \geqslant D_t \end{cases} \tag{3.10}$$

where D_t denotes the demand at time step t. In this case, the utility function equals 1 when $E_t^A \geqslant D_t$, which means *the demand is met*, and allocating more energy does not produce additional utility. Moreover, we use *square-root* instead of *log* to have a slower rate of decrease in utility when $E_t^A < D_t$. We keep the function's concavity, not to dismiss the ability to solve it optimally using convex optimization solvers. If this was not a concern, the utility could be set to zero for $E_t^A < E_{min}^A$, similar to Section 3.3.1. We emphasize that tinyMAN-MO supports any arbitrary utility function and that the utility function can have any shape according to the needs of the application. As a result, we obtain a time-varying utility function characterized by the application's demand. For instance, the insets in Figure 3.3-b) depict two different cases for the utility function. At $t = 14$, the application's demand is 17.33 J, and the utility is 1 if that amount of energy is allocated to the application. Similarly, at $t = 21$, the application demands 3 J to yield a utility of 1. With this revised utility function, *meeting the application's demand is equivalent to maximizing the utility*.

3.5.2 Multi-objective Environment Dynamics

tinyMAN-MO's main objective is to learn the trade-off between meeting the application's varying demand and the battery energy level. We revise the environment dynamics explained in Section 3.4.1 to reflect this change.

State Space: We add the demand for the current time step to the state vector, making it a 7-tuple:

- **Demand** ($\frac{D_t}{E^B_{max}} \in [0, 1]$)**:** Demand at the current time step t divided by the battery capacity.

Action Space: The action is identical to tinyMAN and corresponds to the allocated energy at every time step ($E^A_t \in [E^A_{min}, E^B_t]$).

Reward function: The multi-objective extension incorporates two conflicting goals: meeting the application's demand and maintaining a non-zero battery energy level. Based on the revised utility function, we express meeting the application's demand as maximizing the utility. Thus, the agent aims to learn the balance between maximizing the utility function (given by Equation 3.10) and maximizing the battery energy level while adhering to specific battery-related constraints. As the revised utility function produces values between 0 and 1, we use the percent battery energy level as the second objective to ensure compatible scales between the two objectives. Consequently, the reward function now produces a vector of

two elements as follows:

$$r_t = \begin{cases} [u(E_t^A, D_t), \frac{E_t^B}{E_{max}^B}] & E_{min}^B \leqslant E_t^B \leqslant E_{max}^B \\ [u(E_t^A, D_t), -\frac{E_t^B}{E_{max}^B} + 1] & E_t^B > E_{max}^B \\ [-|E_t^B - E_{min}^B|, -|E_t^B - E_{min}^B|] & E_t^B < E_{min}^B \end{cases} \tag{3.11}$$

In both objectives, the emergency reservoir energy constraint is incorporated by including the term $-|E_t^B - E_{min}^B|$. The reward vector for this condition has a scale of an order of magnitude higher than the first two conditions. This implies that if the battery level falls below a minimum level ($E_t^B \leqslant E_{min}^B$) during the episode, the agent will be heavily penalized. Additionally, a negative reward is given for the battery level objective if it exceeds the maximum battery capacity to ensure that saving more energy than the battery capacity is not advantageous. An episode terminates when either time $T = 24$ is reached or the battery is entirely depleted.

3.5.3 Multi-objective RL Framework

Existing multi-objective reinforcement learning (MORL) methods transform the multidimensional objective space into a single dimension using static weights for each objective and then employ standard RL algorithms to obtain a policy optimized for those weights [29]. Repetitively solving a multi-objective optimization problem by scalarizing the multidimensional objective space is not feasible, and it requires domain expertise in order to assign preferences (ω) to each objective statically. Recent MORL approaches suggest training a single policy network that covers

the entire preference space [4, 1, 25]. The goal is to obtain a policy that yields a solution on the Pareto front for a given preference. The Pareto front is a set of solutions that are non-dominated by each other but are superior to the rest of the solutions in the solution space. To accomplish this, we utilize a multi-objective version of TD3 (MO-TD3) [25] to obtain a single policy network that covers the entire preference space. MO-TD3 is an extension of the TD3 algorithm [42] where the network also takes preferences ($\boldsymbol{\omega}$) of the objectives as inputs along with the states. It also employs Hindsight Experience Replay buffer [31] to efficiently explore the preference space and novel parallelization approach to increase the sample efficiency of the algorithm [25]. This technique is explained in detail in Section 2.

Since our multi-objective environment returns a reward vector, the Q-network is also vectorized to efficiently learn to model multiple objectives for a given preference vector $\boldsymbol{\omega}$. Specifically, the network takes state s and preference vector $\boldsymbol{\omega}$ as inputs and outputs a vector of Q-values. Hence, the update rule of the vectorized network is modified as follows:

$$L_{critic}(\theta_i) = \mathbb{E}_{(s_t, a_t, \mathbf{r}_t, s_{t+1}, \boldsymbol{\omega}) \sim \mathcal{D}} \left[\left(y - \mathbf{Q}(s_t, a_t, \boldsymbol{\omega}; \theta_i) \right)^2 \right] \tag{3.12}$$

$$\mathbf{y} = \mathbf{r} + \gamma \arg_{\mathbf{Q}} \min_{i=1,2} \mathbf{Q}(s_{t+1}, \tilde{a}, \boldsymbol{\omega}; \theta'_i) \tag{3.13}$$

The actor-network also takes the preference as inputs, and the parameters of this network are updated by maximizing the $\boldsymbol{\omega}^T \mathbf{Q}$ using the following loss:

$$L_{actor}(\phi) = \mathbb{E}_{(s_t, a_t, \mathbf{r}_t, s_{t+1}, \boldsymbol{\omega}) \sim \mathcal{D}} \Big[\mathbf{Q}(s_t, a_t, \boldsymbol{\omega}; \theta_1)|_{a_t = \pi(s_t, \boldsymbol{\omega}; \phi)} \Big] \tag{3.14}$$

Similar to tinyMAN, we first identify the characteristics of the target device: the minimum battery energy level as $E^B_{min} = 1$ J to allow the tinyMAN-MO agent to cover the corner cases where the battery is drained and the minimum energy allocation as $E^A_{min} = 0.64$ J. The agent is trained separately on each of the four clusters. Algorithm 3 outlines the training of the tinyMAN-MO agent. Different from tinyMAN, at the beginning of each episode during training, we randomly sample a preference vector ($\boldsymbol{\omega} \in \Omega : \sum_{i=0}^{L} \omega_i = 1$) from a uniform distribution (line 13). For a given preference, the agent interacts with the environment using its actor-network (π_ϕ) and obtains transitions ($s_t, a_t, \mathbf{r}_t, s_{t+1}, \boldsymbol{\omega}, done$), which are then stored in the experience replay buffer $\mathcal{D}$ (lines 15-16). To explore the preference space, we also store N_ω transitions for each transition, where a different preference ($\boldsymbol{\omega}' \in \Omega : \sum_{i=0}^{L} \omega'_i = 1$) is sampled. These additional transitions ($s, a, \mathbf{r}, s', \boldsymbol{\omega}', done$) are also stored in $\mathcal{D}$ (line 17-18). The remaining steps are similar to tinyMAN, except that the critic networks yield vectorized Q-values and that the actor and critic networks are updated using the loss functions described in this section (lines 19-27). The training ends after the total number of time steps reaches N. The steps that differ from the tinyMAN implementation (Algorithm 2) are highlighted with teal in Algorithm 3.

We implement tinyMAN-MO in Python using Adam optimizer with a learning rate of 1E-4. The hyperparameters for tinyMAN-MO are presented in Table 3.4. The

hyperparameters listed in Table 3.4 are selected based on extensive experimentation and parameter tuning centered around the original values and discussion presented in the original paper for TD3 algorithm [42].

Algorithm 3: tinyMAN-MO - Multi-objective RL based Energy Manager

Input: Number of total time steps N, Minibatch size $\mathcal{B}$, Discount factor γ, Policy update delay p_{delay},
The standard deviation for smoothing noise added to target policy σ,
The standard deviation for Gaussian exploration noise added to the policy σ',
The clipping factor of the target policy smoothing noise c, Number of preferences sampled for HER N_ω.
Initialize: Replay buffer $\mathcal{D}$, Critic networks $\mathbf{Q}_{\theta_1}$,$\mathbf{Q}_{\theta_2}$ and actor-network π_ϕ with parameters θ_1, θ_2, ϕ.
Target networks $\mathbf{Q}_{\theta_1'} \leftarrow \mathbf{Q}_{\theta_1}$, $\mathbf{Q}_{\theta_2'} \leftarrow \mathbf{Q}_{\theta_2}$, $\pi_{\phi'} \leftarrow \pi_\phi$
$done \leftarrow$ True
Parameter update count, $p_n = 0$.
while *Total Number of Steps* $< N$ **do**
    if $done =$ True **then**
        # End of an episode, start a new one
        Reset the environment to a randomly chosen initial battery energy E_0^B and an EH pattern.
        Sample a preference vector $\boldsymbol{\omega}$.
        $done \leftarrow$ False
    Choose an action with exploration noise $a_t \sim \pi(s_t, \boldsymbol{\omega}; \phi) + \epsilon : \epsilon \sim \mathcal{N}(0, \sigma')$
    Collect samples $\{s_t, a_t, \mathbf{r}_t, s_t', \boldsymbol{\omega}, done\}$ by interacting with the environment and store them in $\mathcal{D}$.
    for $j = 1: N_\omega$ **do**
        Store transition $(s_t, a_t, \mathbf{r}_t, s', \boldsymbol{\omega}_j', done)$ in $\mathcal{D}$
    Sample random $\mathcal{B}$ transitions from $\mathcal{D}$.
    $\tilde{a}_t \leftarrow \pi(s_{t+1}, \boldsymbol{\omega}; \phi') + \epsilon : \epsilon \sim clip(\mathcal{N}(0, \sigma), -c, c)$
    $\mathbf{y} \leftarrow \mathbf{r} + \gamma \arg_{\mathbf{Q}} \min_{i=1,2} \mathbf{Q}_{\theta_i'}(s_{t+1}, \tilde{a}_t, \boldsymbol{\omega})$
    $Loss_{critic}(\theta_i) = \mathbb{E}\Big[\Big(\mathbf{y} - \mathbf{Q}_{\theta_i}(s_t, a_t, \boldsymbol{\omega})\Big)^2\Big]$
    Update critic network parameters.
    $p_n = p_n + 1$
    if $p_n \bmod p_{delay}$ **then**
        $Loss_{actor}(\phi) = \mathbb{E}\Big[\mathbf{Q}_{\theta_1}(s_t, a_t, \boldsymbol{\omega})|_{a_t = \pi(s_t, \boldsymbol{\omega}; \phi)}\Big]$
        Update actor parameters and target network parameters.

Table 3.4: Definition of the hyperparameters for tinyMAN-MO and their values

Parameter	Description	Value	Parameter	Description	Value
N	Total number of time steps	3×10^6	σ'	Exploration noise std.	0.1
$\mathcal{B}$	Minibatch size	128	σ	Target policy smoothing noise std.	0.2
$\mathbb{D}$	Replay buffer size	5×10^5	c	Clipping factor of target policy noise	0.5
γ	Discount factor	0.995	N_{Layer}	Number of hidden layers	3
N_ω	Number of preferences-HER	3	N_{Neuron}	Number of hidden neurons	64
p_{delay}	Policy update delay	10	lr	Learning Rate	1×10^{-4}

3.6 Experimental Evaluations

This section evaluates the tinyMAN framework, presenting results for both single-objective and multi-objective cases. We provide execution time, energy overhead, and memory footprint measurements of the tinyMAN framework when deployed on a wearable device prototype in Section 3.7.

3.6.1 Single-objective Results

3.6.1.1 Experimental Setup

We compare the performance of tinyMAN to three prediction-based approaches in the literature [62, 54, 57], as discussed in Section 3.2, as well as to an oracle solution. The oracle solution is obtained by an offline solver (e.g., CVX) with the actual harvested energy data for the day. It solves the following optimization problem for a given preference and finite horizon:

$$\begin{aligned} &\text{maximize} \quad \sum_{t=0}^{T-1} \gamma^t u(E_t^A) \\ &\text{subject to} \quad E_t^A > E_{min}^A \text{ and } E_{min}^B \leqslant E_t^B \leqslant E_{max}^B \end{aligned} \tag{3.15}$$

where γ is the discount factor. We emphasize that this oracle solution is inherently unfair and unrealistic. However, it serves as a reference point, offering insight into the maximum theoretical utility achievable given the harvested energy.

For each cluster, we begin by sorting all users based on their cumulative daily harvested energy. Subsequently, we exclude users corresponding to the {25%, 50%, 75%} of this sorted list from the training set for illustrative purposes. Additionally, 10% of the remaining users are randomly selected as test users and are also removed from the training data for evaluation purposes. Furthermore, we assess the performance of our approach for each cluster at four different initial battery energy levels: $E_0^B = \{16, 48, 112, 144\}$J.

3.6.1.2 Performance Evaluation

Figure 3.4 provides a summary of the average daily utility and the total number of constraint violations obtained for all approaches using the test users for each cluster. In general, we observe an increase in total utility as the harvested energy increases from cluster 1 to cluster 4, given the greater energy available for allocation on the device. *tinyMAN consistently maintains utility levels within 10% of the optimal utility. Moreover, it results in at least 20% fewer constraint violations compared to prior approaches*, as illustrated in Figure 3.4-b). Specifically, in scenarios with low battery energy levels (16J), tinyMAN achieves 80% fewer constraint violations. This performance is attributed to the constraints enforced in the reward function, where the agent incurs heavy penalties for violations. By exploring various energy harvesting and battery energy conditions during training, tinyMAN can make more informed decisions,

leading to fewer constraint violations. However, in Figure 3.4a), for cluster-1, we observe an unexpected increase in the total number of violations with the increasing battery level. This phenomenon may appear counterintuitive, considering that one might expect a higher battery level to lead to better performance in terms of constraint violations. One plausible explanation for this behavior is that with a higher battery level, the agent becomes more exploratory or aggressive in its energy allocation decisions, leading to riskier actions that result in more constraint violations, particularly under conditions of low energy harvesting, such as those observed in cluster 1. We also observe that tinyMAN only marginally violates the ENO constraint in scenarios with high battery energy levels. Furthermore,

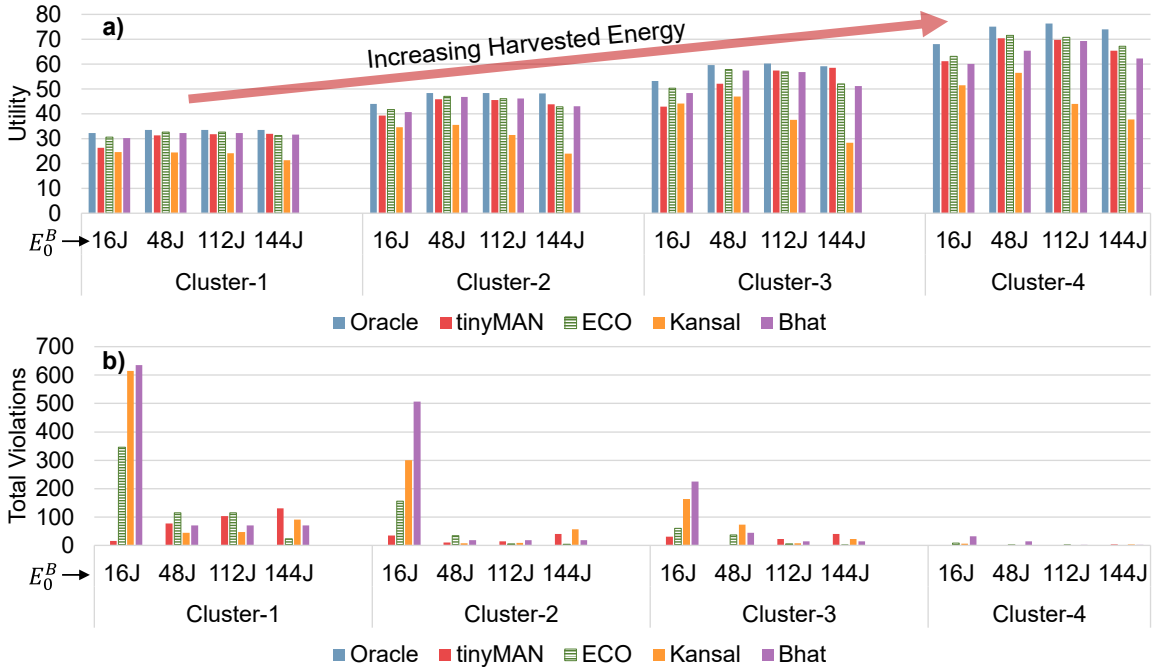


Figure 3.4: Summary of the performance of all approaches a) Total constraint violations b) Average utility

we emphasize that tinyMAN is trained to adapt to various battery energy levels and energy harvesting patterns without requiring energy harvesting predictions during runtime. This contrasts with prior approaches, whose performance heavily depends on accurate predictions.

Figure 3.5-(1a, 1b, 1c) illustrates the allocations made by tinyMAN for the initial battery level of $E_0^B = 16$J for three users (25%, 50%, 75%) in cluster 4. In addition to the behavior of the tinyMAN agent, Figure 3.5 also illustrates the energy allocations computed by three prior prediction-based approaches and the offline oracle solution. In contrast to tinyMAN, which implicitly learns the actual EH patterns during training, the prior approaches rely on specific expected EH patterns for users, depicted by the blue line in Figure 3.5-(3a, 3b, 3c). These expected EH patterns are derived using the cluster's mean EH pattern, representing the average EH pattern of users within that cluster. Conversely, tinyMAN's approach does not require explicit EH predictions, making it prediction-free. The oracle solution utilizes the actual harvested energy during the day, illustrated by the red line in Figure 3.5-(3a, 3b, 3c). This actual EH pattern corresponds directly to each user's own EH pattern, as discussed in Section 3.3.1. It can be seen that tinyMAN's allocations (actions) are around the optimal values shown with the red line in Figure 3.5-(1a, 1b, 1c). In contrast, prior approaches tend to deplete the battery and severely under allocate to avoid depletion, especially in the early hours of the day (before 10 AM). As a result of this, they often violate minimum and target battery energy level constraints. Moreover, this behavior causes their total daily utility, $U = \sum_{t=0}^{T} u(E_t^A)$, to decrease. If the deviation between the actual and expected EH values were greater

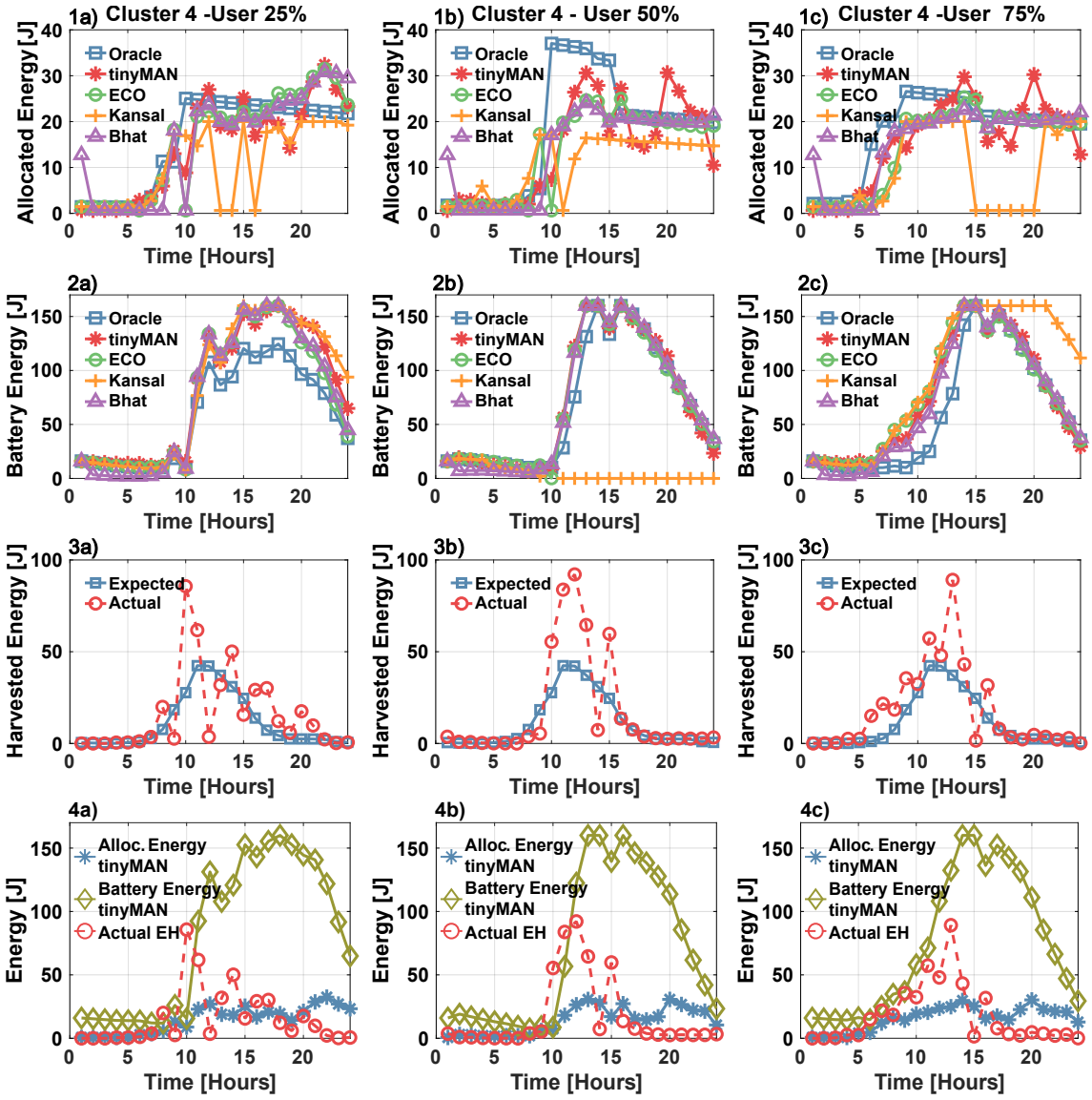


Figure 3.5: Comparison of tinyMAN with the prior approaches and the offline oracle solution for Cluster 4

in Figure 3.5-(3a, 3b, 3c), the performance of prior approaches would deteriorate significantly. Figure 3.5-(4a, 4b, 4c) clearly demonstrates tinyMAN's behavior throughout the day by superimposing the allocated energy, battery energy, and the

actual harvested energy on the same plot. It is evident that tinyMAN efficiently conserves battery energy early in the day to utilize it during periods of low and variable harvested energy later on. Indeed, tinyMAN achieves up to 17% higher utility than the prior approaches and within 8% of the oracle solution *while not causing any violations* for the three example cases depicted in Figure 3.5. In contrast, [57] has 6, [62] has 23, and [54] has 24 total violations. Thus, we argue that tinyMAN is the preferred energy management solution for wearable edge devices operating under conditions of uncertainty in harvested energy.

3.6.2 Multi-objective Results

3.6.2.1 Experimental Setup

We evaluate the performance of tinyMAN-MO by comparing it to an oracle solution derived using an offline solver (e.g., CVX) with the actual harvested energy data for the day. This offline solver transforms the objective vector into a scalar using the preferences for each objective (e.g., utility and battery energy level) as follows:

$$\begin{aligned} &\text{maximize} \quad \sum_{t=0}^{T-1} \gamma^t (\omega_1 u(E_t^A, D_t) + \omega_2 \frac{E_t^B}{E_{max}^B}) \\ &\text{subject to} \quad E_t^A > E_{min}^A \text{ and } E_{min}^B \leqslant E_t^B \leqslant E_{max}^B \end{aligned} \tag{3.16}$$

where γ is the discount factor and ω represents the preference for each objective. As in the single-objective case, this oracle solution is inherently unfair and unrealistic. However, it serves as a benchmark for comparison.

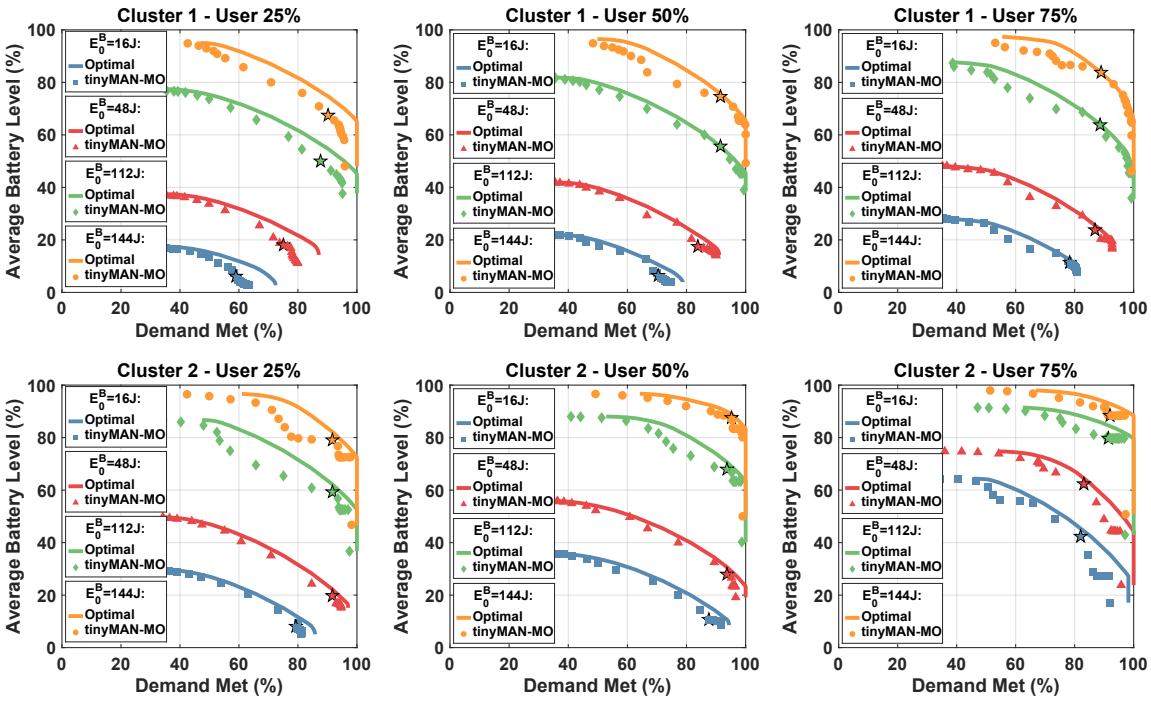
Similar to tinyMAN, we start by sorting all users in a cluster based on their

cumulative daily harvested energy. Subsequently, we exclude users corresponding to the {10%, 25%, 50%, 75%, 90%} of this sorted list from the training set for evaluation purposes. For each user, we evaluate the performance of tinyMAN-MO at four different initial battery energy levels: $E_0^B = \{16, 48, 112, 144\}$J. Furthermore, we divide the preference space into twenty equally spaced vectors {[0,1], [0.05, 0.95], ..., [1,0]}.

3.6.2.2 Performance Evaluation

Figure 3.6 illustrates the Pareto-front solutions obtained by tinyMAN-MO and the Pareto-optimal curve of the oracle for three users (25%, 50%, 75%) for all clusters. In this plot, the x-axis shows the percentage of the total daily demand met (objective-1), and the y-axis shows the average battery level during the day (objective-2). Consequently, the left-most data point in a given curve corresponds to the preference vector of [0,1] (i.e., only care about the battery level), and the right-most data point corresponds to [1,0] (i.e., only care about meeting the demand). Therefore, a preference vector of [0,1] signifies that the agent prioritizes maintaining the battery level over meeting the energy demand. Consequently, under this preference, the agent allocates the minimum required energy ($E_t^A = E_{min}^A$) to the application, except when the battery is fully charged by the harvested energy. In such cases, any excess energy is directly allocated to the application, inadvertently leading to an increase in objective-1, which focuses on meeting the demand. Conversely, a preference vector of [1,0] indicates that the agent's primary objective is to meet the energy demand, potentially disregarding the minimum battery level constraint, as

shown in 1a) Additionally, during periods of abundant energy, the demand can be met at a preference as early as [0.5,0.5], resulting in vertical "falls" on the right side of the plots. This suggests that under balanced preferences, both objectives are optimized simultaneously. Moreover, both objectives increase in value as the harvested energy escalates from Cluster 1 to Cluster 4. The Pareto-front solutions obtained by tinyMAN-MO closely resemble the Pareto-optimal solutions derived by the oracle, as depicted in Figure 3.7. Specifically, the mean absolute percentage error (MAPE) between the Pareto curves consistently remains below 10%, signifying a high degree of similarity. Furthermore, the MAPE tends to decrease with increasing harvested energy, indicating improved performance in scenarios with higher energy availability.



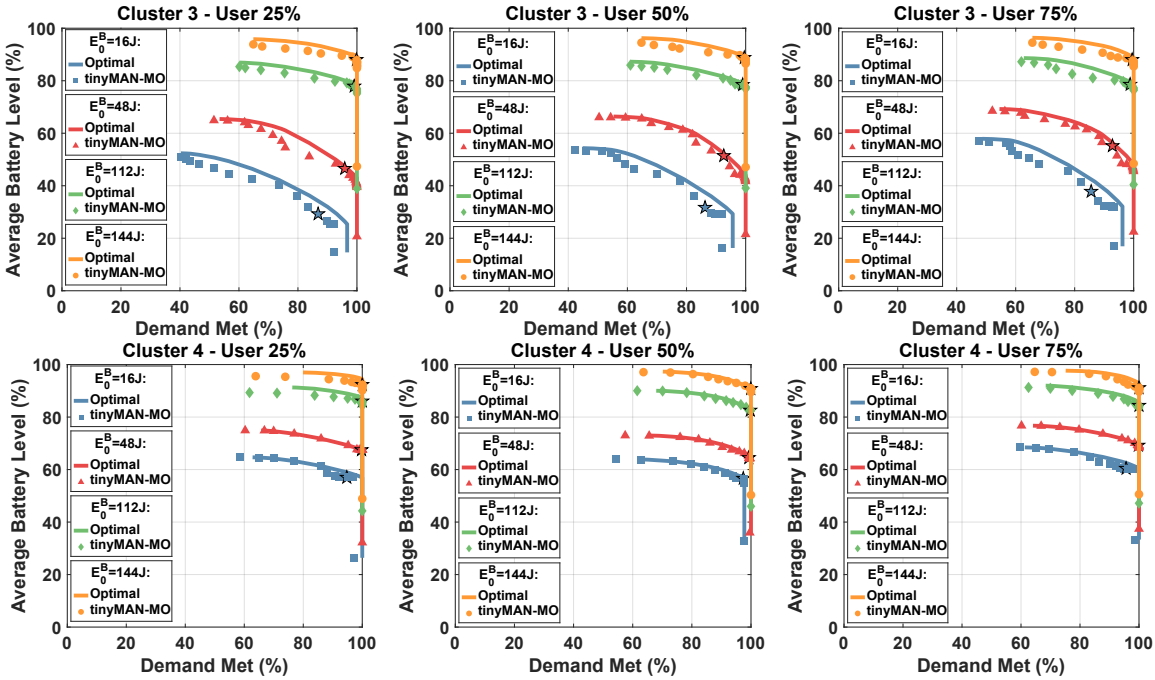


Figure 3.6: The Pareto front solutions obtained by tinyMAN-MO and the optimal solution. Solutions obtained using preference vector [0.5, 0.5] are highlighted by the 'star' marker.

3.7 Deployment on a Wearable Device

The TI CC2652R microcontroller used on our prototype device incorporates an ARM Cortex M4F running at 48 MHz and has 352KB of flash memory and 80KB of SRAM. These scarce resources highlight the importance of evaluating the trained RL agent regarding its deployability on the target platform. As the network architecture used by tinyMAN and tinyMAN-MO is the same, their memory footprint, energy consumption, and execution time overhead are nearly identical. Thus, in this section, we collectively refer to both frameworks as tinyMAN. In evaluating the deployability of tinyMAN, we consider three critical aspects: (i) The execution

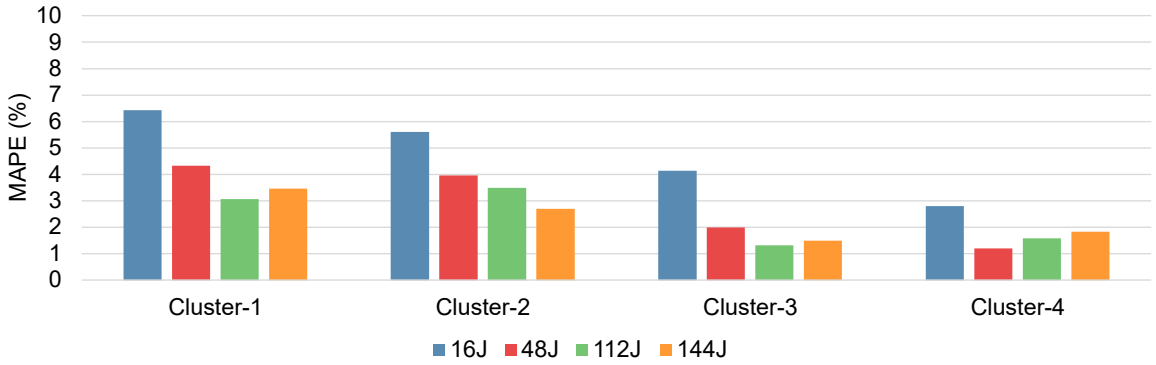


Figure 3.7: Summary of the mean absolute percentage error from five test users

time per inference, (ii) the energy consumption per inference, and (iii) the memory utilization of the target hardware platform. Our analysis follows the Tensorflow Lite Micro (TFLM) flow for converting and deploying the trained models on the target device, as outlined in [71]. Then, we measure the current consumption of the TI microcontroller. Based on these measurements, the execution time and energy consumption overhead of a single policy network call of the proposed tinyMAN is measured as 1.98 ms and 23.17 µJ, respectively. As a result, the total time and energy consumption of the tinyMAN framework for a single day are 47.52 ms and 0.556 mJ, respectively. These values are insignificant compared to the 24-hour period and 160 J battery capacity. Additionally, to assess memory utilization, we use the "Memory Allocation" report of TI Code Composer Studio. This report indicates that the memory footprint of tinyMAN amounts to 118 KB, a size that comfortably fits within the onboard memory of the target device. We note that the reported memory footprint is for the entire application, including necessary drivers and I/Os for debugging, such as UART and timers. These findings strongly support the conclusion that tinyMAN is readily deployable on resource-constrained wearable

IoT devices, thanks to its minimal impact on execution time, energy consumption, and memory utilization, making it an ideal solution for powering low-energy wearable devices.

Given that all the data needed to define the state space in the tinyMAN framework is readily accessible on the device, it ensures efficient operation and allows for continuous adaptation to changing environmental conditions. This real-time data availability allows the system to dynamically adjust its decision-making process based on current environmental states and user needs. By integrating online learning functionality, tinyMAN can iteratively refine its energy management strategies over time, enhancing its effectiveness and adaptability. This iterative learning approach enables tinyMAN to respond effectively to unforeseen environmental changes, ensuring optimal energy utilization in dynamic and unpredictable scenarios. Thus, by leveraging onboard data and online learning capabilities, tinyMAN is positioned to deliver robust performance while remaining responsive to evolving energy harvesting conditions.

4 GEM-RL: GENERALIZED ENERGY MANAGEMENT OF WEARABLE DEVICES USING REINFORCEMENT LEARNING

4.1 Motivation and Contributions

Wearable devices have become widely popular for health monitoring applications, activity recognition, smart entertainment systems, and smart fashion [72, 73, 74]. These devices typically include a lightweight, small, rechargeable battery to avoid user discomfort. However, small battery capacities limit the computational power and operating lifetime of the device. Energy harvesting (EH) from ambient sources has emerged as a promising solution for self-sustainable wearable devices [54, 50]. However, relying only on EH is not sufficient to achieve self-sustainability due to the uncertainties of ambient sources. Therefore, recent research has proposed energy management (EM) approaches to improve the application performance while removing the manual recharge requirements [54, 57, 64, 62].

The primary goal of EM is to maximize the utilization of the device by judiciously allocating the available energy according to the users' and applications' needs. EM approaches should cope with the stochastic behavior of the harvested energy and user patterns. State-of-the-art research employs dynamic optimization approaches along with a prediction method that obtains expected EH values [62, 63]. Thus, their performance depends critically on the accuracy of these predictions. The major drawback of predictive approaches is the critical dependency on user activity and location [63]. Additionally, solving an optimization problem for a specific

objective in a highly dynamic environment is not practical. For example, users may want to intervene to save battery for a given day which introduces a second dimension to the optimization problem. Therefore, there is a critical need for a *prediction-free* and *user-independent* EM approach that also *enables a multi-objective optimization*.

This work presents a **g**eneralized **e**nergy **m**anagement framework using multi-objective **r**einforcement **l**earning, GEM-RL. It learns the trade-off between utilization and the battery energy level of the target device under dynamic EH patterns and battery conditions without relying on forecasts of the harvested energy. GEM-RL transforms multi-dimensional objective space into a single dimension by assigning weights (preferences) to each objective [29]. It employs a multi-objective version of the TD3 (MO-TD3) [25] algorithm to obtain a single policy network that covers the entire preference space. The *inputs* of the policy network consist of battery and harvested energy-related values (*state*) and the preferences. This formulation enables an optimal policy (energy allocations) for any user-specified preference at run-time. We randomly sample an EH pattern, an initial battery energy level, and a preference vector for each episode during training. This enables the GEM-RL agent to generalize its policy for different EH patterns and battery energy conditions. GEM-RL also uses a lightweight approximate dynamic programming (ADP) approach that utilizes the trained agent to optimize the utilization of the device over a longer horizon. We also implemented an offline Oracle and two baseline EM approaches [54, 62] as comparison points. Extensive experiments show GEM-RL achieves Pareto front solutions within 5.4% of the Oracle for a given day. For a

horizon of 7 days, it achieves utility up to 4% within the Oracle and up to 50% higher utility compared to baseline approaches. Hardware implementation of GEM-RL shows the execution time and energy consumption overhead per inference are 1.98 ms and 23.17 μJ, while the memory footprint of the framework is only 118 KB.
Our major contributions are:

- GEM-RL, *a **g**eneralized **e**nergy **m**anagement framework* using *multi-objective **r**einforcement **l**earning* that learns the trade-off between any multiple objectives under various energy harvesting patterns and battery conditions,
- A broadly usable wearable device environment that can be used as a benchmark for evaluations of RL algorithms,
- Extensive evaluations that show near-optimal results within 5.4% of the Oracle, on average, for a given day and utility up to 4% within the Oracle and up to 50% higher utility compared to baseline approaches for a horizon of 7 days,
- Wearable hardware implementation with 118 KB memory footprint and 23.17 μJ per inference energy consumption.

4.2 Overview and Preliminaries

The proposed GEM-RL framework considers an environment that consists of an energy harvesting wearable device with processing capability, an EH source, and a rechargeable battery, as depicted in Figure 4.1. This section overviews these components as a background for the approach presented in Section 4.3.

Wearable Device and its Utility: The wearable device implements the target application, such as health monitoring. It should provide a high quality of service (QoS) to the application while maintaining the battery energy at a certain level. This QoS is modeled by a utility function, which is a non-decreasing function of the energy allocated to the application. That is, the application can perform better (e.g., provide higher accuracy, process data faster, remain active longer) when allocated more energy. The benefits can diminish and eventually saturate if the extra energy becomes redundant due to another constraint, such as the device's maximum capacity. We emphasize that *GEM-RL supports any arbitrary non-decreasing utility function* unlike prior work that requires convex functions [62, 63, 75].

EH Source and Battery Dynamics: The EH source generates energy that can be either transferred to the device or stored in the battery. GEM-RL is oblivious to the type (e.g., solar) of energy harvesting source. It also *does not* rely on any predictions of the harvested energy, GEM-RL uses only the actual harvested energy (E_t^H) at the end of a time step.

In this work, GEM-RL uses one-day long episodes ($T = 24$ hours) divided into one-hour time steps (t). We denote the battery energy level at the start of time interval t as E_t^B, the harvested and the allocated energy in time interval t as E_t^H, E_t^A respectively. There are two physical constraints on the battery energy level. The first constraint ensures that the battery maintains a minimum energy level, E_{min}^B, to stay idle so that the device does not turn itself off. The second constraint ensures that the battery energy level cannot exceed its limits, E_{max}^B. We can define

the battery energy dynamics in this environment as:

$$\begin{aligned} E^B_{t+1} &= E^B_t + E^H_t - E^A_t,\ t \in T \\ E^B_{min} &\leqslant E^B_t \leqslant E^B_{max},\ t \in T \end{aligned} \tag{4.1}$$

EH Dataset: GEM-RL can be trained with any available EH data. *To provide reproducible results that can be compared with prior work*, we use the publicly available *American Time Use Survey* [61] and solar/motion EH models from literature [50]. Our environment randomly draws the EH on each time step from this dataset. We emphasize that the proposed GEM-RL framework is general to work with any dataset.

4.3 Proposed Generalized Energy Management Framework - GEM-RL

4.3.1 Wearable Device Environment Dynamics for RL

RL techniques enabled breakthrough results in a wide range of topics, including autonomous driving, robotics, and gaming [25]. Impressive research progress in applying RL to these areas is primarily due to the broadly used open-source ML environments. For example, Google DeepMind's StreetLearn [76] environment and Microsoft Research's AirSim [77] lead to several novel approaches for autonomous driving. The first major contribution of this work is a general wearable device environment for RL. This environment enables us to apply the proposed MORL

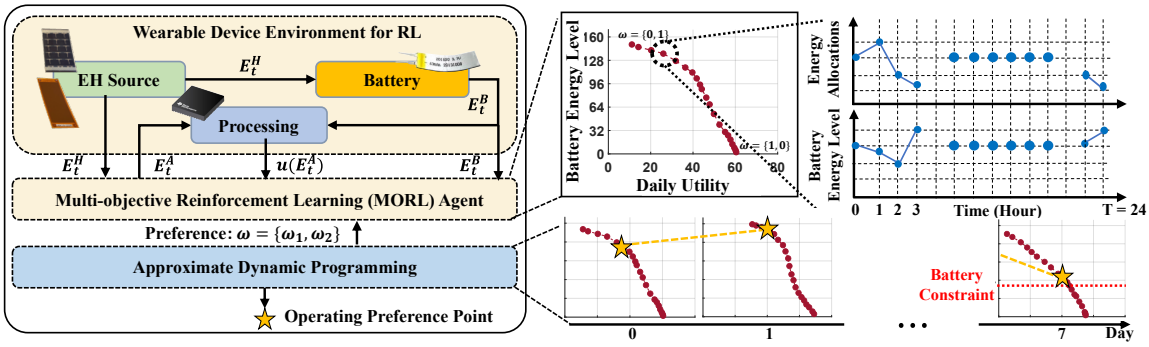


Figure 4.1: Overview of GEM-RL

techniques and ADP to wearable device energy management.

The proposed wearable device environment is designed with generality in mind to enable any reinforcement learning technique. The **state space** is defined as $\mathcal{S} \subseteq \mathbb{R}^6$:

- E_t^B: The battery energy level at time t.
- E_{t-1}^H: The EH during previous time $t-1$.
- $\sum_{\tau=0}^{t-1} E_\tau^A$: The cumulative energy allocations until time t
- E_0^B: Initial battery energy at the beginning of a episode.
- t: The current time step.
- $\sum_{\tau=0}^{t-1} E_\tau^H$: The cumulative EH until time t.

The environment either generates random EH patterns (E^H) and initial battery energy conditions (E_0^B) during training or takes these as inputs to the system for evaluation purposes.

The one-dimensional **action space** corresponds to the allocated energy at every time step ($E_t^A \in \mathbb{R}$). It is bounded by E_{min}^A from below such that the device can stay in an idle state.

The agent can have one or **multiple objectives** to consider. In this work, we consider two competing objectives: the utility of the target device and the battery energy level. The goal of the agent is to learn the trade-off between the utility of the device and the battery energy level under certain constraints. As mentioned before, GEM-RL supports any arbitrary utility function. To provide a fair comparison with prior work, our experimental evaluations use the following utility function $u(E_t^A) = \ln(\frac{E_t^A}{E_{min}^A})$, used in literature [62].

In the MORL setting, the objectives generally have different scales. To alleviate this problem and to impose the constraints on the battery, we define our *reward function* as:

$$r_t = \begin{cases} [\alpha\gamma^t u(E_t^A),\ \beta\gamma^t \ln(E_t^B)] & E_t^B \geqslant E_{min}^B \\ [\left|E_t^B - E_{min}^B\right|,\ \left|E_t^B - E_{min}^B\right|] & E_t^B \leqslant E_{min}^B \\ [\alpha\gamma^t u(E_t^A),\ \left|E_t^B - E_{max}^B\right| + \beta\gamma^t \ln(E_t^B)] & E_t^B > E_{max}^B \end{cases} \tag{4.2}$$

For our experiments, we use the coefficients $\alpha = 1$ and $\beta = 3$ and the discount factor as $\gamma = 0.95$. We impose the minimum battery energy level constraint using $\left|E_t^B - E_{min}^B\right|$ and the maximum battery energy level constraint using $\left|E_t^B - E_{max}^B\right|$ and assigning $E_t^B = E_{max}^B$ if the battery energy level exceeds the maximum capacity of the battery.

An episode terminates if time $T = 24$ is reached or the battery is completely

exhausted, $E_t^B \leqslant 0$. We develop our environment in Python and register it as an OpenAI's Gym [67] environment such that any RL algorithm can further adopt it.

4.3.2 Proposed MORL Framework

The goal of the MORL algorithm is to obtain a policy that yields a solution on the Pareto front for a given preference, as shown in Figure 4.1. The Pareto front is a set of solutions that are non-dominated by each other but are superior to the rest of the solutions in the solution space. In this work, we employ a recent MORL algorithm [25], MO-TD3, to learn to trade-off between utility and battery energy level. In this algorithm, the network takes preference vectors of the objectives as inputs along with the state vectors. It also employs novel approaches for efficient exploration of the preference space.

Algorithm 4 describes the training of the MORL agent for our framework. We measured the idle energy consumption of the target wearable device for an hour as 0.64 J. Therefore, we set E_{min}^A to this value. Similarly, the minimum battery energy level (E_{min}^B) is set as 1 J to allow our MORL agent to cover the corner cases where the battery is drained.

At each episode, we randomly choose an EH distribution from the dataset, an initial battery energy level, and a preference vector, $\boldsymbol{\omega}$. The agent then interacts with the environment using its actor-network (π_ϕ). The transitions collected $(s_t, a_t, \mathbf{r}_t, s', \boldsymbol{\omega}'_j, done)$ are then stored in the experience replay buffer $\mathcal{D}$. Then, the algorithm samples a minibatch (B) of transitions from $\mathcal{D}$ and updates both the actor and critic networks. The training terminates when N number of time steps

Algorithm 4: GEM-RL: MO-TD3

Initialize: Replay buffer $\mathcal{D}$, Critic networks $\mathbf{Q}_{\theta_1}$,$\mathbf{Q}_{\theta_2}$
and actor network π_ϕ with parameters θ_1, θ_2, and ϕ,
Target networks $\mathbf{Q}_{\theta_1'} \leftarrow \mathbf{Q}_{\theta_1}$, $\mathbf{Q}_{\theta_2'} \leftarrow \mathbf{Q}_{\theta_2}$, $\pi_{\phi'} \leftarrow \pi_\phi$.
for $n = 0$: N **do**
    if $\mathtt{done} = \mathtt{True}$ **then**
        Reset the environment to random E_0^B and EH pattern.
        Sample a preference vector ω.
    Observe state s_t and select action a_t.
    Interact with the environment and store the transition
    $(s_t, a_t, \mathbf{r}_t, s', \boldsymbol{\omega}, \mathtt{done})$ in $\mathcal{D}$.
    Sample random B transitions from $\mathcal{D}$; $\tilde{a} \leftarrow \pi_{\phi'}(s', \boldsymbol{\omega})$
    $\mathbf{y} \leftarrow \mathbf{r} + \gamma \arg_{\mathbf{Q}} \min_{i=1,2} \boldsymbol{\omega}^T \mathbf{Q}_{\theta_i'}(s', \tilde{a}, \boldsymbol{\omega})$
    $\mathtt{Loss}_{\mathtt{critic}_k}(\theta_i) = \mathbb{E}\Big[\Big(\mathbf{y} - \mathbf{Q}_{\theta_i}(s, a, \boldsymbol{\omega})\Big)^2\Big]$
    $\nabla_\phi \mathtt{Loss}_{\mathtt{actor}_k}(\phi) = \mathbb{E}\Big[\nabla_a \ \boldsymbol{\omega}^T \mathbf{Q}_{\theta_1}(s, a, \boldsymbol{\omega}) \nabla_\phi \pi_\phi(s, \boldsymbol{\omega})\Big]$
    Update critic, actor, and target network parameters.

is reached. We implement GEM-RL in Python. The modified hyperparameters from [25] are given in Table 4.1.

4.3.3 Proposed Approximate Dynamic Programming Technique

The MO-TD3 algorithm presented in the previous section determines near-optimal energy allocations in a single day for a given preference vector. Next, we need to

Table 4.1: Definition of the hyperparameters and their values.

Hyperparameter	**Description**	**Value**
N	Number of Time Steps	6×10^6
$\mathbb{D}$	Experience Replay Buffer Size	1×10^6
B	Minibatch size	128
η	Learning Rate	1×10^{-4}
N_L	Number of hidden layers	3
N_N	Number of hidden neurons	64

determine the preferences that maximize the utility over a longer period of time, such as a week or month. This section presents our second major contribution, a lightweight approximate dynamic programming (ADP) algorithm that *maximizes the device utility over a longer horizon* (H) *while maintaining a certain battery energy level* $E^B_{T_H}$ *at the end of the horizon*.

Algorithm 5 outlines the proposed novel ADP algorithm. Suppose the time horizon H is divided in the days h, $0 \leqslant h \leqslant H$. The state $S_h = \{E^B_{0_h}, E^B_{T_h}\}$ consists of the initial battery energy level ($E^B_{0_h}$) and the target battery energy level $E^B_{T_h}$ for step h (Line 1). The action/decision corresponds to a preference vector at the beginning of each day $\omega \in \Omega$ (Line 1). We define $U_h(\omega, S_h) = \sum_{t=0}^{T-1} u(E^A_t)$ as the total utility obtained at the end of day h, starting from state S_h for the preference ω. Using these definitions, our DP formulation becomes maximizing the total utility while meeting a minimum battery constraint, E^B_{Cstr}, at the end of the horizon (Line 2):

$$\max_{\omega \in \Omega} \sum_{h=0}^{H} U_h(\omega, S_h), \;\; st \;\; E^B_{T_H} \geqslant E^B_{Cstr} \tag{4.3}$$

Next, we express the principle of optimality [78] as: $\max_{\omega \in \Omega} \; U_{h=H}(\omega, S_h) + U^*_{h-1}(\omega, S_{h-1})$. Here, the $*$ symbol denotes the optimality of a given function. Then, we solve this equation using a backward recursion since we know $E^B_{T_H} \geqslant E^B_{Cstr}$ should be satisfied at the end of the horizon. However, an exact DP solution is not feasible on a wearable device due to the limited computational resources. Therefore, we approximate $U^*_{h-1}(\omega, S_{h-1})$ at each step except the first step ($h = 0$). To this end, we first generate an energy battery level set by dividing the battery energy

level range, $[0, 160]$, into M equal values (Line 4). We then use these M levels as the initial battery energy levels in state vector $S_{h,m} = \{E^B_{0_{h,m}}, E^B_{T_h}\}\, \forall m \in M$. Then, we use uniformly distributed energy allocations based on the expected EH, and actual initial energy battery level, $E^B_{0_{h=0}}$:

$$E^A_{t,m} = \frac{E^B_{0_{h=0}} - E^B_{0_{h,m}} + \sum_{h=0}^{h-1} \sum_{\tau=0}^{T} E^H_\tau}{T(h-1)}, t = \{0, T(h-1)\} \tag{4.4}$$

Using these energy allocations, we then obtain the previous utility values as $U^*_{h-1}(\omega, S_{h-1,m})$.

We run our trained MORL agent for each $\{\omega, S_{h,m}\}$ pair to calculate the utility for day h. Then, we simply select the pair $\{\omega, S_{h,m}\}$ that achieves the maximum of the $U_h(\omega, S_{h,m}) + U^*_{h-1}(\omega, S_{h-1,m})$ while achieving the E^B_{Cstr} constraint. We then use this state's initial battery energy ($E^B_{0_{h-1,m}}$) as the target battery energy E^B_{Cstr} of the previous step, $h-1$. We recursively apply this to decide the battery constraints E^B_{Cstr} for the previous steps until we reach the first step, $h = 0$. As a result, the proposed ADP approach optimizes the utility of the device by determining the preference vector ω at the beginning of each day over a longer horizon. It also calculates the difference between the actual battery energy level at the end of the day and the expected E^B_{Cstr} for step $h = 0$ and feeds this difference (Δ) back to the algorithm to correct the E^B_{Cons} at step $h = H$ for the next day.

Algorithm 5: GEM-RL: DP Technique

State: $S_h = \{E^B_{0_h}, E^B_{0_h}\}$, **Action:** $\omega \in \Omega$
Formulation: $\max_{\omega \in \Omega} \sum_{h=0}^{H} U_h(\omega, S_h)$ **s.t** $E^B_{T_H} \geqslant E^B_{Cstr}$
$\max_{\omega \in \Omega} U_{h=H}(\omega, S_h) + U^*_{h-1}(\omega, S_{h-1})$
Initialize: $E^B_{0_m}$ $m \in M$ and $E^B_{Cstr} = 80 + \Delta$
for h = H: 0 **do**
    if $h = 0$ **then**
        $\arg\max_{\omega} U_h(\omega, S_h)$ **subject to** $E^B_{T_h} \geqslant E^B_{Cstr}$
    else
        $E^A_{t,m}$ given by Equation 4.4
        Find $U^*_{h-1}(\omega, S_{h-1,m})$ $\forall$ $\omega \in \Omega, m \in M$ with $E^A_{t,m}$.
        Find $U_h(\omega, S_{h,m})$ $\forall \omega \in \Omega, m \in M$ using MORL.
        $\arg\max_{\omega,m} U_h(\omega, S_{h,m}) + U^*_{h-1}(\omega, S_{h-1})$
        subject to $E^B_{T_h} \geqslant E^B_{Cstr}$
        Assign $E^B_{Cstr} = E^B_{T_{h-1}}$

4.4 Experimental Evaluations

4.4.1 Experimental Setup

Optimal Oracle: We designed an offline Oracle using CVX [79]. It solves the following optimization problem for a given preference and finite horizon:

$$\begin{aligned} &maximize \quad \sum_{t=0}^{T-1} \alpha\gamma^t\omega_1 u(E^A_t) + \beta\gamma^t\omega_2 \ln(E^B_{t+1}) \\ &subject\ to \quad E^A_t > E^A_{min} \text{ and } E^B_{min} \leqslant E^B_t \leqslant E^B_{max} \end{aligned} \tag{4.5}$$

Since the Oracle uses the actual harvested energy values and allocations, it is infeasible, but it provides optimal results.

Baseline Approaches: We implemented two prior approaches [54, 62] for comparison. We use the median user's expected EH pattern in the EH dataset described in Section 4.2, and extend it over the horizon of 7 days. Both approaches use these future EH values to determine future energy allocations.

Evaluation conditions: The training set excludes the users corresponding to 10^{th}, 30^{th}, 50^{th}, 70^{th}, 90^{th} percentile cumulative EH in the dataset. We denote these EH patterns as EH_1, EH_2, EH_3, EH_4, EH_5, respectively. We also initialize the battery (E_0^B) at three different levels: 20% (32 J), 50% (80 J), 80% (128 J) of the battery capacity (160 J) and denote these as EB_1, EB_2, EB_3, respectively. We refer the combinations of the EH patterns and battery levels as the *evaluation set*.

4.4.2 Performance Evaluation of our MORL Agent

Figure 4.2 shows the Pareto front solutions of the Oracle and GEM-RL for each configuration in the evaluation set. Specifically, Figure 4.2(1a) plots the Pareto front solutions for three levels of E_0^B for the user with an EH pattern of EH_1. We observe that GEM-RL closely follows the Pareto front obtained by the Oracle. For example, the mean absolute percentage error (MAPE) with respect to Oracle averaged over all preferences obtained by GEM-RL is 5.5% for EB_1 (red curve). Similarly, the MAPE is 6.4% and 6.8% for EB_2 (blue) and EB_3 (black), respectively. The remaining plots (1b)–(1e) show that GEM-RL consistently performs very close to the offline Oracle. For instance, the MAPE over all preferences for EB_1, EB_2 and EB_3 is 6.0%, 3.81%, 3.88%, respectively, for EH_5 shown in Figure 4.2(1e). Considering all configurations in the evaluation set, the total MAPE averaged over all preferences is only 5.4%. We

further investigate the behavior of our approach by presenting the energy allocations in a day for a given initial battery energy level, EH pattern, and preference in Figure 4.2(2a-e). Our energy allocations closely follow those generated by Oracle. For example, Figure 4.2(2a) and (3a) show the energy allocations and battery energy levels obtained by the Oracle and GEM-RL for EB_1 with $\omega = \{1, 0\}$. In fact, this configuration is a corner case where the initial battery energy level is 20% (EB_1), the harvested energy is low (EH_1), and the preference ($\omega = \{1, 0\}$) is set to maximize the utility without considering the battery energy level. GEM-RL closely follows the Oracle even in this condition in terms of energy allocations and battery energy level, and its utility is within 0.5% of the Oracle's, without violating any constraints. Moreover, we observe that GEM-RL responds to changes in EH

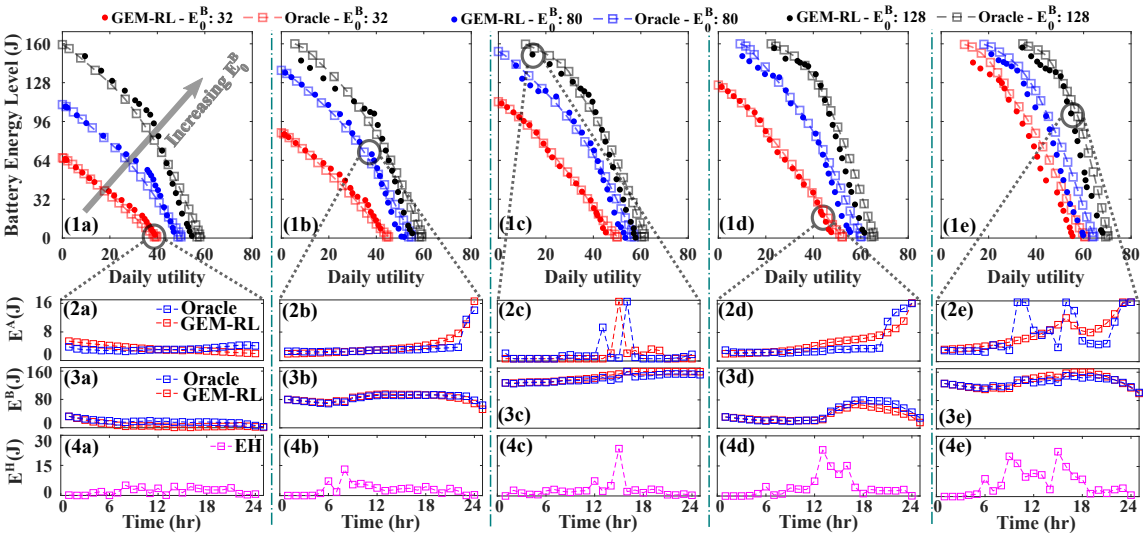


Figure 4.2: (1): Comparison of Pareto front solutions between an offline Oracle and GEM-RL using EB_1, EB_2 and EB_3 (a-e) EH_1, EH_2, EH_3, EH_4, EH_5, respectively. (2-4): Energy allocations, battery energy level, harvested energy obtained by Oracle and GEM-RL for a) EB_1, $\omega = \{1, 0\}$, b) EB_2, $\omega = \{0.5, 0.5\}$, c) EB_3, $\omega = \{0, 1\}$, d) EB_1, $\omega = \{0.75, 0.25\}$, e) EB_3, $\omega = \{0.5, 0.5\}$

similar to the Oracle. The error values and the Pareto front and individual solutions illustrated in Figure 4.2 clearly demonstrate that *GEM-RL successfully optimizes the trade-off between utilization and the battery energy level for any unseen EH pattern, battery condition, and preference without relying on the forecasts of harvested energy using a single extremely light-weight policy, as demonstrated in Section 4.4.4.*

4.4.3 Performance Evaluation of GEM-RL for Longer Period

The previous section shows the performance of the MORL algorithm throughout a single day. This section evaluates GEM-RL's novel ADP approach, over 7-day horizon ($H = 7$) with $M = 25$ battery levels and $E^{B}_{Cstr} = 80$ J final battery level constraint. Figure 4.3 shows the cumulative utility, daily utility, and battery energy levels obtained by the Oracle, baseline approaches [54, 62], and GEM-RL for EB_1, EB_2 and EB_3. Since the prior approaches [54, 62] decide future allocations from the predicted EH values, they completely drain the battery on the first day for corner cases such as low initial battery and low EH as shown in Figure 4.3(1-3a). To analyze the performance of the approaches, we calculate the MAPE of cumulative utilities over 7 days with respect to Oracle for each EH pattern in the evaluation set. We set the MAPE to 100% if the algorithm completely fails to keep the device on. Table 4.2 shows that GEM-RL achieves cumulative utility up to 4% within the Oracle and up to 50% higher cumulative utility compared to baseline approaches for 7-day horizon. Additionally, GEM-RL does not violate any constraints, whereas the baseline approaches [54] and [62], cause violations 16% and 10% of the total instances, respectively. Thus, we conclude that GEM-RL achieves near-optimal

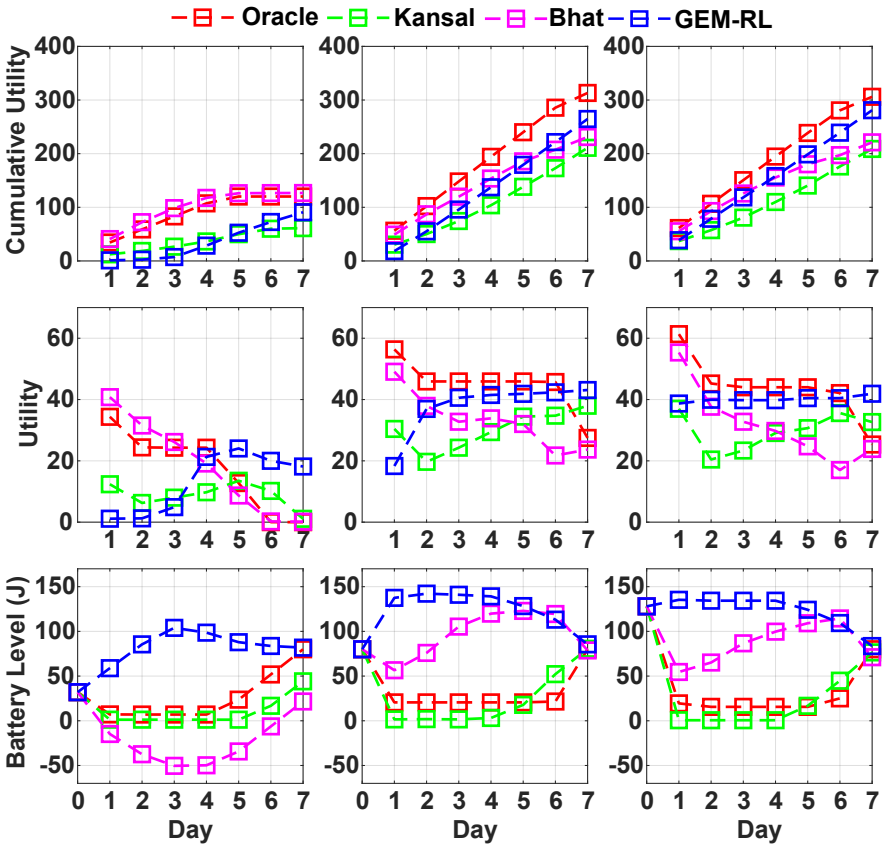


Figure 4.3: Comparison of cumulative utility, daily utility, and battery energy level for a longer period of time between an offline Oracle, baseline approaches, GEM-RL for (a) EB_1, EH_1, (b) EB_2, EH_4, and (c) EB_3, EH_3.

energy allocations with various EH patterns and battery conditions for an arbitrary horizon.

4.4.4 Energy-Performance Evaluation of GEM-RL on Hardware

We deployed our MORL agent using TensorFlow Lite for Micro (TFLM) flow [71] and implemented the GEM-RL framework on the TI CC2652R microcontroller (MCU). The MCU has an ARM Cortex M4F running at 48 MHz and has 352 KB of

flash memory and 80 KB of SRAM. The execution time and energy consumption overhead of a single policy network call are measured as 1.98 ms and 23.17 μJ, respectively. There are 24 network calls in one day for the specific preference. To determine this preference, the total execution time and energy consumption overhead of the ADP solutions are 7.32 s and 84.12 mJ, respectively. Hence, in total, the GEM-RL framework takes up to 7.37 s and consumes 84.58 mJ energy in a single day. These results are negligible compared to 24 hours and 160 J battery capacity. Moreover, considering the TFLM operators, inputs, outputs, intermediate values, and our agent's network weights, the memory footprint of GEM-RL becomes 118 KB. These results suggest that GEM-RL provides an efficient general EM framework that is deployable on a wearable device.

Table 4.2: MAPE of cumulative utilities over 7-day horizon for each configuration in the evaluation set w.r.t. Oracle. For each EH pattern, the MAPE of EB_1, EB_2, EB_3 are averaged.

	EH_1	EH_2	EH_3	EH_4	EH_5
[54]	100%	100%	28%	25%	44%
[62]	48%	47%	32%	32%	22%
GEM-RL	4%	5%	8%	16%	20%

5 A SELF-SUSTAINED CPS DESIGN FOR RELIABLE WILDFIRE MONITORING

5.1 Motivation and Contributions

Wildfires occur naturally in many ecosystems worldwide, with increasing frequency and severity in recent years, as depicted in Fig. 5.1. According to data from the European Forest Fire Information System, wildfires in 2022 have burnt 3× more area than the historical average, as shown in Fig. 5.1-a) [80]. Similarly, The United States National Interagency Fire Center reports that the total burned area has doubled in the last 20 years, as shown in Fig. 5.1-b) [81]. The devastating wildfires have far-reaching ecological, social, and economic impacts, such as the loss of human lives, homes, property, and infrastructure, as well as adverse effects on animal health and air quality.

The increasing trend in wildfire occurrences is due to various human-caused factors, including logging, agriculture, and urbanization, which alter the landscape, and external ignition sources, such as cigarettes, campfires, and electric power infrastructure. Among these, electric power infrastructure poses a particular risk because it can cause highly deadly wildfires and be disabled by them. For example, the 2018 Camp Fire in California caused 85 fatalities, over $16 billion in damage, and led the power utility to bankruptcy [82]. Thus, monitoring areas near power grids is essential to developing effective strategies for preventing or promptly detecting grid-caused wildfires.

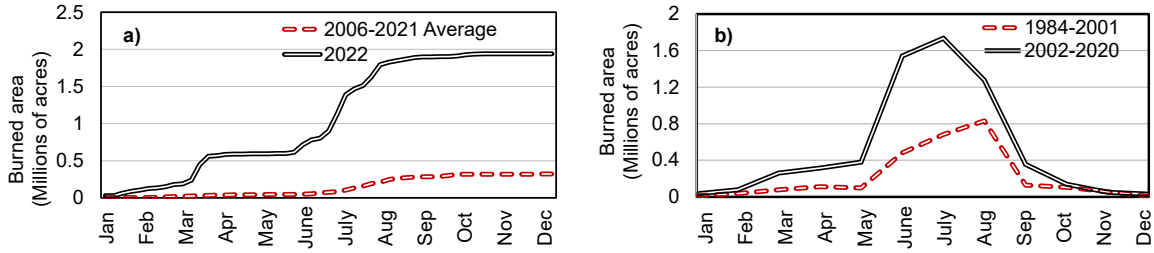


Figure 5.1: a) Cumulative burnt area in Europe in 2022 compared to the 2006-2021 average. b) Average monthly burnt area in the US between 2002-2020 compared to between 1984-2001.

Current preventive techniques are based on various risk indexes that represent a rough likelihood of fire occurrences, such as the Wildland Fire Potential Index (WFPI), the National Fire Danger Rating System (NFDRS), and others [83, 84, 85]. These indexes use meteorologic data from weather stations hundreds of miles away from the actual location of interest (i.e., the electric grid in this case). Hence, the temporal and spatial resolutions of these index maps are poor. Similarly, the existing satellite and airborne wildfire detection systems collect intermittent data. Moreover, they are unaware of local ambient factors such as high wind speeds or low humidity, which results in poor awareness and long detection times [86]. Motivated by these shortcomings, recent research has focused on a new class of data-informed, energy-harvesting cyber-physical systems (CPS), i.e., sensor suites, to improve wildfire prevention and detection [87]. The deployed sensor suites must comprise multiple sensors for accuracy and be self-sustainable due to low maintenance requirements. However, accommodating many sensors with varying levels of power consumption, as well as the dynamic nature of the environment, make achieving self-sustainability difficult. The current practice to alleviate this problem

is over-designing the system (i.e., using larger than necessary batteries and energy harvesters) and employing heuristics to control the total energy consumption by duty-cycling the sensors. However, these solutions increase the cost of the system and are far from optimal. Therefore, there is a strong need for theoretically grounded and practical cyber-physical systems that maximize monitoring accuracy while ensuring self-sustainability.

This work presents a CPS framework that achieves self-sustained operation throughout the system lifetime (years). We consider sensor suites that integrate multiple sensors, such as temperature and particle, with different accuracy and power consumption. The sensor suites replenish the battery by harvesting solar energy to avoid manual maintenance. On the one hand, the sensors must stay on with a high sampling rate to maximize the detection accuracy and response time when a wildfire approaches. On the other hand, the sensor suite must preserve energy for arbitrarily long durations (five years in our evaluations) since wildfires can happen anytime, and the battery cannot be replaced easily due to remote locations. One can formulate this challenge as a constrained optimization problem and solve it with modern solvers like CPLEX [58] and Gurobi [59]. However, these approaches are impractical for online deployment since they require long execution times [60] and significant computational power beyond what simple sensor suites can provide. As a powerful and practical alternative, we propose a novel RL framework to co-optimize the sensor energy consumption and wildfire monitoring accuracy. The primary advantages of the proposed CPS framework are:

- *Efficient online deployment* – After the training stage is completed, the execution

time of the policy network is negligible, facilitating online deployment.

- *Model-free* – RL implicitly learns the energy harvesting and consumption patterns, which makes it a *prediction-free* approach, i.e., it does not rely on forecasts of the future.

The proposed framework trains an RL agent that controls the sensor sampling rates such that the accuracy of sensor readings is maximized without depleting the battery and forcing the sensor suite to enter a sleep state. As a result, it achieves *reliable wildfire monitoring with a self-sustained operation* for as long as five years, as demonstrated in Section 5.5. To train the agent, we develop a detailed environment for wildfire monitoring CPS that models the energy consumption due to sampling, data processing, communication, and standby power, as well as the sensing accuracy. Evaluations under different wildfire scenarios demonstrate that the trained RL agent successfully controls the sensors under dynamic, uncertain conditions. Specifically, it achieves 89% system uptime in 5-year long simulations, 46% higher than a carefully tuned heuristic approach. Similarly, the RL agent averages a 2-minute initial response time, which is at least 2.5$\times$ faster than the heuristic approach. In addition, when deployed on the TI CC2652R microcontroller, the agent consumes only 0.6 mJ in a day, which is negligible compared to the energy consumption of the sensors. *In summary, our major contributions are:*

- A novel RL based CPS design for self-sustained, reliable and continuous monitoring of wildfires,

- An open-sourced, detailed RL environment that can simulate wildfires and model the energy consumption of a sensor suite, and the behavior of the sensor readings, which can be used *beyond* training RL agents,
- Detailed experimental evaluation of the proposed CPS in terms of long-term performance and energy consumption on the TI CC2652R microcontroller.

5.2 Related Work

There are three current mechanisms for wildfire monitoring: Unmanned aerial vehicles (UAVs), satellites, and sensor networks. UAVs commonly refer to vehicles or systems that are remotely operated and travel by flight. They have become an increasingly popular tool for detecting wildfires due to their ability to cover large areas quickly and provide real-time data to ground crews [88, 89]. Despite their advantages, UAVs are unsuitable for continuous, uninterrupted monitoring in wildfire-prone areas owing to their limited flight times [86]. Geostationary satellites address the lack of continuous availability of UAVs, as they can monitor a specific area continuously for fires [86]. With more accessible access to satellites through projects such as the CubeSat [90, 91], recent research has focused on wildfire detection algorithms using the low-resolution images taken by the satellites. Furthermore, since sending images to the ground stations may take hours, system designs for processing the images at the edge gained traction [92]. Despite these efforts, the limited spatial and temporal nature of the data makes timely detection challenging. For example, detecting young fires smaller than a pixel is extremely

difficult [86]. In addition, dense clouds, smoke, and other atmospheric effects may lead to false alarms or missed detections.

Wireless sensor networks (WSNs) address the limited spatial and temporal coverage of airborne approaches. They consist of many distributed sensor suites that accommodate various sensors, such as humidity, temperature, and gas particles, to monitor the covered area for wildfires. As a result of their promising outlook, recent research efforts focused either on improving the detection probability through the use of machine learning [93, 94], improving the bandwidth of the link between suites [95] or network-level optimizations that minimize network-wide communication energy overhead [96, 97, 98]. However, the benefits of WSNs quickly diminish if the sensor suites run out of battery, as maintenance and replacement require significant manual labor and logistic effort. In addition, many wilderness zones are protected from e-waste [86]. Therefore, *guaranteeing self-sustained operation* to achieve maximum device lifetime (at the suite level) is extremely critical for this technology. Yet, most existing literature does not discuss the performance of the suites for extended periods of time, such as 5+ years. Most importantly, the existing literature targets perpetual monitoring rather than stochastic events like wildfire occurrence. In contrast, our formulation aims to address the stochasticity of wildfire occurrence and improve performance at the node level. We provide 1-year and 5-year simulations that depict how the performance of the RL agent is affected over time.

Energy harvesting and management approaches for self-sustained operation gained attention thanks to decreasing energy consumption requirements of sens-

ing, processing, and communication hardware. One of the early studies in this field [54] proposes a linear programming approach to determine the duty cycle of the IoT device for self-sustained operation. This approach relies on the predictions of harvested energy to determine the future on-time of the device. A more recent approach proposes a rollout-based optimization to determine the energy consumption budget for a wearable IoT device [57]. Similarly, this approach first finds initial solutions based on the predicted values for future harvested energy. It then corrects the initial solutions in runtime by considering variations between predicted and actual harvested energy to achieve self-sustainability. *The performance of such approaches depends critically on prediction accuracies.* RL-based approaches eliminate this dependency and enable prediction-free techniques. Various studies have used RL to manage sampling rates and energy consumption in WSNs [99, 100, 101, 64, 102]. For example, RLMan [64] utilizes RL to optimize the packet generation rate in a point-to-point communication system. Similarly, tinyMAN [102] utilizes RL to allocate energy for a wearable IoT device for self-sustainability. However, the environment dynamics, the RL algorithm, and the performance metrics they optimize do not translate to wildfire monitoring. For example, they focus on optimizing either the data processing energy or the communication energy only (i.e., they do not consider the sensor sampling energy). In addition, these approaches lack thorough energy consumption and harvesting models. They do not model the sensor behavior, which is crucial for wildfire monitoring. In contrast, the current work explains the underlying models in detail.

To the best of our knowledge, there are no prior sensor accuracy-energy co-

optimization techniques for wildfire monitoring. The self-sustainable design choices have been either over-designing the energy harvester and the battery [103] or using very low-power, low-cost sensors [87]. Over-designing increases the cost and makes deployment difficult, and it still does not guarantee self-sustainability. Conversely, using low-cost sensors may compromise the reliability of a long-term solution in terms of hardware lifetime and measurement accuracy. Therefore, a self-sustained wildfire monitoring CPS design is an essential contribution to the wildfire monitoring literature. To this end, we propose an RL-based framework to co-optimize the sensor energy consumption and wildfire monitoring accuracy for the first time in the literature.

5.3 Overview and Preliminaries

5.3.1 Objective of the Proposed RL-based Framework

The proposed CPS design considers an energy-harvesting sensor suite with sensing, processing, and communication capabilities, an energy-harvesting source, and a rechargeable battery, as depicted in Fig. 5.2. The sensor suites are attached to existing electric towers in rural areas to monitor the environment using the various sensors. We assume the battery energy is complemented by harvested energy since the power utility companies do not allow any direct physical interface to the wires due to the safety, liability concerns, and cost of power conversion equipment. This work employs solar energy harvesting because it is a robust and mature technology [57], but our formulation does not exclude any other energy harvesting modality, such

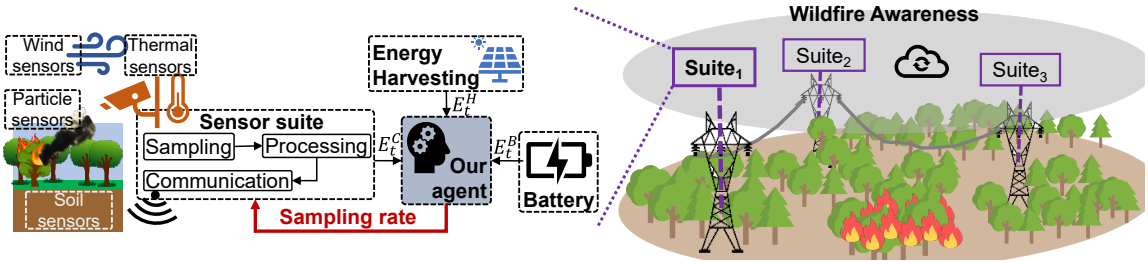


Figure 5.2: Overview of the proposed CPS design: An energy harvesting sensor suite with sensing, processing, and communication capabilities. Our RL-based framework trains an agent to control the sampling rates of the sensors on the sensor suite.

as electromagnetic coupling with the power distribution lines [104]. As there are no trees near the towers, shade from trees is not a concern. When the sensor suites detect a potential wildfire, they communicate the preprocessed data to a central node where decision-making (e.g., risk index calculation and intervention) occurs. This work focuses on producing accurate sensor readings at the edge regardless of the onset time of a wildfire. Therefore, the suite must obtain accurate sensor readings for robust decision-making while avoiding running out of battery. This is challenging because the response time and accuracy of the sensor readings improve by using higher sampling rates. In turn, energy consumption increases with sampling rates due to higher sensing, data processing, and communication energy. Consequently, the energy in the battery drains faster, and achieving self-sustainability becomes a challenging objective. Our framework poses this objective as an optimization problem and solves it using RL:

Objective: Maximize the accuracy of sensor readings by increasing the sampling rates while keeping the device operational at all times (i.e., the battery is not depleted).

The rest of this section presents the mathematical background that allows us to describe the dynamics of the sensor node as an RL environment. Finally, it formulates the optimization problem.

5.3.2 Sensor and Energy Models

5.3.2.1 Sensor Noise and Accuracy

Each of the sensor suites in Fig. 5.2 has multiple sensors. The sensors in different sensor suites will read different values due to their relative locations with respect to the wildfire. For example, in this case, the temperature sensors on suite_1 and suite_3 will read higher temperatures than the sensor on suite_2. Similarly, the sensor readings on suite_1 and suite_3 will be more noisy (higher variance) due to a nearby heat source. There are no existing studies in the literature that model how sensor readings behave during a wildfire. Therefore, we used the Fire Dynamics Simulator (FDS) [105], a sophisticated computational fluid dynamics model of fire-driven fluid flow, to simulate the smoke particles and heat during the progression of a wildfire. Our experiments with representative wildfire scenarios reveal that the sensor readings are exponentially correlated with the distance from the fire. Using the measurements from detailed FDS simulations, we use a normalized model for the expected sensor readings:

$$\mathbb{E}[S(d)] = Me^{-kd} + N \tag{5.1}$$

where M is the maximum of the sensor reading range, k is a constant that controls the decay rate of sensor readings with distance (i.e., the sensor's far-sightedness), N is the nominal reading value for the sensor, and d is the distance between the sensor and the firefront. For example, Fig. 5.3-a) shows three sensors with different k values. Sensor 1 starts to deviate from the nominal at greater distances to fire than sensors 2 and 3, i.e., sensor 1 is more sensitive to fire than sensors 2 and 3.

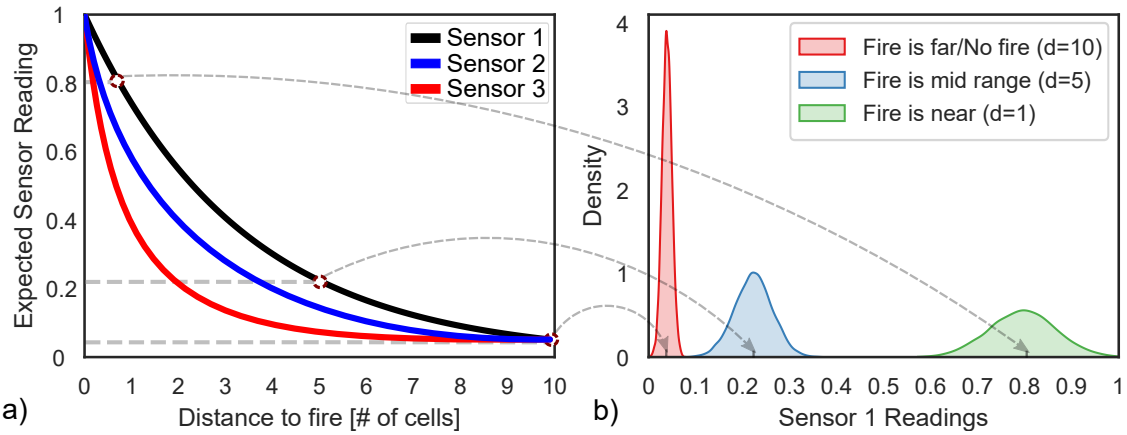


Figure 5.3: An overview of the sensor model. a) The expected sensor readings are modeled with exponentially decaying readings as a function of the distance to fire. b) The actual sensor readings are drawn from a distribution that is centered around the expected sensor reading.

Given the expected sensor reading model in Equation 5.1, the actual sensor readings are drawn from a normal distribution with $\mu = \mathbb{E}[S(d)]$ and $\sigma = 1/d$.

$$S_i(d) \sim \mathcal{N}(\mathbb{E}[S_i(d)], \frac{1}{d}), \quad S_i \in \mathbb{R}^{a_i} \tag{5.2}$$

Here, the standard deviation of the distribution is inversely proportional to the distance to fire. This relationship models the increase in noisy readings when the

fire is nearer. For example, sensor 1 is expected to read 0.22 at a distance of 5 units from the fire, as shown in Fig. 5.3-a). This constitutes the mean of the distribution of the sensor readings in Fig. 5.3-b). The larger the distance to fire, the smaller the deviation and the narrower the distribution gets, and vice versa as shown in Fig. 5.3-b). Finally, a_i samples are drawn from the distribution (Equation 5.2), and their sample mean constitutes the actual sensor reading $\hat{S}$:

$$\hat{S}_i(a_i, d) = \frac{1}{a_i} \sum_{n=0}^{a_i} S_i(d) \tag{5.3}$$

where i denotes the sensor index and a_i is the sampling rate per hour for sensor i. Thus, higher sampling rate improves sensing accuracy by sampling more samples from the distribution. Using higher sampling rates is more critical in the presence of a wildfire to maximize sensing accuracy.

5.3.2.2 Energy Consumption of the Suite

We model the energy consumption of the sensor suite as shown in Fig. 5.4. This figure shows six main stages for the operation of the sensor suite: i) The suite wakes up from sleep (T^W), ii) the sensor i measures a sample (T^S), iii) the microcontroller (MCU) processes the collected data (T^P), iv) data is transmitted to neighboring sensor suites (T^{Tx}), v) an acknowledgment is expected before going back to sleep (T^{Rx}), vi) the suite goes to sleep mode until the next wake up. Since each sensor on the suite can be assigned different sampling rates, *this cycle is executed on a per-sensor basis*.

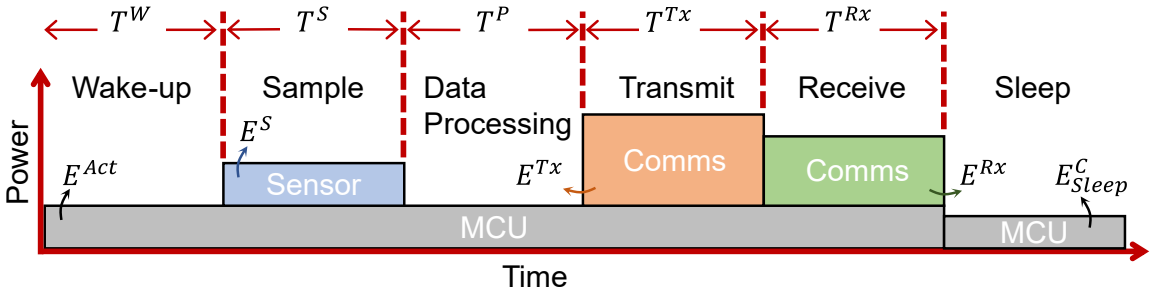


Figure 5.4: The energy consumption model. Each sampling event repeats these steps. *Comms: Communications.

The total active time for a sensor i is

$$T_i^{Act} = T^W + T_i^S + T_i^P + T_i^{Tx} + T^{Rx} \tag{5.4}$$

Since different sensors use different techniques for measurements, the sampling time T^S is a function of sensor (i). For example, a temperature sensor typically has a shorter T^S than an image sensor. Similarly, different sensors use different number of bits to represent a sample. For example, a temperature sensor uses 4 bytes per sample, whereas an image sensor may use 640x480x3 bytes per sample. As a result, T^P and T^{Tx} are also functions of individual sensors.

The energy consumption for the node due to sensor i has the following four main components:

$$\begin{aligned} E_i^{Act} &= T_i^{Act} P_{active}^{MCU} & E_i^S &= T_i^S P_i^S \\ E_i^{Tx} &= T_i^{Tx} P^{Tx} & E^{Rx} &= T^{Rx} P^{Rx} \end{aligned} \tag{5.5}$$

where P_{active}^{MCU} is the power consumption of the microcontroller, P_i^S is the power consumption of the sensor i, and P^{Tx} and P^{Rx} are the power consumption owing to the transmit and receive phases of the communication protocol, as listed in Table 5.1.

Using the above, *the hourly energy consumption* of the node due to a sensor i is:

$$E^C(a_i) = (E_i^{Act} + E_i^S + E_i^{Tx} + E^{Rx})a_i \tag{5.6}$$

Consequently, the *the hourly energy consumption* of the node due to sleep is:

$$E_{Sleep}^C = (3600 - \sum_i T_i^{Act} a_i) P_{idle}^{MCU} \tag{5.7}$$

where P_{idle}^{MCU} is the power consumption of the microcontroller during sleep mode. Finally, *the hourly energy consumption of the node* is given by:

$$E^C(\mathbf{a}) = \sum_i E^C(a_i) + E_{Sleep}^C \tag{5.8}$$

where $\mathbf{a} \in \mathbb{N}^n$ is the vector of sampling rates a_i assigned to sensors $i \in [1, 2, ...n]$.

5.3.2.3 Energy Harvesting and Battery Dynamics

The energy harvesting source converts the energy in the surrounding environment into usable electrical energy, which is often used to extend the battery lifetime. There are many energy harvesting modalities, including light, motion, heat, and electromagnetic coupling based harvesters. The proposed framework is oblivious to the type (e.g., light) of energy harvester, and it does not rely on any predictions of the harvested energy. Our evaluations use solar energy harvesting data, as detailed in Section 5.4.1.2.

Our framework uses 150-hour long episodes (T=150 hours) divided into one-

hour time steps (t). We denote the battery energy level at the start of time step t as E_t^B, the harvested and the consumed energy in time step t as E_t^H, E_t^C respectively. Using the above, the following equation governs the evolution of the battery energy:

$$E_{t+1}^B = E_t^B + \eta E_t^H - E_t^C(\mathbf{a}),\ t \in T \tag{5.9}$$

where η accounts for the inefficiencies at the energy harvesting and battery charging interfaces. Furthermore, there are two physical constraints on the battery energy level, such that the battery level is bounded between empty (i.e., $E_{min}^B = 0\%$) and full (i.e. $E_{max}^B = 100\%$):

$$E_{min}^B \leqslant E_t^B \leqslant E_{max}^B,\ t \in T \tag{5.10}$$

5.3.2.4 Problem Formulation

Using equations 5.1, 5.3, 5.8–5.10, we formulate the optimization problem as follows:

$$\begin{aligned} &minimize \quad \sum_i |\hat{S}_i(a_i, d) - \mathbb{E}[S_i(d)]| \\ &\text{subject to} \quad E_{t+1}^B = E_t^B + \eta E_t^H - E_t^C(\mathbf{a}),\ t \in T \\ &\qquad\qquad E_t^B \geqslant E_{min}^B \quad E_t^B \leqslant E_{max}^B \quad E_T^B \geqslant E_0^B \end{aligned} \tag{5.11}$$

where i denotes the different sensors. Thus, the optimal solution minimizes the absolute error in sensor readings (i.e., the objective) while satisfying battery constraints by optimizing the sampling rates of the sensors.

Table 5.1: Symbols used in Section 5.3.2 and their description.

Symbol	Description	Symbol	Description
d	Distance from the fire	T^W	Wake-up time for the node
M	Maximum sensor reading	T_i^S	Sensing time for sensor i
k	Sensor distance constant	T_i^P	MCU data processing time for sensor i
N	Nominal sensor reading	T_i^{Tx}	Comms transmit time for sensor i
n	Number of sensors on the node	T^{Rx}	Comms ack receive time
i	Sensor index $i \in [1, 2, ...n]$	T_i^{Act}	Total active time for sensor i
a_i	Samples/hour for sensor i	η	Efficiency of the energy harvester
$\mathbf{a}$	Vector of a_is: $[a_1, a_2, ...a_n]$	t	Time step (interval length)
E_t^B	Battery level at the start of interval t	T	Episode length
E_t^H	Harvested energy in interval t	P_{active}^{MCU}	Power consumption of the MCU
E_{min}^B	Minimum battery level	P_{idle}^{MCU}	Idle power consumption of the MCU
E_{max}^B	Maximum battery level	P_i^S	Power consumption of the sensor i
E_T^B	Battery level at the end of episode	P^{Tx}	Comms transmit power consumption
E_i^S	Energy consumption per sample of sensor i	P^{Rx}	Comms receive power consumption
E_i^{Tx}	Energy consumption per sample for data transmit due to sensor i		
E^{Rx}	Energy consumption per sample for ack receive		
$E^C(a_i)$	Hourly energy consumption of the node due to sensor i		
E_{Sleep}^C	Hourly energy consumption of the node due to sleeping		
$E^C(\mathbf{a})$	Total hourly energy consumption of the node		
E_i^{Act}	Energy consumption per sample of the microcontroller due to sensor i		
$\mathbb{E}[S_i(d)]$	Expected sensor reading for sensor i at distance d from the fire		
$S_i(d)$	Sensor readings sampled from $\mathcal{N}(\mathbb{E}[S_i(d)], \frac{1}{d})$		
$\hat{S}_i(a_i, d)$	Actual sensor reading for Sensor i set to a_i samples/hr at distance d from the fire		

5.4 Proposed RL-based CPS Design Framework

RL methods have led to remarkable advancements in diverse fields, such as autonomous driving, robotics, and gaming. The remarkable strides in RL's application in these domains are largely attributed to the widespread adoption of open-source ML frameworks. For example, Google DeepMind's StreetLearn [76] and Microsoft Research's AirSim [77] environments have facilitated several innovative approaches to autonomous driving. Thus, designing a representative environment for wildfire monitoring is of utmost importance as it enables the application of RL techniques

for wildfire detection. As such, a significant contribution of this work is an RL environment for wildfire propagation and sensor suite dynamics. This environment enables us to apply the proposed RL-based framework for reliable wildfire monitoring. This section first presents the design of this RL environment. Then, it describes the proposed TD3 framework that leverages this environment.

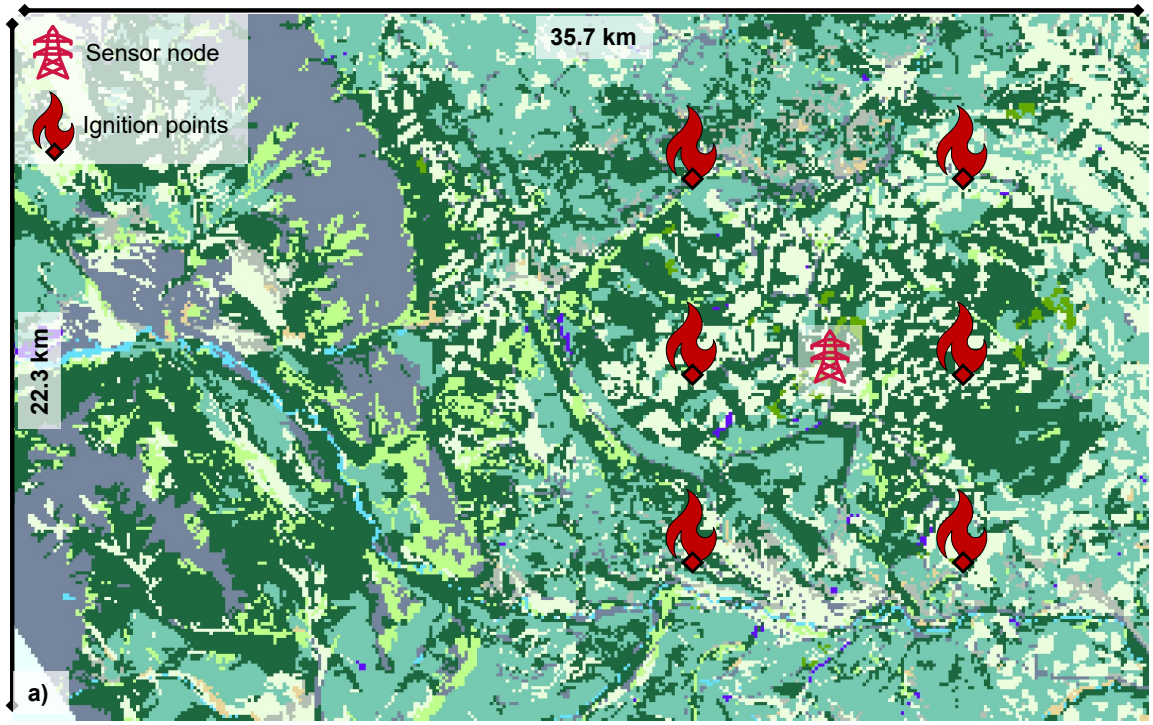
5.4.1 RL Environment Design for Wildfire Monitoring CPS

5.4.1.1 Wildfire Simulation

Detailed fire simulators, such as FDS, are extremely slow due to the underlying differential equations. For example, executing the small-scale test fires in Section 5.3.2.1 takes several hours. Therefore, we employ FDS only for characterizing the sensor readings and revert to Cell2fire [106], another open-source wildfire simulator, for large-scale wildfire simulations. Cell2fire employs cellular-automata networks to model wildfires at a higher level than FDS. It uses real-world temperatures, wind measurements, and terrains, as shown in Fig. 5.5. The Cell2fire simulator has been validated using several real-world fires. Among these, we use the Dogrib Creek instance for the underlying elevation and vegetation map. This instance consists of a rectangular grid of 357x223 cells, where each cell is a 100 m x 100 m square (the total area is 35.7 km x 22.3 km). Each cell has an elevation and vegetation value that determines the fire's rate of spread within that cell. The simulator combines this information with ambient temperature and wind data to simulate the direction and the rate of spread of the fire across the grid given an ignition point. Therefore, to generate a burn trace, we choose a random 150-hour segment from an hourly

temperature and wind dataset from the BANFF CS Canadian weather station. In addition, we randomly pick an ignition point from the set of 6 points shown in Fig. 5.5-a). By doing this, we model wildfires approaching from different directions with upwind and downwind conditions. In total, we generate 6000 different 150-hour-long wildfire burn traces. An example burn trace that shows the wildfire at three time instants during its evolution is shown in Fig. 5.5-b). Finally, we stress that our framework is not limited to the Dogrib Creek map or Cell2fire.

Despite the relatively faster speed of Cell2fire, embedding a wildfire simulator inside an RL environment is not feasible for training since RL algorithms require thousands of episodes to converge, and running a new simulation at each episode slows down the training process significantly. To address this challenge, we de-



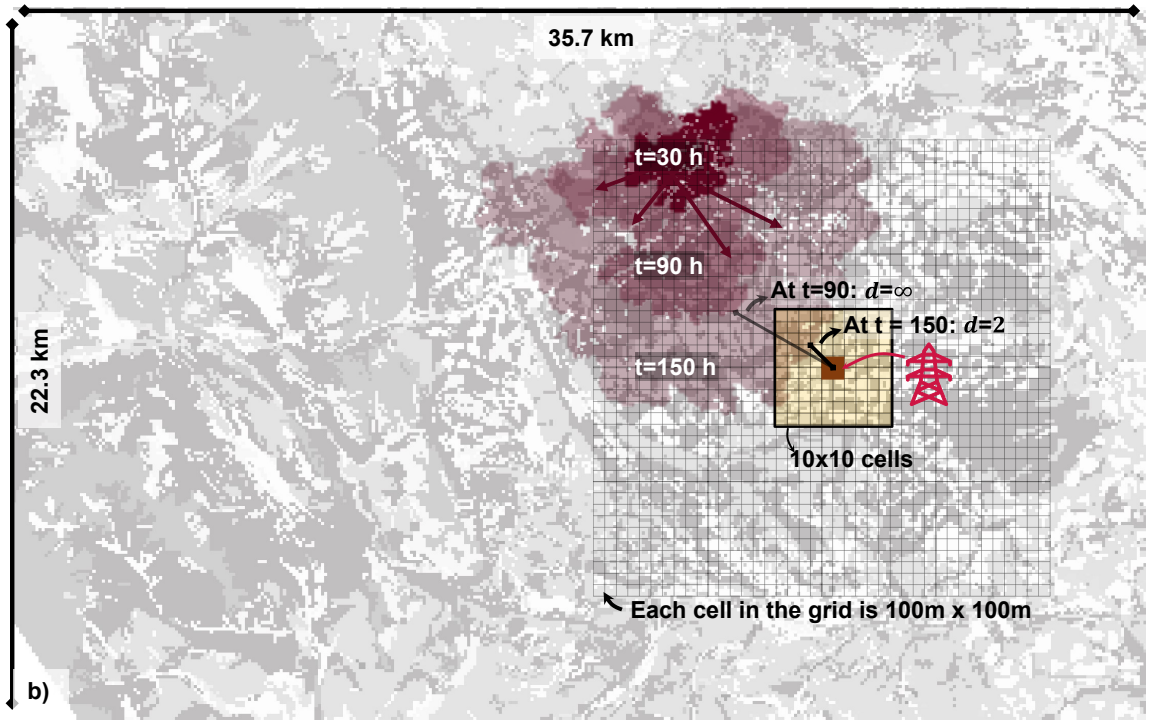


Figure 5.5: The Dogrib Creek, CA map illustrated using the Cell2fire simulator. a) Our sensor suite placement and six different ignition points. b) An example wildfire trace that shows the spread of the fire in three time instants. The wildfire gets within 10 cells of the sensor suite after around 100 hours.

couple the fire simulations from the RL environment. Thus, we use Cell2fire to generate various wildfire scenarios in a batch and store them as trace files for use during training, as outlined above.

5.4.1.2 Energy Harvesting Dataset

Our framework can use any available energy harvesting data. In this work, we assume the sensor suite has a solar energy harvester. To generate a dataset for solar energy harvesting, we use pvlib, a well established tool for simulating the

performance of photovoltaic (PV) energy systems developed at Sandia labs [107]. Using this tool, we combine real hourly solar irradiance data from 2015 with a realistic electrical model [108] of a 0.1m^2 PV-Cell (Uni-Solar US-5). As a result, we obtain a year-long hourly energy harvesting data with seasonal changes (8760 hours in total).

5.4.1.3 Choice of Sensors, Microcontroller and Battery

Numerous sensors can be incorporated into a sensor suite, such as temperature, humidity, pressure, wind, particle sensors, thermal camera, and others. In this work, we use three widely adopted meteorological sensors: Temperature sensor [109], particle sensor [110], and wind speed/direction sensor [111]. We choose these three as they represent different levels of energy consumption and different levels of sensitivity to fire (e.g., particle sensor's range is higher than temperature sensor). However, we emphasize that the proposed framework works with any arbitrary type and number of sensors. We use LoRa as the communication protocol [2]. For the microcontroller, we use the STM32WLE5 series, a system-on-chip that accommodates an ARM Cortex M4 CPU and a LoRa radio on the same package [112]. Finally, we use a 10 Ah @ 3.3 V LiPo battery with a 5% annual capacity degradation [57]. The values of the parameters are listed in Table 5.2.

5.4.1.4 State space, action space, and reward function

The RL environment for wildfire monitoring CPS is designed with generality in mind to enable any reinforcement learning technique.

Table 5.2: Parameter values. S_1: Temperature sensor, S_2: Particle sensor, S_3: Wind sensor. All power values are in mW. P^S and T^S are the power consumption and response time of the sensors that are provided in their datasheets. We ignore the effects of T^W since we assume MCU wakes up instantly from idle state (i.e., we do not use the stand-by mode). We assume transmission is one way and no acknowledgment is expected (i.e., T^{Rx}=0). We assume 4 bytes per sample for S_1 and S_2, and 8 bytes per sample for S_3. Using these, T^{Tx} is calculated (1.63 ms/byte [2]). Finally, we assume processing time T^P for these sensors is negligible owing to their small data sizes.

	M	k	N	T^W	T^S	T^P	T^{Tx}	T^{Rx}	P^S
$\mathbf{S_1}$	1	2	0	0	5 s	0	6.5 ms	0	4 mW
$\mathbf{S_2}$	1	0.7	0	0	4 s	0	6.5 ms	0	250 mW
$\mathbf{S_3}$	1	0.3	0	0	2s	0	13.0 ms	0	11 mW

	P^{Tx}	P^{Rx}	P^{MCU}_{active}	P^{MCU}_{idle}
MCU/LoRa	92.4 mW	0	12.4 mW	1 μW

	T	α	η	T^{Steps}	$*\|\mathcal{D}\|$	$*\|\mathcal{B}\|$	lr
Others	150 hrs	1	1	10^6	10^6	128	10^{-4}

*|.| denotes the size. lr denotes the learning rate.

State Space: The state space is a 14-tuple $\mathcal{S} \subseteq \mathbb{R}^{14}$ that consists of:

- *Current battery energy* ($\frac{E^B_t}{E^B_{max}} \in [0,1]$)*:* The energy level of the battery at the beginning of the current step t divided by the battery capacity.

- *Harvested energy in the previous time step* ($\frac{E^H_{t-1}}{E^B_{max}} \in [0,1]$)*:* Harvested energy during the previous step t−1 divided by the battery capacity.

- *Cumulative EH* ($\sum_{\tau=0}^{t-1} E^H_\tau \in \mathbb{R}$)*:* Cumulative harvested energy in the previous time steps.

- *Initial battery energy level* ($\frac{E_0^B}{E_{max}^B} \in [0,1]$*:* The energy level of the battery at the beginning of the episode ($t=0$) divided by the battery capacity.

- *Target battery energy level* ($\frac{E_T^B}{E_{max}^B} \in [0,1]$)*:* The desired energy level of the battery at the end of the episode ($t = T$) divided by the battery capacity.

- *Actions in the previous step* ($\mathbf{a}_{t-1} \in [-1,1]^3$)*:* The sampling rates assigned to the three sensors in the previous step $t-1$.

- *Sensor readings in the previous step* ($\hat{\mathbf{S}}_{t-1} \in [0,1]^3$)*:* The sensor readings of the three sensors in the previous step $t-1$.

- *Moving average of sensor readings* ($\frac{1}{5}\sum_{\tau=t-5}^{t-1} \hat{\mathbf{S}}_{\boldsymbol{\sigma}} \in [0,1]^3$)*:* The moving average of sensor readings in the previous 5 time steps ($t-5$ to $t-1$).

Action Space: The actions are the assigned *hourly sensor sampling rates* at every time step ($\mathbf{a}_t \in [-1,1]^3$). We limit the actions in the $[-1,1]$ interval, and map this interval to $[0,60]$ when calculating the energy consumption of the suite (i.e., -1 maps to 0 samples/hr and 1 maps to 60 samples/hr.) The training performance takes a significant hit if we use the full $[1,60]$ range instead of $[-1,1]$.

Reward function: Our objective is to minimize the difference between the expected sensor readings (i.e., golden values) and the actual sensor readings under battery energy constraints. In an RL setting, the objective and the constraints are imposed by the reward function. In this case, two constraints can be imposed on the reward function: (i) minimum battery level constraint and (ii) target battery level constraint. We do not include the maximum battery level constraint because under-utilizing

the energy implicitly decreases the objective. Considering the objective and the constraints on the battery, the reward function becomes:

$$r_t = \begin{cases} -100 & E_t^B \leqslant E_{min}^B \\ \alpha\left(\frac{E_t^B - E_T^B}{E_{max}^B}\right) - |\hat{S}(\mathbf{a}, d) - \mathbb{E}[S(d)]| & otherwise \end{cases} \tag{5.12}$$

Here, we impose the minimum battery energy level by punishing the training heavily (-100 reward) if this constraint is violated. Otherwise, we impose the target battery level constraint using the first term ($\frac{E_t^B - E_T^B}{E_{max}^B}$). The objective is considered through the second term $|\hat{S}(\mathbf{a}, d) - \mathbb{E}[S(d)]|$. α is a coefficient to scale the weight of the target battery level constraint, i.e., higher α will push the agent to be more conservative and linger around the target battery level.

5.4.1.5 Implementation

We develop our environment in Python and register it as an OpenAI Gym [67] environment using the components explained in this section. When starting training after a reset, the environment randomly chooses a 150-hour slice from the energy harvesting dataset, one wildfire trace, and an initial battery energy level, as summarized in Algorithm 6. During testing, the environment takes the energy harvesting data, wildfire trace, and the initial battery level as inputs for repeatable results. Then, the state vector $\mathcal{S}$ is initialized and the done signal, which shows whether the episode ended, is initialized as False.

At each step, the environment gets the data rates $\mathbf{a}$ and the state vector $\mathcal{S}_{t-1}$ from

Algorithm 6: Pseudocode for the wildfire RL environment

Reset()
  Training: Randomly choose energy harvesting and wildfire traces, and an initial battery level E_0^B
  Test: Take the energy harvesting and wildfire traces, and the initial battery energy as inputs
  Reset the state $\mathcal{S}$ and done $\leftarrow$ False
  Output: $\mathcal{S}$, done
Step()
  Input: Data rate a (i.e., actions), $\mathcal{S}_{t-1}$
  Get E_t^H from the energy harvesting trace
  Get distance to fire d from the wildfire trace
  Use d to get expected sensor readings (Equation 5.1)
  Use **a** and d to get actual sensor readings (Equation 5.3)
  Use **a** to calculate E_t^C (Equation 5.8)
  Calculate E_t^B (Equation 5.9)
  Calculate r_t (Equation 5.12)
  Calculate $\mathcal{S}_t$
  if $E_t^B < E_{min}^B$ ***or*** $d = 0$ ***or*** $t = 150$ **then**
    done $\leftarrow$ True # End of episode
  else
    done $\leftarrow$ False
  Output: r_t, $\mathcal{S}_t$, done

the previous time step as input. Then, it reads the harvested energy for the current time step E_t^H from the energy harvesting data. In addition, it also reads the status of the wildfire from the wildfire trace. The environment uses this trace to calculate the distance d between the sensor suite and the fire, as shown in Fig. 5.5-b). When calculating d, sensors have a maximum range beyond which they cannot sense environmental changes. For example, Fig. 5.5-b) shows a case where the sensors are affected only by a wildfire that is within 10 cells of the sensor suite. If the wildfire is outside of this range, we assign the distance to a very high value (e.g., ∞), so that

the sensors read nominal values. If the wildfire is inside the range, we calculate the distance from the closest point of the wildfire to the sensor suite. In the example presented in Fig. 5.5-b), at t=30h and t=90h, the wildfire has not reached within 10 cells of the sensor suite and thus the distance is set to ∞. In contrast, at t=150h, the closest point of the wildfire to the sensor suite is within 10 cells, so the distance is now calculated as d=2. Next, the environment uses d, **a**, and $\mathcal{S}_{t-1}$ to calculate the expected and actual sensor readings, the energy consumption of the sensor suite, the battery level at the end of the step, and the reward for this step. Finally, it updates the state vector and outputs the reward, the state vector and the done signal. An episode terminates (i.e., $done \leftarrow True$) if the battery is depleted or if the wildfire reaches the sensor suite (i.e., d=0) or if time step reaches 150.

5.4.2 Proposed TD3 Framework

The proposed framework trains an RL agent by interacting with the wildfire CPS environment. The trained agent is a policy that observes the state vector and yields actions (i.e., data rates **a**) that maximize the Q value. Algorithm 7 summarizes the proposed TD3 framework. First, we initialize an empty replay buffer $\mathcal{D}$ and critic and actor networks with random weights. We use a multi-layer perceptron with three hidden layers and 64 neurons in each layer within these networks. We also initialize the target critic and actor networks with the same parameters. Then, the framework starts training. At each episode, we reset the environment to choose new energy harvesting and wildfire traces, and an initial battery energy level. The agent then interacts with the environment using its actor-network (π_ϕ). The transitions

collected $(\mathcal{S}_t, \mathbf{a}_t, r_t, \mathcal{S}_{t+1}, done)$ are then stored in the experience replay buffer $\mathcal{D}$. Then, the algorithm samples a minibatch $(\mathcal{B})$ of transitions from $\mathcal{D}$. Using this minibatch of samples, the smoothed target action values are computed according to the noise ϵ and the clipping factor c. The target value y is computed using the target critic-networks and target action values. The actor and critic networks are then updated using their corresponding loss functions. The training terminates when number of time steps reaches T^{Steps}. The hyperparameters for our approach are presented in Table 5.2.

Algorithm 7: TD3 framework for RL training

Initialize: Replay buffer $\mathcal{D}$, Critic networks $\mathbf{Q}_{\theta_1}$,$\mathbf{Q}_{\theta_2}$ and actor network π_ϕ with parameters θ_1, θ_2, ϕ, Target networks $\mathbf{Q}_{\theta_1'} \leftarrow \mathbf{Q}_{\theta_1}$, $\mathbf{Q}_{\theta_2'} \leftarrow \mathbf{Q}_{\theta_2}$, $\pi_{\phi'} \leftarrow \pi_\phi$

$done \leftarrow$ True

for $t = 0$: T^{Steps} **do**

  if $done =$ True **then**

    $\mathcal{S}_t$, done $\leftarrow$ **Reset**()

  Observe state $\mathcal{S}_t$ and take actions: $\mathbf{a}_t \leftarrow \pi_\phi(\mathcal{S}_t)$

  r_t, $\mathcal{S}_{t+1}$, done $\leftarrow$ **Step**()

  Store the transition ($\mathcal{S}_t$,$\mathbf{a}_t$,r_t,$\mathcal{S}_{t+1}$,done) in $\mathcal{D}$.

  Sample random B transitions from $\mathcal{D}$;

  $\tilde{\mathbf{a}}_t \leftarrow \pi(\mathcal{S}_{t+1}, \phi') + \epsilon : \epsilon \sim clip(\mathcal{N}(0, \sigma), -c, c)$

  $y \leftarrow r + \gamma \arg_Q \min_{i=1,2} Q_{\theta_i'}(\mathcal{S}_{t+1}, \tilde{\mathbf{a}}_t)$

  $Loss_{critic}(\theta_i) = \mathbb{E}\Big[\Big(y - Q_{\theta_i}(\mathcal{S}_t, \mathbf{a}_t)\Big)^2\Big]$

  $Loss_{actor}(\phi) = \mathbb{E}\Big[\ Q_{\theta_1}(\mathcal{S}_t, \mathbf{a}_t)|_{\mathbf{a}_t = \pi(\mathcal{S}_t;\phi)}\Big]$

  Update critic, actor, and target network parameters.

5.5 Experimental Evaluations

This section presents the experimental evaluation of the proposed framework. It first describes the evaluation scenarios. Then, it introduces a class of heuristic algorithms as a baseline since there are no alternative techniques in the literature. Finally, it presents the evaluation results and execution time, energy overhead and memory footprint measurements of the proposed framework when deployed on the TI CC2652R MCU [113].

5.5.1 Experimental Setup

5.5.1.1 Evaluation Scenarios.

We evaluate the proposed CPS with five wildfire traces that were generated as part of the 6000 traces as explained in Section 5.4.1.1. *These five traces are excluded from the training process* explained in Section 5.4.2. During testing with each of the five traces, we use six different ignition times for long term simulations: (6 months, 1 year, 2 years, 3 years, 4 years and 5 years), leading to a total of 30 combinations. For example, one of the combinations is to use the first trace with the first ignition time, which runs the simulation with *no-fire* conditions for 24 weeks, and the ignition happens in the 25th week (after 6 months). We use a fixed 150-hour long energy harvesting data for each week, such that each approach is evaluated fairly.

Evaluation metrics: Since a wildfire can occur at any time, the proposed system *should* preserve energy under uncertainty and leverage this energy to minimize sensor reading error when a wildfire danger arises. Using this observation, we

extract three quantities from these simulations: i) The cumulative number of inactive hours where the battery was depleted, ii) the cumulative sampling error, and iii) the initial response time to a wildfire, which is defined as the time delay between the emergence of the wildfire and the first measurement after that. If the initial response time is too long, the fire is detected late, leading to *catastrophic* consequences.

Ideal Case: The ideal case should have no inactive hours (i.e., battery is never depleted), the cumulative sampling error is minimized and the initial response time is minimized. However, we emphasize that achieving this is challenging since the CPS does not know when the fire takes place and the stored and harvested energies are limited and varying.

5.5.1.2 Baseline Heuristic Approach.

Since no similar technique exists in the literature, we developed a hierarchical heuristic as the baseline, as summarized in Algorithm 8. This heuristic first sorts the sensors according to their energy consumption in ascending order (line 5). Then, it uses an harvested energy predictor for predicting the battery energy in the next interval (lines 6-7). The energy predictor is obtained by averaging the energy harvested at each hour in a day over the 365 days in the dataset. As a result, we obtain an estimator for each hour in a day. Using this predictor, the heuristic calculates the battery level for the next interval.

If the predicted battery level exceeds 75%, the heuristic allocates 30 samples/hr to the sensor with the lowest energy consumption and 10 samples/hr to the remaining

Algorithm 8: Pseudocode for the heuristic.

```
1  Input: List of sensors: sensors = [S_1, S_2, ..., S_n],
2  threshold values γ ∈ ℝ^n, sensor readings Ŝ,
3  constant sampling rates c, time of day t and
4  battery level E^B_t
5  sensors ← Sort(sensors)
6  Ê^H_t ← predictEnergy(t)
7  Ê^B_{t+1} ← E^B_t + Ê^H_t
8  # Assign according to predicted battery level
9  if Ê^B_{t+1} > 75% then
10     sensors[1].setSamplingRate(30)
11     sensors[2...n].setSamplingRate(10)
12 else if Ê^B_{t+1} < 25% then
13     sensors[1].setSamplingRate(15)
14     sensors[2...n].setSamplingRate(0)
15 else
16     sensors[1...n].setSamplingRate(c_1)
17 # Then check for wildfire occurrence
18 if Ŝ_1 > γ_1 then
19     sensors[1].setSamplingRate(c_2)
20     sensors[2].setSamplingRate(c_2)
21     if Ŝ_2 > γ_2 then
22         sensors[1].setSamplingRate(c_3)
23         sensors[2].setSamplingRate(c_3)
24         sensors[3].setSamplingRate(c_3)
25         if Ŝ_3 > γ_3 then
26             # Goes on like this ...
```

sensors (lines 9-11). Conversely, if the predicted battery level falls below 25%, the sensor with the lowest energy consumption is assigned 15 samples/hr while the other sensors remain in a sleep state (lines 12-14). If the predicted battery level lies between 25% and 75%, the sensors are assigned a constant sampling rate, denoted as c_1 (lines 15-16). Finally, the heuristic checks the sensor readings for a potential

wildfire occurrence (lines 18-26). If the reading of the first sensor exceeds the threshold γ_1, the heuristic wakes up the next sensor in line and sets the sampling rates of both active sensors to c_2. Then, if the reading of the second sensor exceeds γ_2, the next sensor wakes up, all three are set to c_3, and so on. This way, the heuristic overrides the low rates set by the low-battery level condition.

We perform an extensive sweep of 285 different combinations of c_1,c_2,c_3 values for this heuristic, with the complete list provided in Table C.1 of the Appendix. From these sets, we selected two representative variants to present here, showing distinct levels of "aggressiveness" as depicted in Table 5.3. The first variant, the balanced heuristic, employs a sampling rate of 15 samples/hr for c_1, ensuring that it maintains a moderate energy consumption level most of the time. It slightly increases the sampling rates for potential fire situations to 21 samples/hr for c2 and 27 samples/hr for c3. The second variant, referred to as the aggressive heuristic, adopts sampling rates of 33 samples/hr or higher for all scenarios. This approach prioritizes maximum data collection, disregarding potential energy constraints.

Table 5.3: Three heuristics with different energy consumption aggressiveness. Conservative uses very little energy all the time. Balanced uses less energy most of the time. Aggressive uses high energy most of the time.

	Conservative	**Balanced**	**Aggressive**
γ_1	0.001	0.001	0.001
γ_2	0.3	0.3	0.3
γ_3	2	2	2
c_1	1 S/hr	15 S/hr	33 S/hr
c_2	1 S/hr	21 S/hr	39 S/hr
c_3	1 S/hr	27 S/hr	45 S/hr

Furthermore, we implemented a conservative heuristic that consistently employs the minimum sampling rate of 1 sample/hr for all sensors (i.e., c_1, c_2, c_3).

5.5.2 Performance Evaluation

5.5.2.1 Short-term Analysis: Weekly Behavior

This section evaluates the proposed CPS design based on one-week simulations, focusing on its behavior during a wildfire that approaches the sensor suite on the final day of the week, as depicted in the upper left plot of Fig. 5.6. All algorithms utilize the same wildfire traces and harvested energy, but the RL agent starts with a considerably lower battery energy to showcase its superiority even under more challenging conditions. The sensor data rate plots in the second row of Fig. 5.6 show that the conservative heuristic consistently employs low sampling rates throughout the week. This choice helps preserve the battery energy, as shown in the top right plot. However, it fails to accurately track the environmental conditions after the wildfire arrives, as demonstrated by the plots in the bottom row. The carefully tuned balanced heuristic utilizes the sensors more effectively by activating sensors 2 and 3 only when the battery level exceeds 25%. Additionally, it increases the sampling rate for all sensors after the wildfire arrives (as evident in the second row of plots). Nevertheless, it operates the sensors more than needed, wasting energy. As a result, it depletes the battery towards the end of the week and eventually shuts down. This behavior highlights the inability of static heuristics to guarantee robust operation in dynamic and uncertain conditions. Similarly, the aggressive heuristic quickly drains the battery even before the wildfire reaches the sensor suite. In strong

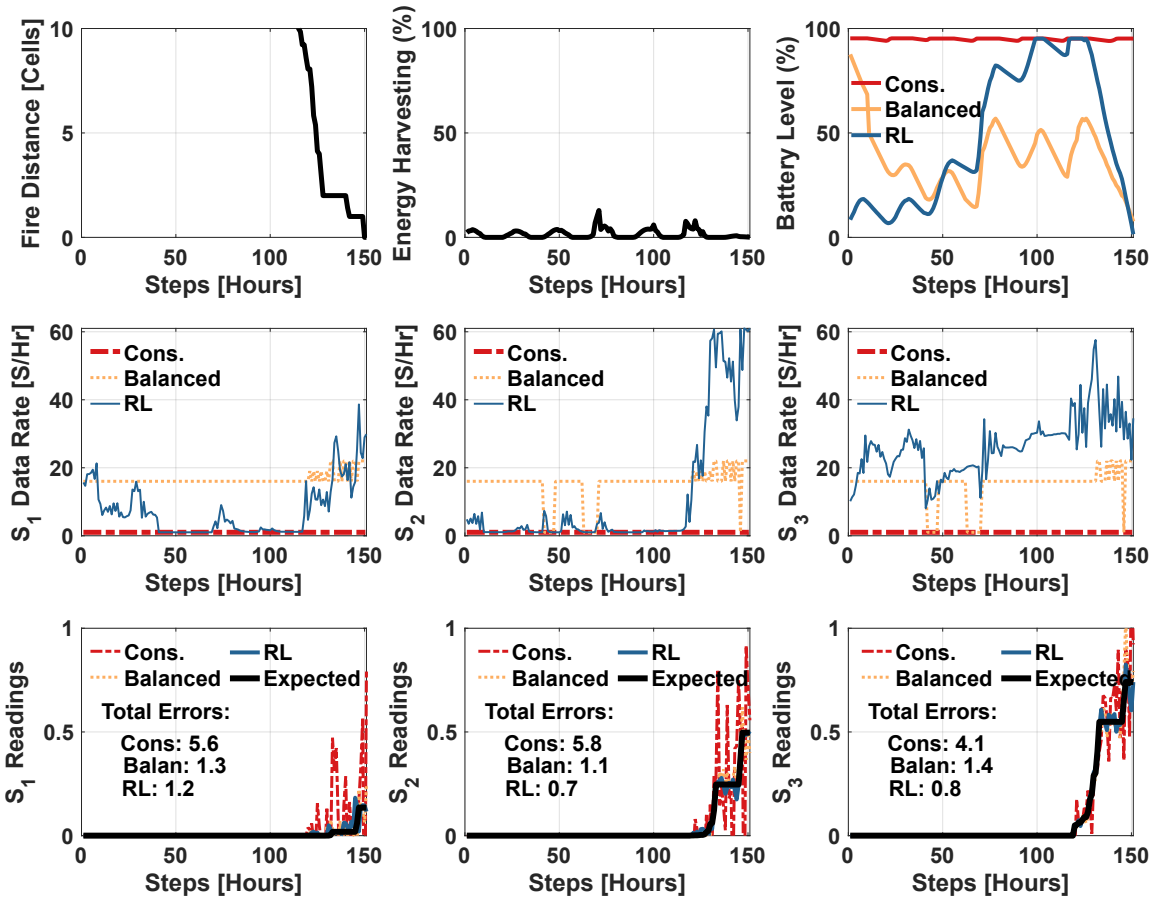


Figure 5.6: Weekly behavior of conservative and balanced heuristics and the RL agent. The RL agent achieves 83% and 29% less error than the conservative and balanced heuristics, respectively.

contrast, our RL agent dynamically controls the sensor data rates, co-optimizing monitoring accuracy and battery energy. It actively uses all sensors to guard against environmental changes (the second row), while simultaneously replenishing the battery (top right plot). *Despite starting with a significant disadvantage in terms of initial battery level, the RL agent outperforms the heuristics in terms of average sampling rate and total accumulated errors, as illustrated in the second and third rows. Specifically,*

the average sampling rate is 1.9× higher, and the total error is 29% smaller than the balanced heuristic.

5.5.2.2 Long-term Analysis: 1-year and 5-year Long behavior

In this section, we conduct evaluations of the proposed CPS design over durations of 1 year and 5 years. Similar to the previous section, each algorithm employs identical wildfire traces and harvested energy. However, in this case, all approaches (including the three heuristics and the RL agent) start with an initial battery level of 90%, as we argue that this represents a typical scenario for initial deployment. Fig. 5.7 compares the conservative, balanced, and aggressive heuristics against the RL agent over a 1-year simulation. The first row illustrates the evolution of the battery level for each approach throughout the simulation, based on one wildfire trace. The conservative heuristic maintains the battery level near-maximum capacity, as it consistently employs low sampling rates. Conversely, the balanced and aggressive heuristics often experience battery depletion, as indicated by the fluctuations in their battery levels. This behavior is illustrated in the second row, where each approach's cumulative number of inactive hours (i.e., depleted battery) is plotted. Specifically, the balanced and aggressive heuristics deplete the battery 41% and 55% of the time, respectively. *In strong contrast*, the RL agent *never* depletes the battery due to its capability to recover from low-battery conditions, as demonstrated in the previous section. Finally, the third row illustrates the distribution of the initial response times for each approach across the five traces. The conservative heuristic has a fixed response time of 60 mins. The balanced heuristic has a median response

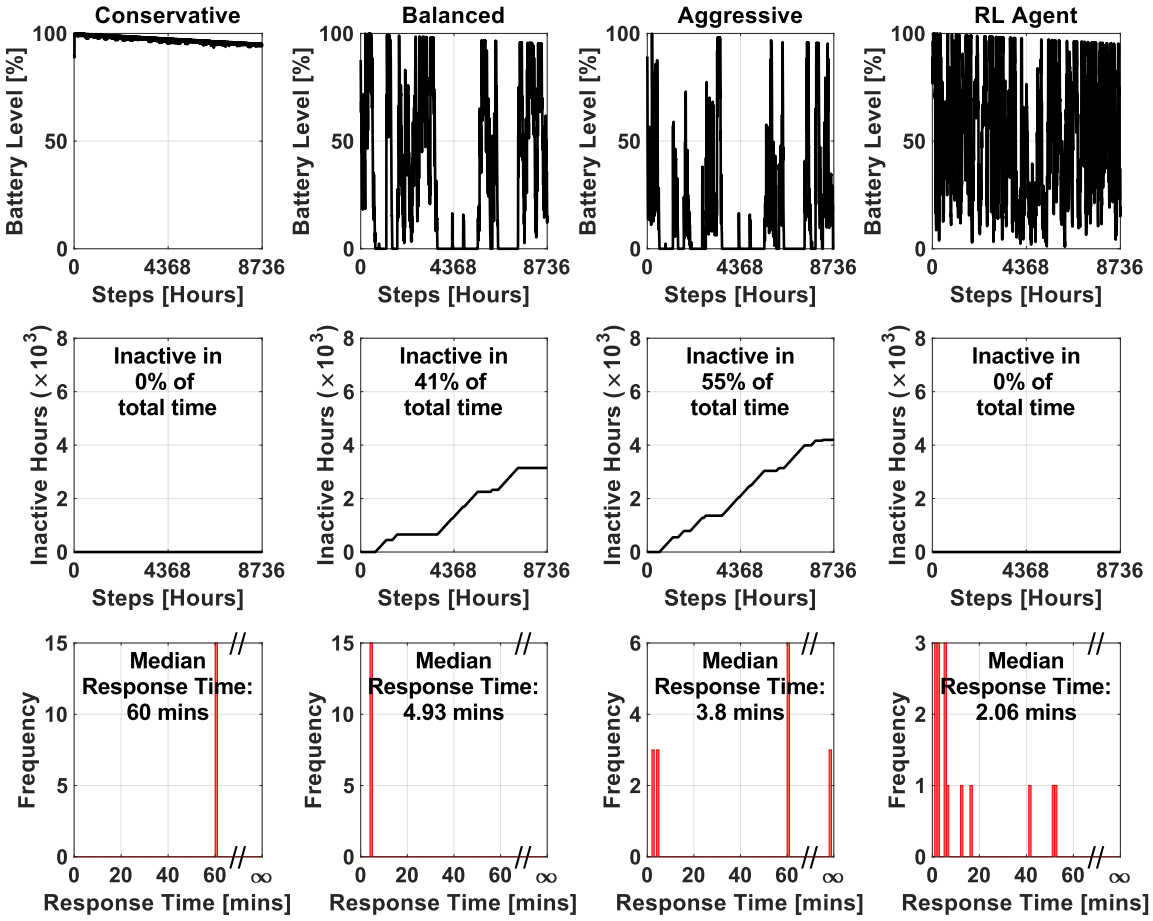


Figure 5.7: Comparison of 1-year long simulation results for the conservative, balanced, aggressive heuristics and the trained RL agent.

time of 4.93 mins, whereas the aggressive heuristic has 3.8 mins. However, the aggressive heuristic often dies out in the final week (after the fire onset), yielding ∞ response time when this happens, as the device shuts down and the sampling rate is set to 0 c2. *In contrast*, the RL agent has a shorter response time compared to heuristic approaches, with a median response time of 2.06 minutes, which is 2.5$\times$ faster than the balanced heuristic. Notably, the RL agent achieves this faster response time while *never* depleting the battery.

Fig. 5.8 compares the conservative, balanced, and aggressive heuristics against the RL agent over a 5-year period. As expected, the conservative heuristic exhibits similar behavior to the 1-year results. The performance of the balanced heuristic deteriorates, evidenced by the increased inactive time of 57% (from 41%). Furthermore, it *consistently* shuts down in the final week, resulting in ∞ response time. The aggressive heuristic is inactive for 66% of the time and always ceases to

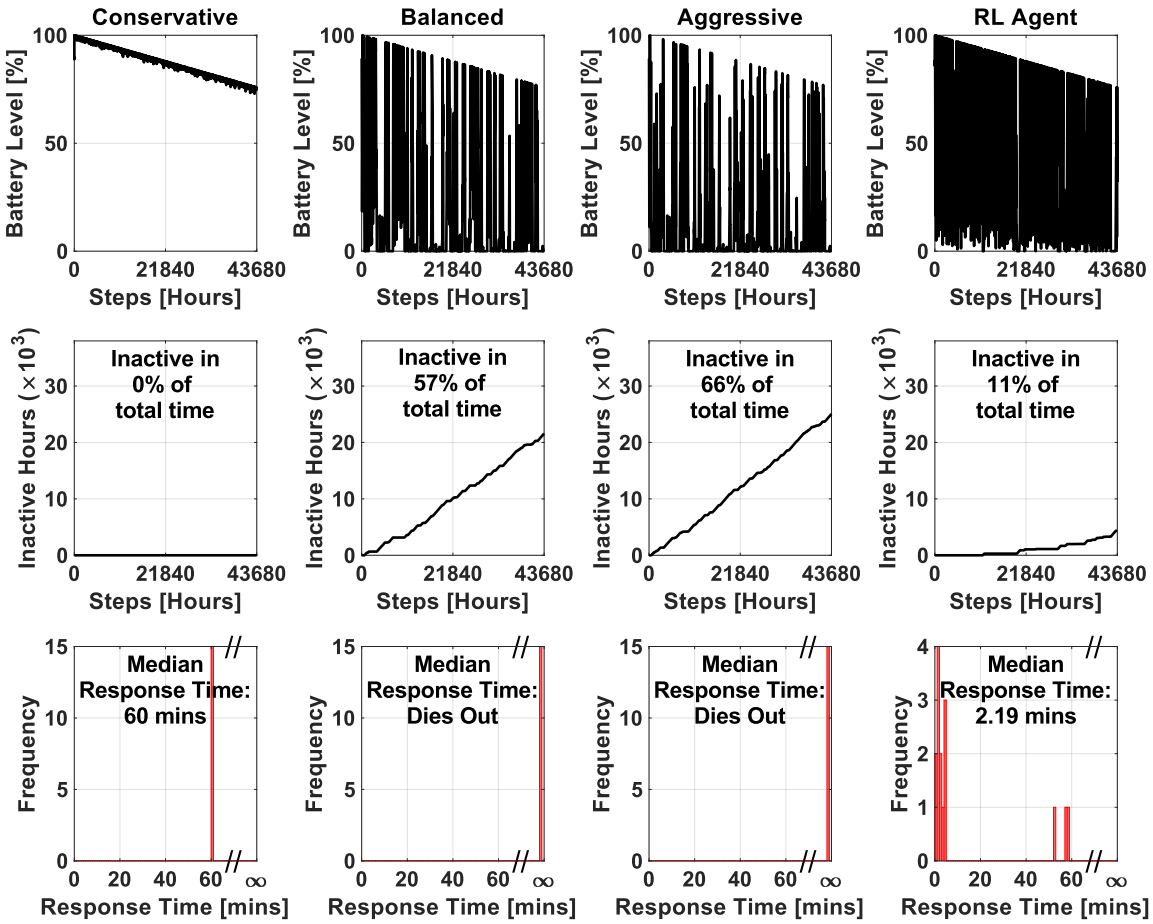


Figure 5.8: Comparison of 5-years long simulation results for the conservative, balanced, aggressive heuristics and the trained RL agent.

operate before the final week. Consequently, both heuristics fail to collect any data during the wildfire period. In contrast, the RL agent significantly outperforms the heuristics, with only 11% of the time being inactive. The increase in inactive hours compared to the 1-year case can be attributed to the 5% annual battery degradation, which becomes more pronounced over the 5-year duration. As the battery capacity diminishes, the RL agent can no longer maintain its flawless record from the previous case. Notably, the inactive hours begin to increase after approximately 21,000 hours (around 30 months). Despite this, the RL agent achieves a median response time of 2.19 minutes and avoids battery depletion in the final week.

5.5.3 Energy Consumption Evaluation on Real Hardware

We deployed the trained RL agent using TensorFlow Lite for Micro (TFLM) flow [71] on the TI CC2652R MCU. The MCU has an ARM Cortex M4F running at 48 MHz and has 352 KB of flash memory and 80 KB of SRAM. The execution time and energy consumption overhead of a single policy network call are measured as 2 ms and 25 µJ, respectively. There are 24 network inference calls daily (i.e., once every hour). Therefore, the daily energy consumption is 0.6 mJ, which is negligible compared to the daily energy consumption of the sensor suite. In addition, the policy network has 13635 parameters. Considering these parameters, the TFLM operators, inputs, outputs, and intermediate values, the total memory footprint of the proposed design is less than 200 KB, which easily fits into the onboard memory. Therefore, deploying and executing the RL agent on the target MCU does not compromise achieving self-sustainable operation.

6 DTRL: DECISION TREE-BASED MULTI-OBJECTIVE REINFORCEMENT LEARNING FOR RUNTIME TASK SCHEDULING IN DOMAIN-SPECIFIC SYSTEM-ON-CHIPS

6.1 Motivation and Contributions

The growing demand for high-performance and energy-efficient processing in various domains, including machine learning, image and video processing, and wireless communication systems, has led to the rise of domain-specific system-on-chips (DSSoCs) [114, 115]. DSSoCs combine specialized hardware accelerators and general-purpose cores to enhance the performance and energy efficiency of a target domain while providing programming flexibility [116, 117, 118]. For instance, DSSoCs designed for image and video processing are often equipped with specialized hardware accelerators like digital signal processors (DSPs) and image signal processors (ISPs), while those designed for wireless communication systems incorporate processing elements (PEs) such as fixed-function accelerators for fast Fourier transform (FFT), encoding, and Viterbi decoding operations.

DSSoCs comprise several heterogeneous PEs, enabling parallel execution of multiple streaming applications [119, 6]. Scheduling a large number of tasks from concurrent applications is a monumental challenge due to the NP-complete nature of the problem and a large number of design and runtime parameters [120, 121]. While static schedulers rely solely on design-time information, dynamic scheduling approaches leverage runtime data to make more informed decisions [122, 123].

Conventional optimization-based approaches can achieve near-optimality but are highly prohibitive due to their complexity and runtime overheads [124]. Heuristics are frequently employed to tackle the complexity challenge, but they are typically designed for specific use cases, lack generalizability, and often fall short of the optimal solution [122]. Certain machine learning approaches for cluster scheduling [125] and task scheduling [6] are also explored in the literature. However, they suffer from limitations that include sub-optimality, high runtime overheads, and substantial developmental/training efforts. Furthermore, they focus on a single optimization objective, e.g., maximizing performance or minimizing power consumption, but not simultaneously. In contrast, DSSoCs require schedulers that make near-optimal decisions considering competing objectives within nanoseconds to be on par with the task execution times in specialized PEs.

Reinforcement learning (RL) has shown promise in addressing several challenging problems, including intelligent chatbots [126], healthcare [127], autonomous driving [128], and scheduling [129, 130, 125]. RL trains a policy that maximizes the reward function by interacting with an environment. During the training, RL algorithms learn optimal decisions using the current state of the system and the desired performance objectives. Real-world problems often include multiple objectives that may conflict with each other. In contrast to single-objective environments, the performance of such problems is evaluated using multiple objectives. Therefore, multiple Pareto-optimal solutions may exist depending on the preference between objectives [24]. Multi-objective reinforcement learning (MORL) approaches [23] address this challenge by maximizing a *vector of rewards* instead of a scalar reward.

Existing MORL methods transform the multidimensional objective space into a single dimension using per-objective static weights and then employ standard RL algorithms to obtain a policy optimized for those weights [29]. However, a distinct policy for each objective is impractical due to storage and design-time requirements, necessitating a set of Pareto front solutions with a single policy. Therefore, obtaining a set of Pareto front solutions that covers the entire preference space with a single training is crucial [4, 1, 30, 25].

Considering the requirements of DSSoCs and the strengths of MORL techniques, we develop the following insights that help us design a runtime task scheduling framework:

Key Insight 1: We can use RL to effectively explore the vast solution space and address the sub-optimality challenges of heuristic approaches and the complexity of optimization-based techniques.

Key Insight 2: We can exploit MORL techniques to jointly optimize for conflicting optimization objectives in DSSoCs, such as maximizing performance and minimizing power consumption.

Key Insight 3: We can combine the benefits of the low inference overheads of decision tree classifiers with MORL to design runtime scheduling policies that co-optimize multiple objectives while incurring minimal inference latency overheads.

This work presents DTRL, a decision-tree-based multi-objective reinforcement learning technique for runtime task scheduling in DSSoCs. DTRL uses a multi-objective variant of the proximal policy optimization (PPO) [131] algorithm and a differentiable decision tree (DDT) policy. We adopt a DDT policy since decision

trees provide significantly lower inference latency overheads due to fewer computations than commonly used neural networks, such as multi-layer perceptrons and convolutional neural networks. To efficiently train an RL algorithm for runtime task scheduling, we also developed a novel RL environment for DSSoCs, utilizing an open-source DSSoC simulator [132] as the foundation of this environment. We also note that our RL environment supports any DSSoC simulation framework. The proposed DTRL framework trains the DDT policy by interacting with this environment to maximize a joint objective function weighted by a preference vector. Furthermore, the framework trains the policy with several preferences enabling it to produce an optimal policy for any preference vector specified at runtime.

DTRL is evaluated with six wireless communication and radar systems domain applications. We employ a complex DSSoC configuration comprising sixteen PEs, including Arm big.LITTLE cores and fixed-function energy-efficient accelerators for matrix multiplication, fast Fourier transform, and Viterbi decoding. We compare DTRL with heuristic– [5], optimization–, and machine learning–based [6] schedulers. Extensive experimental evaluations using our novel reinforcement learning environment demonstrate DTRL achieves high performance on par with performance-optimized integer linear programming (ILP) solution and state-of-the-art heuristic-based scheduler ETF [5] when the highest preference is given to execution time. At the same time, the same DTRL policy achieves up to $3\times$ lower energy consumption than these schedulers when higher preference is given to power consumption. Similarly, DTRL also achieves similar energy consumption to power-optimized schedulers when the highest preference is given to the power

consumption objective while outperforming them by up to 9× lower execution time when the preference is given to the execution time objective. We also implement DTRL on Xilinx Zynq ZCU102 FPGA running at 1.2 GHz, using an open-source emulation framework [133], and measure its runtime overhead. Hardware evaluations show that DTRL has 120 ns inference latency, resulting in negligible runtime overhead. Evaluation on the same hardware shows that DTRL achieves up to 16% higher performance than the state-of-the-art heuristic-based scheduler ETF [5]. We emphasize that DTRL successfully learns the trade-off between execution time and power consumption and *achieves a Pareto set of solutions that covers the entire preference space using a single DDT policy.* To the best of our knowledge, *this is the first decision-tree-based multi-objective reinforcement learning framework for runtime task scheduling in DSSoCs.*

The main contributions of this work are as follows:

- The DTRL framework, a decision-tree-based multi-objective reinforcement learning approach for runtime task scheduling in DSSoCs,
- A novel reinforcement learning (RL) environment for DSSoCs, utilizing a DSSoC simulation framework,
- Extensive experimental evaluations of the proposed DTRL framework demonstrating performance and energy consumption improvements against state-of-the-art heuristic–, optimization–, and machine learning-based schedulers, and

- Hardware emulation platform measurements for comparisons against a state-of-the-art heuristic scheduler and runtime overhead analysis.

6.2 Related Work

Runtime task scheduling in DSSoCs is crucial to exploit their full potential but is highly challenging due to several factors. First, the heterogeneous PEs in an SoC provide diverse power and performance characteristics, thereby increasing the decision-making complexity. Second, the parallelism offered by DSSoCs allows several applications to execute in parallel. The simultaneous application execution requires highly effective runtime decision-making to utilize the PEs and maximize performance and energy efficiency. Scheduling decisions must incur negligible latency and energy overheads since the tasks can execute in the order of nanoseconds in DSSoCs. Finally, different applications in a target domain may have contrasting requirements that require schedulers to dynamically support multiple and contrasting objectives. This work addresses all these aspects, and hence, we classify prior work into the following categories and summarize them in Table 6.1.

Scheduling Techniques: Most scheduling approaches in literature use directed flow graphs (DFGs) to model applications [122, 120, 140, 145]. Optimization-based techniques for DFG scheduling, such as integer linear programming (ILP) and constraint programming (CP) [134, 135, 136], provide optimal design-time solutions. However, these techniques have significant drawbacks. First, optimization-based approaches struggle to derive the optimal decisions in reasonable runtimes for

Table 6.1: Comparison of the proposed DTRL framework with prior approaches on the basis of desired metrics such as optimality, runtime overheads, ability to support multiple objectives, and capability to adapt online.

Prior Approaches	Optimality	Complexity and Runtime Overheads	Multi-Objective Support	Online Adaptability
Optimization Techniques [134, 135, 136]	High	Very High	No	No
Static Techniques [122, 120, 137]	Low	Low	No	No
Dynamic Techniques [138, 139, 140, 6]	Moderate	High	No	No
RL Approaches [141, 130, 125, 129]	High	High	No	Yes
Multi-Objective Techniques [142, 143, 144]	Moderate-High	High	Yes	No
DTRL (Proposed approach)	**High**	**Low**	**Yes**	**Yes**

complex scenarios such as DSSoC scheduling due to a large number of variables and constraints. Secondly, it is highly impractical to re-run the optimization approach for every possible system state, considering factors like PE utilizations, injection rate, workload, and SoC configuration. Thirdly, deploying optimization-based approaches for runtime scheduling is not feasible due to the scheduling overheads and computational resource requirements involved. Despite the existence of Pareto front optimization approaches, they cannot be employed for runtime task scheduling in DSSoCs due to the aforementioned challenges.

The heuristic class of scheduling algorithms trades off runtime with optimality. List scheduling techniques populate the DFG nodes in a list and schedule the tasks to PEs at design time [146, 137, 147]. While design time techniques are suitable for small problem sizes that involve sequential execution, they are insufficient to handle the complexity posed by streaming simultaneous application execution in DSSoCs [148, 6]. Dynamic scheduling techniques exploit the available runtime

information to make more effective decisions [138, 139, 140]. However, they are highly sub-optimal and designed for specific objectives. Furthermore, they still incur high runtime overheads, leaving significant scope for improvements.

The advent of machine learning has led to novel scheduling approaches [6, 141, 149, 130, 125]. Imitation learning (IL) approaches such as [149, 6] achieve low latency overheads but suffer from sub-optimality. IL techniques also lack the ability to adapt to the workload and platform configuration changes. RL-based approaches are highly promising due to the rapid algorithm development in this field, which we review next.

Reinforcement Learning: RL has emerged as a promising approach for solving complex problems and exploring large solution spaces, including runtime task scheduling in DSSoCs. However, prior RL-based approaches have several limitations, including being unable to run on heterogeneous SoC architectures, having high training complexity, and exhibiting high runtime overheads due to unnecessarily complex algorithmic structures [130, 125, 129]. Furthermore, they have shown limited performance on high-intensity workloads and only support a single objective, making them unsuitable for adapting to various objectives [150]. Therefore, developing RL-based approaches that deliver low overheads, efficiently support multiple objectives, and overcome these limitations is crucial.

Multi-Objective Dimensions to Scheduling and RL: The desired objectives and metrics of task scheduling, such as power, performance, quality of service, reliability, and energy consumption, often conflict with each other. Furthermore, different applications have varying requirements that need schedulers to support

these multiple and contrasting objectives. Therefore, multi-objective optimization support is vital for task scheduling models in DSSoCs as they require schedulers that can make optimal decisions while considering multiple competing objectives in runtime. The multi-objective aspect of task scheduling has always triggered interest since it can help balance competing objectives and optimize for multiple metrics simultaneously. Optimization, genetic, evolutionary, and heuristic algorithms such as [142, 143, 144] optimize for multiple objectives but suffer from complexity and overhead drawbacks, similar to their single-objective counterparts. Conventional RL algorithms support multiple objectives by designing a unique scheduling policy for each preference vector for multiple objectives [4]. A fine-grained sweep of the preference space can result in a large number of policies, leading to explosive memory requirements [1, 25].

To the best of our knowledge, DTRL is the first DSSoC task scheduling approach that provides superior metrics (maximizing performance and energy efficiency), low runtime overheads using decision tree classifiers, and support multiple optimization objectives using a single policy at runtime.

6.3 Overview and Preliminaries

This section overviews the key components of DTRL. Section 6.3.1 explains the representations of streaming applications as task graphs. Section 6.3.2 discusses the difference between single- and multiple-objective reinforcement learning formulations. Finally, Section 6.3.3 describes the differentiable decision tree used as

the policy in DTRL.

6.3.1 Runtime Task Scheduling

DSSoCs provide numerous processing elements for task execution at runtime. As illustrated in Fig. 6.1(a), a system can be in different states that arise from varying utilization levels. Applications with streaming behavior, where multiple frames are repeatedly injected into the system with different rates, pose significant challenges due to varying conditions such as system configuration, utilization, busy states, and concurrent applications. This work models applications as directed flow graphs (DFGs), as shown in Fig. 6.1(b). Nodes represent the key computational components of applications (also called *tasks*). The edges denote the dependencies between tasks, and the weights of the edges denote the communication volumes

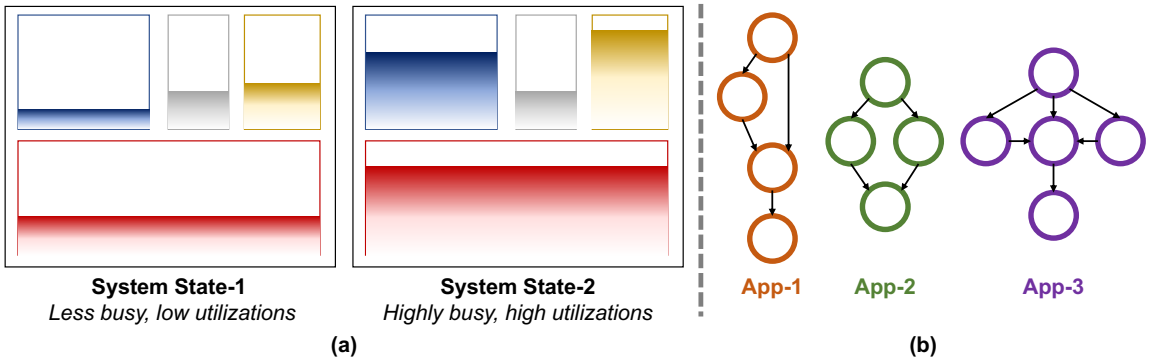


Figure 6.1: (a) An illustration of two different system states showing different levels of utilization of system resources (or PEs), and (b) an illustrative example of applications represented as directed flow graphs (DFGs). Nodes in a DFG represent tasks (key computational components) in an application. The edges denote the dependency between tasks and the weight of the edges represents the communication volume between tasks.

between tasks. The scheduling granularity in this work is at the task level, i.e., the nodes of the DFGs are assigned to processing elements.

6.3.2 Multi-Objective Reinforcement Learning (MORL)

Task scheduling, at its core, is an NP-hard sequential decision-making problem [120, 121]. It can be formulated as a Markov Decision Process (MDP) defined by the tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, r, \gamma \rangle$, where $\mathcal{S}$, $\mathcal{A}$, $\mathcal{P}(s'|s, a)$, r, and γ represent state space, action space, transition distribution, reward vector, and discount factor, respectively. Reinforcement Learning is a class of algorithms that aims to find an optimal policy for an agent to maximize its cumulative reward in an MDP. According to the state s of the environment and the current policy π, the agent chooses an action a. Based on this action, the environment returns the next state s' and reward r. The expected cumulative rewards starting from state s following a policy π can be represented as state value function, $V^{\pi}(s)$. The RL algorithm then iteratively updates the agent's policy (π) and value function (V^{π}) based on the feedback received from the environment in the form of rewards. This process continues until the agent reaches a terminal state or a maximum number of steps.

In a multi-objective setting, each objective is associated with a reward signal, which transforms the scalar reward into a vector $\mathbf{r} = [r_1, r_2, ..., r_M]^T$, where M is the number of objectives. This vectorized reward can be represented by a vectorized state value function $MV^{\pi}(s)$ [23], as illustrated in Fig. 6.2(a). In the RL domain, *scalarization* is the most commonly used approach to solve multi-objective optimization problems [34, 25, 4, 1]. This approach transforms the reward vector into a

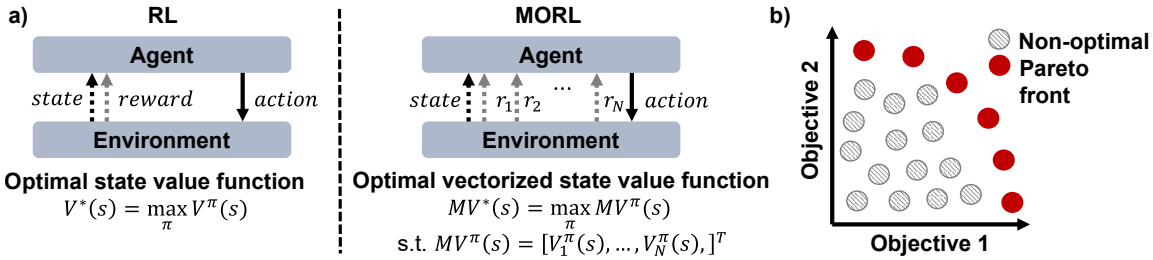


Figure 6.2: (a) A transformation of the RL setting to MORL. (b) An example of a Pareto front curve that trades off between two different optimization objectives.

single scalar, $f_\omega(r) = \omega^T \mathbf{r}$. The MDP is then transformed into a multi-objective Markov decision process (MOMDP), defined by the tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathbf{r}, \Omega, f_\omega \rangle$, where $\mathbf{r}$ and Ω represent the reward *vector* and *preference space*, respectively. Using a preference $\boldsymbol{\omega} \in \Omega$, the function $f_\omega(\mathbf{r}) = \boldsymbol{\omega}^T \mathbf{r}$ yields a scalarized reward. If we fix $\boldsymbol{\omega}$ as a vector, the MOMDP can be treated as a standard MDP and solved using conventional RL methods. Nonetheless, if we consider all possible returns and preferences in Ω, we can obtain a set of non-dominated policies referred to as the Pareto front. As depicted in Fig. 6.2(b), this set includes non-optimal solutions. A policy π is considered Pareto optimal if no other policy π' enhances the expected return for an objective without causing degradation in the expected return of any other objective.

In our framework, we extend the standard proximal policy optimization (PPO) algorithm to a multi-objective (MO-PPO) variant by considering a vectorized reward ($\mathbf{r}$) and state value function ($\mathbf{V}^\pi$). Both the policy and the state value function take preference vector $\boldsymbol{\omega}$ as input, efficiently learning the multi-dimensional objective space. The details of MO-PPO are provided in Section 6.5.

6.3.3 Differentiable Decision Tree (DDT)

A decision tree (DT) consists of root node η_r, decision nodes η_d, and leaf nodes η_l. Considering an input $x \in \mathbb{R}^F$ where F is the number of features, the root node η_r and each decision node η_d are represented with a boolean expression $p_\eta = x_{f_\eta} - \phi_\eta$. Here, x_{f_η} and ϕ_η denote the chosen feature and splitting threshold for node η, respectively. Based on these node expressions, the objective is to identify the optimal path until a leaf node is reached. Each leaf node η_l has a learned probability distribution Q_l. Based on this distribution, a corresponding label is returned as the output of the tree. The transparency and interpretability of decision trees are mainly due to the simplicity with which the feature f and the threshold ϕ_f can be extracted from every decision node [151]. Nevertheless, traditional DTs have a tendency to overfit and fail to generalize well [152].

Differentiable decision trees (DDTs) have been introduced to address the limitations of traditional DTs by combining the interpretability of traditional decision trees with the differentiability of neural networks [152, 153, 154]. DDTs leverage a continuous relaxation of the original decision tree structure, which enables gradient-based optimization methods to be employed during training. In DDTs, the boolean expression p_η is replaced by a sigmoid function [155, 154]:

$$p_\eta = \sigma(\alpha_\eta(w_\eta^T x - \phi_\eta)) = \frac{1}{1 + e^{-(\alpha_\eta(w_\eta^T x - \phi_\eta))}} \tag{6.1}$$

where w_η and ϕ_η are the weights and bias of the node η. α_η is the steepness

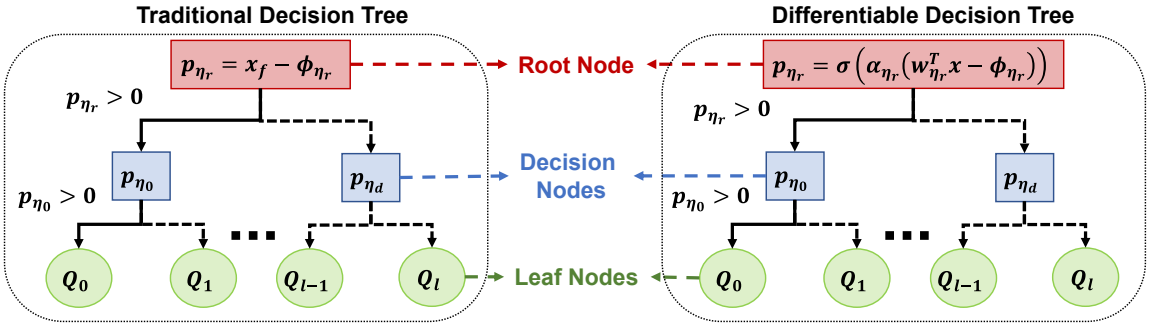


Figure 6.3: An overview of a traditional decision tree and a differentiable decision tree.

parameter of the sigmoid function and is also a learnable parameter. This expression linearly combines input x with node weights and compares it to a bias term to traverse the tree, as shown in Fig. 6.3. This means that all input features are used at each node to make a decision. With this modification, the tree can now be trained using gradient descent for parameters w, ϕ, and α.

6.4 RL Environment for DSSoCs

OpenAI Gym [156] is a widely popular platform for developing and validating RL algorithms since it provides standardized environments [157]. It allows users to plug-and-play different components, such as the RL algorithm and simulated environments. Gym also provides standard API to interact with environments and serves as a benchmark to compare RL algorithms. Therefore, we enhance the capabilities of an open-source DSSoC simulator [158] to integrate it into a Gym environment. We plan to release our integrated infrastructure to the public to stimulate future research.

6.4.1 End-to-End Training Flow with Novel RL Environment for DSSoCs:

Training RL algorithms for task scheduling in DSSoCs involves integrating three distinct components: 1) DSSoC simulator, 2) RL agent and 3) Gym environment. The first step in the process involves instantiating the DSSoC simulator within a standard Gym environment template, shown in Fig. 6.4. This template includes the essential functionalities of initialization, reset, and step. The states, actions, and variables are initialized in the corresponding functions. The most critical function within the Gym environment is the step function, which enables the environment to transition from one state to another by taking action and generating a reward. It is crucial to carefully design the control flow between the functionalities of the

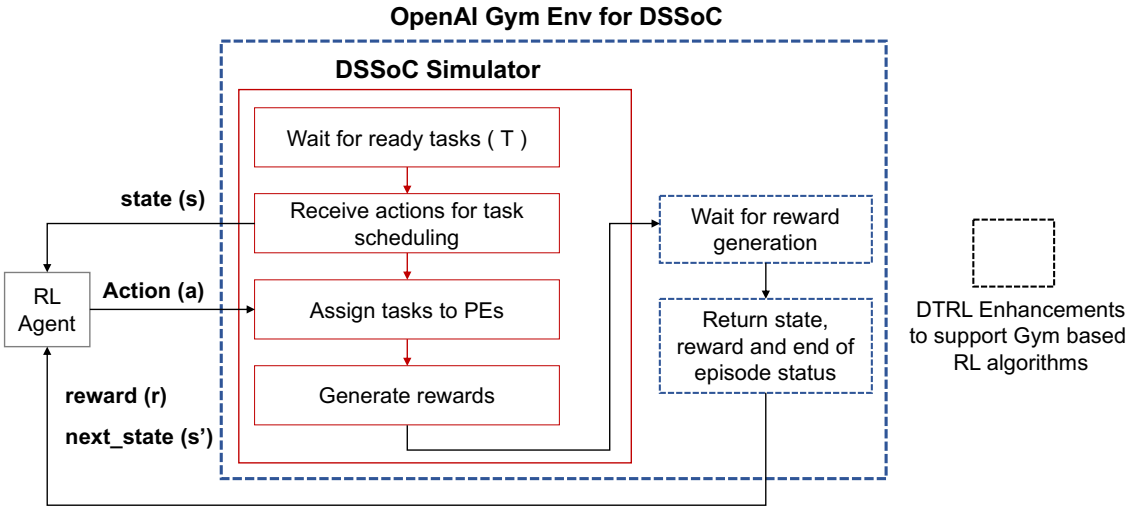


Figure 6.4: The DTRL framework enrolls a DSSoC simulator as an OpenAI Gym environment to enable compatibility with the community standard practice of evaluating RL algorithms.

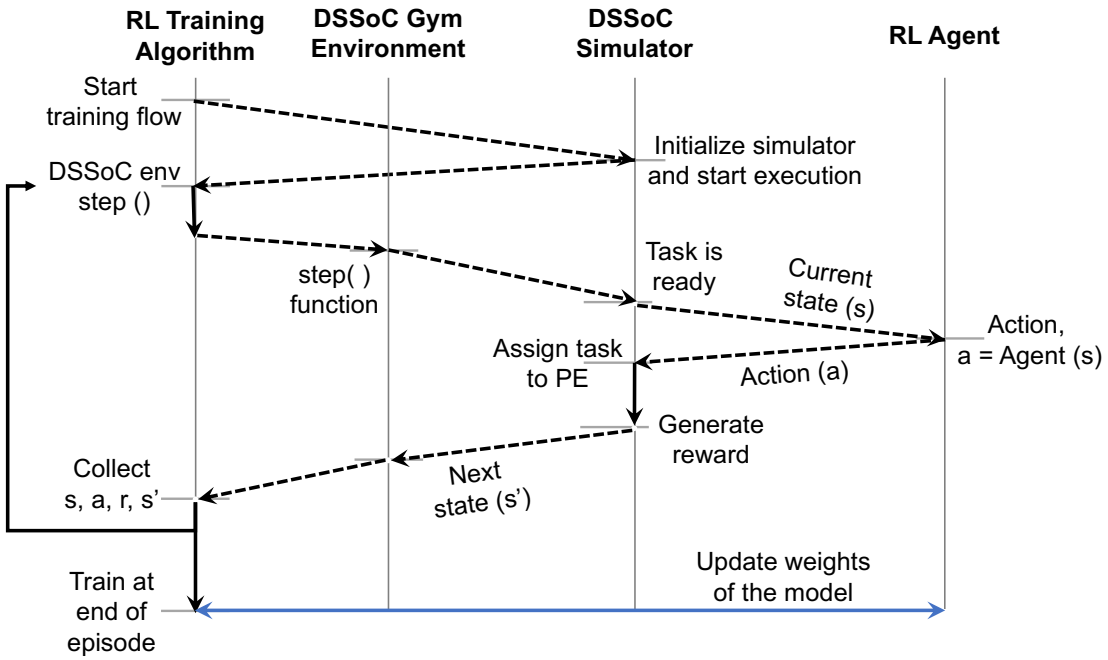


Figure 6.5: The end-to-end RL training flow with the DSSoC simulator as a Gym environment. The dashed lines denote the event-driven handshake between the different components in the training process.

RL environment and DSSoC simulator to ensure that the RL agent can effectively correlate the state, action, and reward.

To this end, we design handshake events to facilitate communication between the simulator and the RL environment. Fig. 6.5 illustrates the complete end-to-end RL training flow with the DSSoC simulator acting as a Gym environment. The RL training algorithm triggers the training process and initializes the DSSoC simulator. The step function then initiated, running the simulator until a task becomes available for scheduling. The current state (s) is characterized by input features describing the task, whereas the simulator populates the system state. Based on the current state, the RL agent generates an action and relays it to the

simulator. The simulator then assigns a processing element (PE) to the task and generates a reward, which is transmitted back to the RL agent. The algorithm tracks the current state (s), action (a), next state (s'), and reward (r) for several tasks until the end of the episode and then updates the parameters of the model. The workload is considered complete when the environment executes all of the tasks.

6.4.2 Environment Dynamics

We ensure that the dynamics of the environment allow DTRL to adapt to any DSSoC configuration and streaming applications. The DSSoC comprises several PEs, and similar PEs are grouped into C processing clusters. A user-selected number of frames are generated during each episode based on the domain applications, with each frame containing several tasks. An episode terminates when all the tasks in the workload complete execution. Scheduling is performed for each task in the workload, and hence the step function transitions between ready tasks, as described in Section 6.4.1. State, action, and rewards are generated for each task, meaning that each time step t in the environment corresponds to a ready task.

State Space: consists of features that describe the task, application, and state of the DSSoC at a given time instant ($\mathcal{S} \subseteq \mathbb{R}^{(2C+4)}$). The task-related features include the position of the task in DFG, the application type and ID, and the execution times

Table 6.2: The list of features that constitute the state space.

Feature Information	Dimension	Feature Information	Dimension
Task ID	$\mathbb{R}$	Execution time on C clusters	$\mathbb{R}^C$
Depth of task in DFG	$\mathbb{R}$	Application ID	$\mathbb{N}$
Application type	$\mathbb{N}$	Earliest availability of C clusters	$\mathbb{R}^C$

among the different processing clusters. The DSSoC state is described is described by the earliest availability times of the PEs within a cluster, indicating their busy or free status. A complete list of the features that comprise the state space is presented in Table 6.2.

Action Space: consists of selecting from a set of C processing clusters, ($\mathcal{A} \subseteq \mathbb{N}^{C}$). In this study, we utilize a configuration comprising of five processing clusters. At each step, the action space is used to select the processing cluster for task execution. Once a cluster is selected, the processing element (PE) with the earliest availability is chosen to execute the task within the cluster.

Reward Vector: comprises execution time and power consumption since DTRL aims to learn the trade-off between these objectives. Both components must be minimized to maximize energy efficiency. Therefore, the negative values of the expected execution time and power consumption of the task on the chosen PE construct the reward vector. The DSSoC simulator generates the values of these quantities using the profiling information in its database.

6.5 DTRL: Decision Tree based Multi-Objective Reinforcement Learning Algorithm

The number and complexity of tasks processed in each episode can vary significantly. These variations can lead to high gradient variance and unstable learning progress. To address this challenge, we utilize a multi-objective variant of the Proximal Policy Optimization (MO-PPO) algorithm, which can learn multiple objectives and ensure

stability of policy updates across all optimization steps. Finally, DTRL employs a differentiable decision tree, rather than a neural network, as the actor to reduce inference latency overheads and enhance the interpretability. This section present the details of the proposed DTRL framework.

6.5.1 Invalid Action Masking

DSSoCs typically consist of general-purpose cores and fixed-function accelerators (e.g. fast Fourier transform (FFT), forward error correction (FEC), finite impulse response (FIR)). These accelerators do not support all tasks streaming into the DSSoC. Consequently, some tasks involve invalid actions during training. DTRL should be able to manage invalid actions for efficient and stable training. The most common approach to penalize invalid actions is giving a high negative reward [159] such that the agent learns to maximize the reward by not taking any invalid action. However, this approach suffers from low explorative capabilities and spends a vast amount of time learning invalid actions at each state, especially when the action space dimension is large. Therefore, in our work, we use invalid action masking [159] to constrain the DTRL agent to only choose clusters of PEs that support the given task.

In our algorithm, the policy (π_θ) generates logits ($l_i,\ i = 1, \ldots, |\mathcal{A}|$), which are subsequently converted to action probabilities ($\pi_\theta(a_i|s)$) via a softmax operation. During training, an action is selected by sampling from a distribution of these probabilities, denoted as $\pi_\theta(\cdot|s)$. The policy is updated using gradient descent, similar to other policy gradient approaches. Invalid action masking is applied

by setting the logits of invalid actions to a large negative number, typically -1×10^8. This ensures that the probability of these masked actions is zero, without compromising the gradient update. In fact, this technique enhances the gradient update, as the gradient corresponding to the logits of masked actions becomes zero.

6.5.2 Multi-Objective PPO with DDT as an Actor: Algorithm Details

This work aims to obtain a single policy that covers the entire preference space for multiple objectives in a runtime task scheduling problem. To achieve this, we adopt an approach similar to the existing literature on Multi-Objective Reinforcement Learning (MORL) [4, 40] to extend the PPO to a multi-objective version (MO-PPO). For this extension, we first consider that the environment returns a vector of reward as exemplified in Section 6.3.2. The value network is vectorized to efficiently learn to model multiple objectives for a given preference vector $\boldsymbol{\omega}$. Specifically, the value network takes state s and preference vector $\boldsymbol{\omega}$ as inputs and outputs $|\mathcal{A}| \times M$ state values, as explained in Algorithm 9, where M is the number of objectives. Therefore, the state value function becomes $\mathbf{V}_{\Phi}(s, \boldsymbol{\omega})$, which returns a vector of expected returns for a given state s and preference $\boldsymbol{\omega}$ by following a current policy π_{θ}. During training, the vectorized value network is updated by minimizing the mean-squared error between estimated and target values using gradient descent as

the optimization algorithm:

$$L_\phi = \frac{1}{T}\sum_{t=0}^{T}\Big(\boldsymbol{V_\phi}(s_t, \boldsymbol{\omega}) - (r_t + \gamma \boldsymbol{V_\phi}(s_{t+1}, \boldsymbol{\omega}))\Big)^2 \tag{6.2}$$

The vectorization of the reward and state value function results in a vectorized advantage function, as presented in equation 6.3:

$$\boldsymbol{A}(s_t, a_t, \boldsymbol{\omega}) = \mathbf{r_t} + \gamma \boldsymbol{V_\phi}(s_{t+1}, \boldsymbol{\omega}) - \boldsymbol{V_\phi}(s_t, \boldsymbol{\omega}) \tag{6.3}$$

To compute the modified advantage function, $\boldsymbol{\omega}^T\mathbf{A}(s_t, a_t, \boldsymbol{\omega})$, a weighted-sum scalarization is applied to the advantage function, similar to the state value function. Furthermore, in our implementation, the policy takes the preference vector, $\boldsymbol{\omega}$, as an additional input along with the state s, to make a decision. The policy loss for the multi-objective PPO (MO-PPO) is then given by:

$$L_\theta = \frac{1}{T}\sum_{t=0}^{T} min\Big(\rho(\theta)\boldsymbol{\omega}^T\mathbf{A}(s_t, a_t, \boldsymbol{\omega}),\ clip(\rho(\theta), 1-\epsilon, 1+\epsilon)\boldsymbol{\omega}^T\mathbf{A}(s_t, a_t, \boldsymbol{\omega})\Big) \tag{6.4}$$

$$\rho(\theta) = \frac{\pi_\theta(a_t|s_t, \boldsymbol{\omega})}{\pi_{\theta old}(a_t|s_t, \boldsymbol{\omega})} \tag{6.5}$$

To ensure efficient runtime task scheduling, having a neural network with high inference overhead is not desirable. Instead, we use a differentiable decision tree (DDT) as the policy with sigmoid as the activation function at each node. The

MO-PPO algorithm can be used for the DDT policy without requiring modifications. For the value network, fully connected layers with hyperbolic tangent activation functions are employed.

Algorithm 9 outlines the training process of the DTRL framework. At the beginning of each episode during training, we randomly sample a preference vector ($\boldsymbol{\omega} \in \Omega : \sum_{i=0}^{L} \omega_i = 1$) from a uniform distribution. To determine the workload intensity of the task scheduling problem, the simulation framework takes the target throughput (e.g., frames per milliseconds) as input. Thus, at the start of each episode, we randomly sample a target throughput y.

A vectorized architecture with a single policy to gather transitions from multiple environments is a common technique in [131, 160]. *To increase the sample efficiency of our algorithm*, we adopt a similar strategy. We initialize P child processes with different seeds. The DDT policy and the value network are shared among child processes and the main process. We divide the preference space into P sub-spaces ($\tilde{\Omega}$) and assign a subspace to each child process. Each child process is responsible for its own preference sub-space, and in each child process, a preference vector is randomly sampled from its assigned sub-space. Using the policy π_θ, we collect T amount of samples. Using these samples, advantages $\mathbf{A_t}$, target values $\mathbf{r_t} + \mathbf{V_\phi}(s_{t+1}, \boldsymbol{\omega})$, and the probabilities $\pi_{\theta_{old}}(a_t|s_t, \boldsymbol{\omega})$ are obtained. The original PPO implementation uses generalized advantage estimation (GAE) to calculate advantages [131, 160]. We also employ this technique with GAE parameter of 0.95. These child processes run in parallel to collect transitions and do necessary computations using the same DDT policy and value network. The obtained transitions are then transmitted to

the main process, where they are stored in a trajectory buffer of size $P \times T$.

The algorithm then updates both the value network and the DDT policy parameters (ϕ, θ) according to the loss functions described in equation 6.2 and 6.4. The total number of optimization steps required to update the parameters is determined by the number of epochs K and the minibatch size b. We use an Adam optimizer with a learning rate of 3E-4 for both the DDT policy and the value network. The hyperparameters for DTRL are presented in Table 6.3.

Algorithm 9: DTRL Framework

Input: Total number of time steps N, Number of steps to run per policy rollout T, Discount factor γ, Number of epochs to update the policy and value network K, Minibatch size b, Number of child processes P, clipping value ϵ.
Initialize: DDT policy π_θ and value network $\mathbf{V_\phi}$ with parameters θ and ϕ, Policy π_θ.
while *Total Number of Steps* $<$ N **do**
    `// Child Process`
    Reset the environment to state s_0 and randomly initialize target throughput y.
    Sample a preference vector $\boldsymbol{\omega}$ from the subspace $\tilde{\Omega}$.
    for $t = 0 : T$ **do**
        Choose a_t according the current policy π_θ and invalid action mask a_t^m.
        Collect samples $\{s_t, a_t, r_t, s'_t\, \boldsymbol{\omega}, done\}$ by interacting with the environment using action a_t.
    Obtain $\mathbf{A_t}, \mathbf{r_t} + \mathbf{V_\phi}(s_{t+1}, \boldsymbol{\omega})$, and $\pi_{\theta_{old}}(a_t|s_t, \boldsymbol{\omega})$ using DDT and the value network.
    Transfer populated $\left(s, a, \mathbf{r}, s', \boldsymbol{\omega}, a^m, \mathbf{A}, \mathbf{r} + \mathbf{V_\phi}(s, \boldsymbol{\omega}), \pi_{\theta_{old}}(a|s, \boldsymbol{\omega})\right)$ to main process.
    `// Main Process`
    Store the incoming transitions from child processes in a trajectory buffer with size $P \times T$.
    for $k = 1$: K **do**
        for $i = 0$: $(P \times T/b)$ **do**
            $idx_{start} = d \times (b - 1)$
            $idx_{end} = d \times (b)$
            Sample a minibatch from the trajectory buffer according to start and end indices.
            Obtain value estimates and new π_θ.
            Calculate L_θ and L_ϕ
            Update θ and ϕ by applying SGD to L_θ and L_ϕ.

Table 6.3: Definition and hyperparameter values used in this paper.

Hyperparameter	Description	Value
P	Number of parallel processes	10
N_{Layer}	Number of hidden layers in the value network	1
N_{Neuron}	Number of hidden neurons in the value network	64
$depth$	Depth of DDT policy	3
N	Total number of time steps for the entire training	3×10^7
T	Number of steps to run per policy rollout	1024
γ	Discount factor	0.99
λ	GAE Parameter	0.95
ϵ	Clipping factor	0.1
K	Number of epochs to update the policy and value network	20
b	Minibatch size	64
lr	Learning Rate	3×10^{-4}

6.6 Experimental Evaluation

This section evaluates the proposed DTRL framework for runtime task scheduling in DSSoCs. Section 6.6.1 first presents the domain applications, DSSoC configuration, and the simulation and emulation frameworks used for evaluation in this work. Then, it introduces the baseline task schedulers that are used for comparison. Finally, Section 6.6.2 presents detailed experimental evaluations showing performance and energy consumption improvements of the proposed DTRL framework over the baseline scheduling approaches. This section emphasizes the generalizability of DTRL learning the trade-off between two conflicting objectives, average execution time and power consumption, as a Pareto front set of solutions.

6.6.1 Experimental Setup

Domain Applications: The evaluation of DTRL involves six applications in the domain of wireless communications and radar systems: WiFi transmitter, WiFi

receiver, range detection, single-carrier transmitter, single-carrier receiver, and temporal mitigation. Workloads comprising 100 frames each are constructed using a mix of the six applications.

DSSoC Configuration: The configuration of DSSoC consists of sixteen PEs classified into five clusters based on their functionalities. These clusters comprise four LITTLE Arm A57 cores, four big Arm Cortex-A53 cores, and fixed-function accelerators, which include two matrix multiplication (MM) cores and four fast Fourier transform (FFT) cores, and two Viterbi decoding cores. The PEs are chosen to fulfill the computational demands of the targeted domain applications. The domain applications and DSSoC configuration employed in this study represent the most comprehensive configuration currently available within the DSSoC simulator[132].

Simulation and Emulation Frameworks: We first evaluate DTRL using our novel OpenAI Gym environment integrated with an open-source DSSoC simulator, DS3 [158], as described in Section 6.4. This simulator is validated against two commercial SoCs, Odroid-XU3 and Xilinx Zynq ZCU102.

We measure the hardware runtime overhead of the global multi-objective DDT scheduling policy by implementing it within CEDR [116] (an open-source Linux-based emulation and runtime environment) on the Xilinx Zynq ZCU102 platform.

Baseline and State-of-the-Art Approaches for Comparison: To begin our comparative analysis, we employ an optimization-based approach that employs integer linear programming (ILP) through IBM ILOG CPLEX Optimization Studio [58]. However, this approach suffers from severe time and complexity issues, especially for high-intensity workloads (high target throughput) due to a large number of

variables and constraints. The solver takes several hours to days to derive the solution and even fails to achieve an optimal decision in several cases. Therefore, we enforce a timeout value of two minutes for each solver invocation. The ILP scheduler is reported as a reference point, and it is not feasible at runtime due to its prohibitive runtime overhead (in the order of minutes).

We also choose heuristic schedulers for comparisons with DTRL. The earliest task first (ETF) [5] scheduler efficient scheduling decisions by iterating through all the ready tasks and available PEs to determine the task with the earliest finish time, thereby making it a suitable choice for comparison. We modify ETF to target different objectives such as power consumption, energy consumption, and energy-delay product. The different ETF variants contribute to distinct comparison points in our evaluation. We also compare DTRL with a machine-learning-based scheduler that uses imitation-learning for task scheduling (ILS) [6]. This framework uses ETF as the Oracle and trains a regression tree to approximate the Oracle decisions. State-of-the-art MORL approaches [1, 4] are introduced to address specific applications, such as continuous robotics tasks and grid world games. However, these approaches do have certain drawbacks. For instance, the Envelope algorithm [1] requires the action space to be discrete and suffers from sample inefficiency, while PG-MORL [4] necessitates a continuous action space and both objectives to be positive. Nevertheless, when it comes to the task scheduling problem, which involves mixed-sign objectives and invalid discrete actions, these approaches are unable to effectively handle such scenarios. Furthermore, these MORL approaches typically involve neural networks with dense layers, resulting in a significantly higher run-

time overhead compared to DTRL and ETF (as provided in Section 6.6.2.3) when implemented on real hardware platforms. The runtime overhead of these neural network-based approaches can be two to three orders of magnitude greater than that of DTRL and ETF [6]. Therefore, to provide an additional basis for comparison, we introduce a modification of an existing MORL method [7]. This modified method, referred to as Scalarized-MOPPO, involves performing updates after computing the inner product of the vectorized value function and the preference vectors. Notably, in Scalarized-MOPPO, the policy DDT and the value network no longer take preferences as inputs, as the method trains separate policies for various preference ratios of the different objectives.

6.6.2 DTRL Evaluation

A global DDT scheduling policy is obtained by employing the training procedure described in Section 6.5. The DDT is constructed with 16 input features (including objective preferences) and uses a maximum depth of 3. Each workload comprises 100 frames, and the frames are dynamically injected into the system based on an exponential distribution. The average metrics from simulations with 10 random seeds are used to avoid bias in the distribution. The target throughput is varied between 1–50 frames per millisecond. The vector indicating the preference for the execution time and power consumption objectives is denoted by $\{a, b\}$ respectively.

6.6.2.1 Performance and Energy Consumption Evaluation

Fig. 6.6(a) compares the average frame execution time of DTRL with baseline and state-of-the-art schedulers. At high target throughputs, frames are injected faster than they are processed; hence, newly injected frames overlap with existing frames. The overlap results in significant competition for the shared computing resources, thereby resulting in a higher frame execution time, as observed in Fig. 6.6. The DTRL policy in Fig. 6.6(a) uses a preference vector of $\{1, 0\}$, whereas the results for Scalarized-MOPPO are obtained using the policy that is separately trained for the same preference vector. DTRL achieves an execution time speedup of 1.03×, 1.05×, 1.25×, 1.9×, and 9× than ILS, Scalarized-MOPPO, ETF-EDP, ETF-Energy, and ETF-Power, respectively. We note that ETF-Power assigns all tasks to LITTLE cores (since it has the lowest power consumption), causing the system to become heavily congested and drastically increasing average frame execution time. Although DTRL is trained with multiple objectives, it still achieves an average execution time within 4% and 7% of ETF and ILP, respectively. Additionally, the runtime overhead of ETF is 2× and 10× higher than that of DTRL, as evaluated in Section 6.6.2.3. It is important to highlight that ETF, ILP, and ILS are designed to optimize a single specific objective, whereas Scalarized-MOPPO and DTRL are specifically trained to handle multiple objectives. On the one hand, Scalarized-MOPPO is trained individually for each preference vector. Training several policies imposes a severe stress on the platform in terms of training time and compute resources. Furthermore, the Scalarized-MOPPO approach requires the platform to store all of them and appropriately choose one such policy based on the preference vector at runtime.

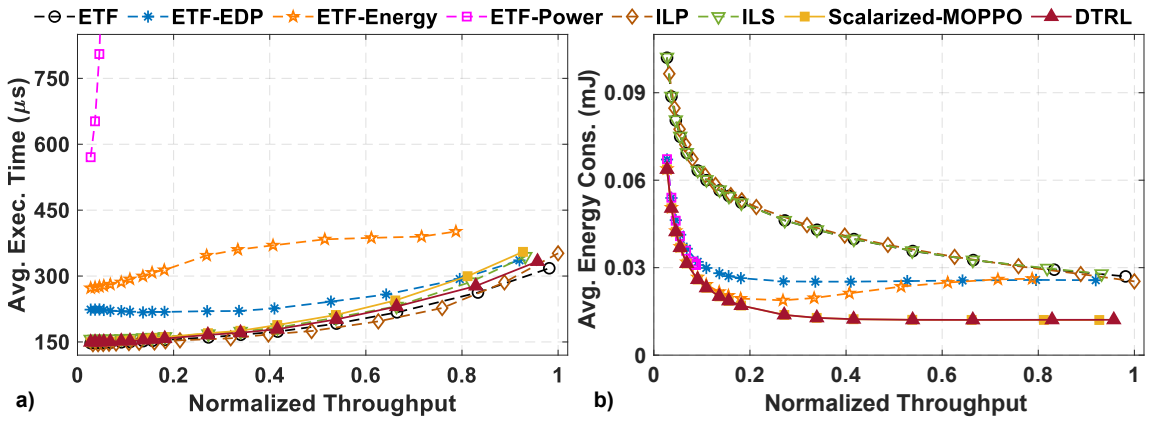


Figure 6.6: Comparison of a) average frame execution time and b) average energy consumption between ETF [5], ETF-EDP, ETF-Energy, ETF-Power, ILP solution, ILS [6], Scalarized-MOPPO ([7]-Adapted) and DTRL to schedule a workload comprising six streaming applications. The x-axis in Fig. 6.6(a) and 6.6(b) is normalized to the throughput achieved by the ILP solution.

On the other hand, DTRL learns a Pareto front set of solutions for execution time and power consumption objectives using a single policy.

DTRL's energy consumption is evaluated by using a preference vector of $\{0, 1\}$ to the global DDT policy, as shown in Fig. 6.6(b). DTRL achieves 3.06×, 3.08×, 3.06×, 1.97×, 1.75×, and 1.05× lower energy consumption compared to ETF, ILP, ILS, ETF-EDP, ETF-Energy, and ETF-Power, respectively. It achieves very similar energy consumption values compared to Scalarized-MOPPO. It is worth noting that *DTRL does not require retraining for the energy objective* since the global DDT policy dynamically generates near-optimal decisions based on the application- or user-defined preference provided at runtime.

6.6.2.2 Multi-objective Functionality of DTRL

This section evaluated the multi-objective aspect of the proposed DTRL framework. To this end, we show the average frame execution time versus energy efficiency curves obtained by DTRL at varying throughputs, using multiple preference vectors. Fig. 6.7(a)-(c) illustrates the curves for low, medium, and high throughput workloads. The evaluation employs preference vectors (ω) separated by a step size of 0.1, $\omega \in \{\{1,0\},\{0.9,0.1\},\ldots,\{0.1,0.9\},\{0,1\}\}$. Fig. 6.7(a)-(c) also shows the energy efficiency (in milli-Joules per frame) of the baseline and state-of-the-art schedulers. We note that ETF, ILP, and ILS will have only one point on the plot since they support only one objective. However, for ETF, its variants correspond to different comparison points in the objective space. For instance, ETF-Power corresponds to the comparison point where the preferences for execution time and power consumption are 0 and 1, respectively. By using a single global DDT policy, DTRL covers the entire preference space and produces solutions comparable to ETF and ILP, as described in Section 6.6.2.1. Furthermore, it outperforms the other schedulers, providing flexibility to generate near-optimal scheduling decisions for any preference vector specified at runtime. We also observe that DTRL scales to several throughputs, achieving comparable or improved metrics compared to the other approaches. Although Scalarized-MOPPO can achieve a similar Pareto front set of solutions to DTRL by training a separate policy for each preference vector, it faces challenges when deployed on a hardware platform due to the need to store all possible scheduling policies. As a result, the scalability of Scalarized-MOPPO is constrained when dealing with numerous objectives and their corresponding

ratios. Moreover, Scalarized-MOPPO struggles when the objectives exhibit substantial differences in magnitudes, as it learns from a scalarized reward function that requires domain expertise for its design.

The DTRL policy, trained with two optimization objectives, namely average frame execution time and power consumption, achieves the Pareto front curves as shown in Fig. 6.8. A workload with 100% target throughput denotes the maximum frame rate supported by the platform. As the target throughput increases, the average job execution time and power consumption increase due to the increase in congestion in the SoC. The power consumption and execution time tradeoff curves at multiple target throughputs strongly demonstrate that DTRL scales to all workload complexities.

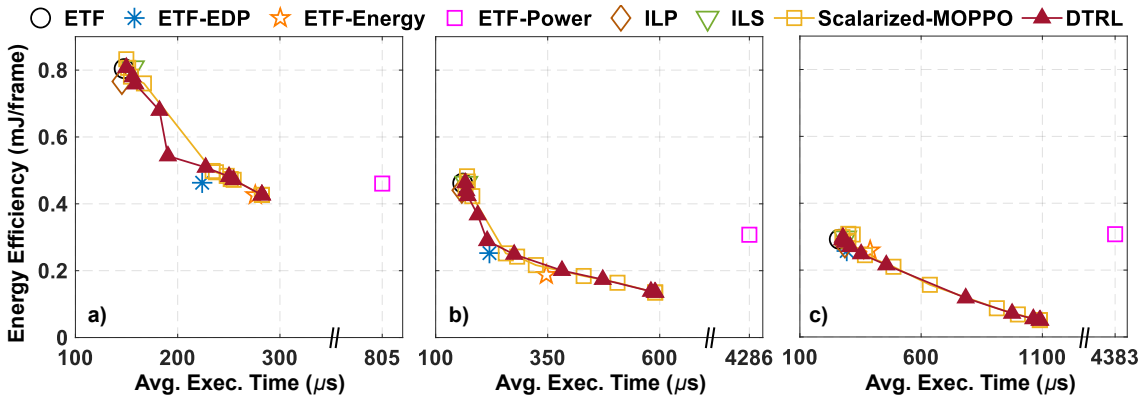


Figure 6.7: Average frame execution time (µs) vs. Energy efficiency (mJ / frame) for a) low b) medium c) high target throughputs. Comparison between ETF [5], ETF-EDP, ETF-Energy, ETF-Power, ILP solution, ILS [6], Scalarized-MOPPO ([7]-Adapted), and DTRL are presented. DTRL achieves multiple near-optimal policies using various preferences.

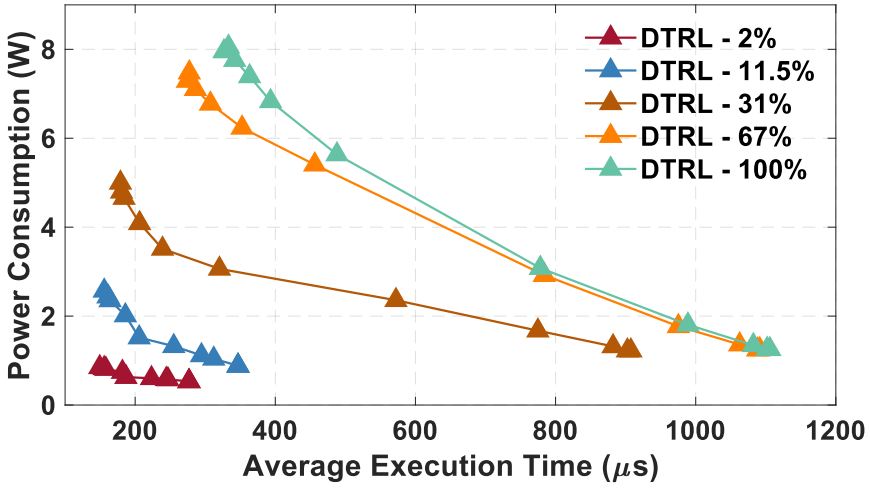


Figure 6.8: Pareto front solutions achieved by the proposed DTRL framework that trades off between the average execution time of applications with the power consumption at various target throughputs (in %).

6.6.2.3 Performance Evaluation of DTRL on a Runtime Emulation Platform

We implement DTRL in CEDR [116] and analyze its performance on a Xilinx Zynq ZCU102 FPGA [161]. This evaluation uses a configuration consisting of one FFT core, one MM core, three general-purpose cores, and three domain applications, namely the WiFi transmitter, range detection, and temporal mitigation. The trained DDT model for the above configuration uses 12 input features and a maximum depth of 3. It is deployed in the CEDR framework as a C++ module.

Fig. 6.9 presents the comparison of average frame execution time between DTRL (with a preference vector of {1,0}) and ETF executing workloads in CEDR on the FPGA at six different target throughputs. DTRL achieves lower execution time than ETF at all workload throughputs. As discussed in Section 6.6.1, ETF incurs high computational complexity due to the quadratic dependency on the number of

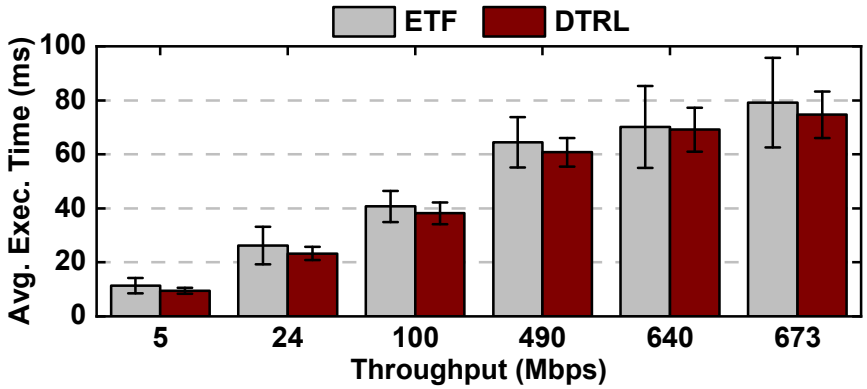


Figure 6.9: Comparison of average job execution time between ETF and DTRL on a real hardware platform (Xilinx Zynq ZCU102 FPGA) to schedule a workload comprising three streaming applications. Each bar in the plot represents the average of 50 trials, with each trial consisting of 1390 tasks. Error bars represent the standard deviation of the trials.

ready tasks, and its runtime overhead varies between several hundred–thousands of nanoseconds. On the contrary, the DDT DTRL policy achieves a runtime overhead of 120 ns per scheduling decision. Therefore, we demonstrated the ability of the proposed DTRL framework to outperform state-of-the-art approaches in both a DSSoC simulation framework and a real hardware platform (Xilinx Zynq ZCU102 FPGA).

6.6.2.4 Scalability and Limitations

DTRL uses a differentiable decision tree (DDT) at its core to make scheduling decisions. As explained in Section 6.3.3, at each node in the tree, features are linearly combined with node weights and compared to a bias term. This enables DDTs to handle high-dimensional input features and complex interdependencies between the input features, which can be challenging for traditional decision trees. Addi-

tionally, DDTs can provide interpretable models that allow users to understand the reasoning behind scheduling decisions. However, a potential limitation of using DDTs is their increased complexity and the size of the feature space, which may limit scalability in certain scenarios. As the number of tasks and processors increases, the feature space also grows, making it challenging to find an optimal solution at runtime. If the state space is constructed with features for individual PEs, the time and space complexity of DTRL can reach $O(2N)$, where N represents the total number of PEs. To mitigate this challenge, we address it by grouping PEs into processing clusters (C), thereby reducing the number of features that would otherwise increase with a larger system-on-chip (SoC) configuration. Consequently, the time and space complexity of DTRL is reduced to $O(2C)$, with C being significantly smaller than N. Additionally, it is worth noting that the complexity of selecting a specific PE within a cluster is $O(k)$, where k is the number of PEs within that cluster. Furthermore, as with traditional decision trees, overfitting can occur if the model becomes too complex and with the increase in the tree depth. In our work, we limit the DDT depth to 3 and transitions from randomly generated scenarios at each episode to avoid overfitting.

7 CONCLUSIONS AND FUTURE DIRECTIONS

7.1 Conclusion

MORL faces several challenges: balancing conflicting objectives requires sophisticated techniques, existing approaches often need repetitive and costly training, and real-world applications demand optimal, feasible, and energy-efficient solutions in resource-constrained environments. This dissertation aims to tackle these challenges and advance reinforcement learning to enable its application to real-world problems by developing multi-objective, energy-aware, and hardware-friendly algorithms that can be effectively applied in edge AI applications.

The key contributions of this work are as follows:

- **PD-MORL: Preference-Driven Multi-Objective Reinforcement Learning Algorithm** [25]: It proposes a novel MORL algorithm that trains a *single universal network to cover the entire preference space scalable to continuous robotic tasks*. The proposed approach, Preference-Driven MORL (PD-MORL), utilizes the preferences as guidance to update the network parameters. It also employs a novel parallelization approach to increase sample efficiency. We show that PD-MORL achieves up to 25% larger hypervolume for challenging continuous control tasks and uses an order of magnitude fewer trainable parameters compared to prior approaches.

- **A Comprehensive Multi-Objective Energy Management Approach for Wearable Devices with Dynamic Energy Demands**: It learns the trade-off between

meeting the application's energy demand and maintaining the battery energy level. We deployed our framework on a wearable device prototype using TensorFlow Lite for Micro, leveraging its small (less than 120 KB) memory footprint. Evaluations show that tinyMAN-MO operates within 10% of the Pareto-optimal solutions with only 1.98 ms execution time and 23.17 μJ energy consumption overhead.

- **GEM-RL: Generalized Energy Management of Wearable Devices using Reinforcement Learning** [26]: GEM-RL learns the trade-off between utilization and the battery energy level of the target device under dynamic EH patterns and battery conditions. It also uses a lightweight approximate dynamic programming (ADP) technique that utilizes the trained MORL agent to optimize the utilization of the device over a longer period. Thorough experiments show that, on average, GEM-RL achieves Pareto front solutions within 5.4% of the offline Oracle for a given day. For a 7-day horizon, it achieves utility up to 4% within the offline Oracle and up to 50% higher utility compared to baseline EM approaches. The hardware implementation on a wearable device shows negligible execution time (1.98 ms) and energy consumption (23.17 μJ) overhead.

- **A Self-Sustained CPS Design for Reliable Wildfire Monitoring** [27]: It presents the first self-sustained cyber-physical system that dynamically co-optimizes the wildfire detection accuracy and active time of sensors. The proposed approach employs reinforcement learning to train a policy that controls the sensor operations as a function of the environment (i.e., current

sensor readings), harvested energy, and battery level. The proposed cyber-physical system is evaluated extensively using real-life temperature, wind, and solar energy harvesting datasets and an open-source wildfire simulator. In long-term (5 years) evaluations, the proposed framework achieves 89% uptime, which is 46% higher than a carefully tuned heuristic approach. At the same time, it averages a 2-minute initial response time, which is at least 2.5× faster than the same heuristic approach. Furthermore, the policy network consumes 0.6 mJ per day on the TI CC2652R microcontroller using TensorFlow Lite for Micro, which is negligible compared to the daily sensor suite energy consumption.

- **DTRL: Decision Tree-based Multi-Objective Reinforcement Learning for Runtime Task Scheduling in Domain-Specific System-on Chips** [28]: DTRL trains a single global differentiable decision tree (DDT) policy that covers the entire objective space quantified by a preference vector. Extensive experimental evaluations demonstrate that DTRL captures the trade-off between execution time and power consumption, thereby generating a Pareto set of solutions using a single policy. Furthermore, comparison with state-of-the-art heuristic–, optimization–, and machine learning-based schedulers shows that DTRL achieves up to 9× higher performance and up to 3.08× reduction in energy consumption. The trained DDT policy achieves 120 ns inference latency on Xilinx Zynq ZCU102 FPGA at 1.2 GHz, resulting in negligible runtime overheads. Evaluation on the same hardware shows that DTRL achieves up to 16% higher performance than a state-of-the-art heuristic scheduler.

These contributions collectively advance the field of MORL by addressing key challenges such as balancing conflicting objectives, scalability, efficiency, and real-world applicability. The research outcomes bridge the gap between theoretical advancements and practical implementations, paving the way for more intelligent and adaptive systems across various domains.

7.2 Future Work and Research Directions

One of the critical aspects of RL algorithms is the required time to train an RL agent. The MORL algorithms proposed in this dissertation can take several hours of training due to the need to cover the entire preference space to obtain a diverse and representative Pareto front set of solutions. Future research should focus on optimizing the implementation of these RL algorithms to reduce training time. Potential approaches include developing more efficient training techniques, leveraging parallel processing, and incorporating transfer learning to reuse knowledge from previously trained models.

In addition to improving training efficiency, specific enhancements can be made to the wildfire monitoring framework. Integrating cameras as the final level of sensors in the hierarchy is crucial for accurately identifying wildfires, as camera images provide detailed visual information that can significantly enhance the system's detection capabilities. Building on this, developing a multi-agent, neighbor-aware, self-sustained sensor-suite network will improve the system's robustness and reliability. Such a network will enable sensors to collaborate, share information, and

optimize their operations collectively, leading to more efficient energy use and more accurate wildfire monitoring.

Further advancements can also be made to the Decision Tree-based Multi-Objective Reinforcement Learning (DTRL) algorithm. Including additional optimization objectives will broaden the algorithm's applicability and improve its performance in more complex scenarios. Additionally, reducing the computational overheads during runtime will make the algorithm more efficient and suitable for real-time applications. Exploring the applicability of DTRL in other domains, such as automotive systems, smart grids, and industrial automation, will validate its versatility and robustness across various multi-objective optimization problems.

A crucial aspect that was beyond the scope of this dissertation is the online learning feature of RL algorithms. For energy management, extending the prototype wearable device to log harvested energy over the course of a day will provide valuable data for real-time adaptation. Implementing online learning functionality within tinyMAN will enhance its adaptability and energy efficiency, enabling the system to adjust dynamically to changing energy conditions and improving its overall performance and sustainability.

By pursuing these future work and research directions, we can build on the foundations laid by this dissertation, further advancing the field of MORL and expanding its applications across various domains.

A APPENDIX A: THEORETICAL ANALYSIS OF THE DIRECTIONAL ANGLE GUIDED MORL ALGORITHM

This section introduces a theoretical analysis of the proposed PD-MORL algorithm. We follow the theoretical framework for value-based MORL algorithms proposed by Yang et al. [1], which is based on Banach's Fixed-Point Theorem (also known as Contraction Mapping Theorem). This theorem states that every *contraction* on a *complete* metric space has a unique fixed-point. Considering this, we define (i) contraction operators and (ii) a metric space to design our value-based MORL algorithm.

A.1 Metric Space

In standard Q-learning, the value space is defined as $\mathcal{Q} \in \mathbb{R}^{\mathcal{S}\times\mathcal{A}}$, containing all bounded functions $Q(s,a)$ which are the estimates of the total expected rewards when the agent is at state s, taking action a. We extend it to a multi-objective value space by defining the value space as $\mathcal{Q} \in \mathbb{R}^{L^{\mathcal{S}\times\mathcal{A}}}$, containing all bounded functions $\mathbf{Q}(s,a,\boldsymbol{\omega})$ which are the estimates of expected total rewards under preference $\boldsymbol{\omega} \in \mathbb{R}^{L}_{\geqslant 0} : \sum_{i=0}^{L} \omega_i = 1$. We then define a metric in this value space as:

$$d(\mathbf{Q},\mathbf{Q}') := \sup_{s\in\mathcal{S},a\in\mathcal{A},\boldsymbol{\omega}\in\Omega} |\boldsymbol{\omega}^{T}(\mathbf{Q}(s,a,\boldsymbol{\omega}) - \mathbf{Q}'(s,a,\boldsymbol{\omega}))|. \tag{A.1}$$

This metric gives the distance between $\mathbf{Q}$ and $\mathbf{Q}'$ as the supremum norm of the scalarized distance between these two vectors. Notice that d should satisfy the following axioms to be considered as a metric:

- **Non-negativity**: $d(\mathbf{Q},\mathbf{Q}') \geqslant 0$. This axiom holds for our metric d. The definition of supremum norm is the largest value of a set of absolute values, and thus, it is greater than or equal to zero.

- **Symmetry**: $d(\mathbf{Q}, \mathbf{Q}') = d(\mathbf{Q}', \mathbf{Q})$. Similarly to above axiom, this axiom also holds for d since absolute values are considered in supremum norm.

- **Triangle Inequality**: $d(\mathbf{Q}, \mathbf{Q}') \leqslant d(\mathbf{Q}, \mathbf{Q}'') + d(\mathbf{Q}'', \mathbf{Q}')$. This axiom holds for metric d as shown by the following proof:

$$\begin{aligned}\sup_{\substack{s\in\mathcal{S},a\in\mathcal{A}\\ \omega\in\Omega}} |\omega^T(\mathbf{Q}(s,a,\omega) - \mathbf{Q}''(s,a,\omega))| + \sup_{\substack{s\in\mathcal{S},a\in\mathcal{A}\\ \omega\in\Omega}} |\omega^T(\mathbf{Q}''(s,a,\omega) - \mathbf{Q}'(s,a,\omega))| \\ \geqslant |\omega^T(\mathbf{Q}(s,a,\omega) - \mathbf{Q}''(s,a,\omega))| + |\omega^T(\mathbf{Q}''(s,a,\omega) - \mathbf{Q}'(s,a,\omega))| \\ \geqslant |\omega^T(\mathbf{Q}(s,a,\omega) - \mathbf{Q}'(s,a,\omega))|\end{aligned}$$

  $\sup_{\substack{s\in\mathcal{S},a\in\mathcal{A}\\ \omega\in\Omega}} |\omega^T(\mathbf{Q}(s,a,\omega) - \mathbf{Q}''(s,a,\omega))| + \sup_{\substack{s\in\mathcal{S},a\in\mathcal{A}\\ \omega\in\Omega}} |\omega^T(\mathbf{Q}''(s,a,\omega) - \mathbf{Q}'(s,a,\omega))|$ is the upper bound on the set $|\omega^T(\mathbf{Q}(s,a,\omega) - \mathbf{Q}'(s,a,\omega))|$. Hence, by definition of supremum norm (least upper bound):

$$\begin{aligned}\sup_{\substack{s\in\mathcal{S},a\in\mathcal{A}\\ \omega\in\Omega}} |\omega^T(\mathbf{Q}(s,a,\omega) - \mathbf{Q}'(s,a,\omega))| \leqslant \sup_{\substack{s\in\mathcal{S},a\in\mathcal{A}\\ \omega\in\Omega}} |\omega^T(\mathbf{Q}(s,a,\omega) - \mathbf{Q}''(s,a,\omega))| \\ + \sup_{\substack{s\in\mathcal{S},a\in\mathcal{A}\\ \omega\in\Omega}} |\omega^T(\mathbf{Q}''(s,a,\omega) - \mathbf{Q}'(s,a,\omega))|\end{aligned}$$

- **Identity of indiscernibles**: $d(\mathbf{Q}, \mathbf{Q}') = 0 \Leftrightarrow \mathbf{Q} = \mathbf{Q}'$. This axiom does not hold for metric d since the dot product of different $\mathbf{Q}(s, a, \omega)$ functions may result in same value for a specific preference vector ω.

In summary, the metric d is a pseudo-metric since the identity of indiscernibles does not hold for it.

In the following Theorem, we showed that the metric space equipped with this metric is complete since the limit point of the sequence of operators is required to be in this space.

Theorem A.1 (Multi-objective Metric Space $(\mathcal{Q}, d)$ is Complete)**.** *The metric space* $(\mathcal{Q}, d)$ *is complete and every Cauchy sequence* $\mathbf{Q}_n(s, a, \omega)$ *is convergent in metric space* $(\mathcal{Q}, d)$ $\forall\, s, a, \omega \in \mathcal{S}, \mathcal{A}, \Omega$.

Proof. Let $\mathbf{Q}_n(s, a, \boldsymbol{\omega})$ be a Cauchy sequence in $\mathcal{Q}$. Given $\epsilon > 0$, there exist $n, m \geqslant N > 0$ such that $\sup_{s\in\mathcal{S}, a\in\mathcal{A}, \boldsymbol{\omega}\in\Omega} |\boldsymbol{\omega}^T(\mathbf{Q}_n(s, a, \boldsymbol{\omega}) - \mathbf{Q}_m(s, a, \boldsymbol{\omega}))| < \epsilon$. Hence,

$$|\boldsymbol{\omega}^T(\mathbf{Q}_n(s, a, \boldsymbol{\omega}) - \mathbf{Q}_m(s, a, \boldsymbol{\omega}))| \leqslant \sup_{\substack{s\in\mathcal{S} \\ a\in\mathcal{A} \\ \boldsymbol{\omega}\in\Omega}} |\boldsymbol{\omega}^T(\mathbf{Q}_n(s, a, \boldsymbol{\omega}) - \mathbf{Q}_m(s, a, \boldsymbol{\omega}))| < \epsilon,\ \forall\, s, a, \boldsymbol{\omega} \in \mathcal{S}, \mathcal{A}, \Omega$$

This implies that for each $s, a, \boldsymbol{\omega} \in \mathcal{S}, \mathcal{A}, \Omega$, the sequence of L-dimensional real numbers $\mathbf{Q}_n(s, a, \boldsymbol{\omega})$ is a Cauchy sequence. Since $\mathbb{R}^L$ is complete, $\mathbf{Q}_n(s, a, \boldsymbol{\omega})$ is convergent. Let $\mathbf{Q}(s, a, \boldsymbol{\omega}) = \lim_{n\to\infty} \mathbf{Q}_n(s, a, \boldsymbol{\omega})$. Cauchy sequence in a normed space must be bounded. Let there be an $M > 0$ such that $|\boldsymbol{\omega}^T(\mathbf{Q}_n(s, a, \boldsymbol{\omega}))| \leqslant \sup_{s\in\mathcal{S}, a\in\mathcal{A}, \boldsymbol{\omega}\in\Omega} |\boldsymbol{\omega}^T(\mathbf{Q}_n(s, a, \boldsymbol{\omega}))| \leqslant M,\ \forall\, s \in \mathcal{S}, a \in \mathcal{A}, \boldsymbol{\omega} \in \Omega$. Taking $n \to \infty$, we find $|\boldsymbol{\omega}^T(\mathbf{Q}(s, a, \boldsymbol{\omega}))| = \lim_{n\to\infty} \boldsymbol{\omega}^T(\mathbf{Q}_n(s, a, \boldsymbol{\omega})) \leqslant M,\ \forall\, s \in \mathcal{S}, a \in \mathcal{A}, \boldsymbol{\omega} \in \Omega$. This shows that $\mathbf{Q}(s, a, \boldsymbol{\omega})$ is a bounded function and thus $\mathbf{Q} \in \mathcal{Q}$. For all $n \geqslant N$,

$$|\boldsymbol{\omega}^T(\mathbf{Q}_n(s, a, \boldsymbol{\omega}) - \mathbf{Q}(s, a, \boldsymbol{\omega}))| = \lim_{m\to\infty} |\boldsymbol{\omega}^T(\mathbf{Q}_n(s, a, \boldsymbol{\omega}) - \mathbf{Q}_m(s, a, \boldsymbol{\omega}))| \leqslant \epsilon,\ \forall\, s, a, \boldsymbol{\omega} \in \mathcal{S}, \mathcal{A}, \Omega$$

and hence $\forall\ n \geqslant N, \sup_{s\in\mathcal{S}, a\in\mathcal{A}, \boldsymbol{\omega}\in\Omega} |\boldsymbol{\omega}^T(\mathbf{Q}_n(s, a, \boldsymbol{\omega}) - \mathbf{Q}(s, a, \boldsymbol{\omega}))| \leqslant \epsilon$. This implies that $\mathbf{Q}(s, a, \boldsymbol{\omega})$ is the limit of $\mathbf{Q}_n(s, a, \boldsymbol{\omega})$ and proves that $\mathbf{Q}_n(s, a, \boldsymbol{\omega})$ is convergent in $\mathcal{Q}$.

This completes our proof that the metric space $(\mathcal{Q}, d)$ is complete. □

A.2 Multi-Objective Bellman's Evaluation and Preference-Driven Optimality Operators

Given a policy π and sampled transition τ, we can define multi-objective Bellman's evaluation operator $\mathcal{T}_\pi$ using the metric space $(\mathcal{Q}, d)$ as:

$$(\mathcal{T}_\pi \mathbf{Q})(s, a, \boldsymbol{\omega}) := \mathbf{r}(s, a) + \gamma \mathbb{E}_{\mathcal{T}\sim(\mathcal{P},\pi)} \mathbf{Q}(s', a', \boldsymbol{\omega}) \tag{A.2}$$

where $(s', a', \boldsymbol{\omega})$ denotes the next state-action-preference pair and $\gamma \in (0, 1)$ is the discount factor.

Theorem A.2 (Multi-objective Bellman's Evaluation Operator is Contraction)**.** *Let* $(\mathcal{Q}, \mathrm{d})$ *be a complete metric space (as in Theorem 2.1). Let* $\mathbf{Q}$ *and* $\mathbf{Q}'$ *be any two multi-objective Q-value functions in this space. The multi-objective Bellman's evaluation operator is a contraction and* $\mathrm{d}(\mathcal{T}_\pi \mathbf{Q}, \mathcal{T}_\pi \mathbf{Q}') \leqslant \gamma \mathrm{d}(\mathbf{Q}, \mathbf{Q}')$ *holds for the Lipschitz constant* γ *(the discount factor).*

Proof. We start by expanding the expression $\mathrm{d}(\mathcal{T}_\pi \mathbf{Q}, \mathcal{T}_\pi \mathbf{Q}')$:

$$
\begin{aligned}
\mathrm{d}(\mathcal{T}_\pi \mathbf{Q}, \mathcal{T}_\pi \mathbf{Q}') &= \sup_{\substack{s \in \mathcal{S}, a \in \mathcal{A} \\ \boldsymbol{\omega} \in \Omega}} |\boldsymbol{\omega}^\intercal (\mathcal{T}_\pi \mathbf{Q}(s, a, \boldsymbol{\omega}) - \mathcal{T}_\pi \mathbf{Q}'(s, a, \boldsymbol{\omega}))| \\
&= \sup_{\boldsymbol{\omega} \in \Omega} |\gamma \boldsymbol{\omega}^\intercal \mathbb{E}_{\mathcal{T} \sim (\mathcal{P}, \pi)} \mathbf{Q}(s', a', \boldsymbol{\omega}) - \gamma \boldsymbol{\omega}^\intercal \mathbb{E}_{\mathcal{T} \sim (\mathcal{P}, \pi)} \mathbf{Q}'(s', a', \boldsymbol{\omega})| \\
&= \gamma \cdot \sup_{\boldsymbol{\omega} \in \Omega} |\boldsymbol{\omega}^\intercal \mathbb{E}_{\mathcal{T} \sim (\mathcal{P}, \pi)} (\mathbf{Q}(s', a', \boldsymbol{\omega}) - \mathbf{Q}'(s', a', \boldsymbol{\omega}))| \\
&\leqslant \gamma \cdot \sup_{\boldsymbol{\omega} \in \Omega} \boldsymbol{\omega}^\intercal \mathbb{E}_{\mathcal{T} \sim (\mathcal{P}, \pi)} |\mathbf{Q}(s', a', \boldsymbol{\omega}) - \mathbf{Q}'(s', a', \boldsymbol{\omega})| \quad (|\mathbb{E}[\cdot]| \leqslant \mathbb{E}[|\cdot|]) \\
&= \gamma \cdot \sup_{\boldsymbol{\omega} \in \Omega} \mathbb{E}_{\mathcal{T} \sim (\mathcal{P}, \pi)} |\boldsymbol{\omega}^\intercal (\mathbf{Q}(s', a', \boldsymbol{\omega}) - \mathbf{Q}'(s', a', \boldsymbol{\omega}))| \\
&\leqslant \gamma \cdot \sup_{\boldsymbol{\omega} \in \Omega} \sup_{\substack{s' \in \mathcal{S} \\ a' \in \mathcal{A} \\ \boldsymbol{\omega} \in \Omega}} |\boldsymbol{\omega}^\intercal (\mathbf{Q}(s', a', \boldsymbol{\omega}) - \mathbf{Q}'(s', a', \boldsymbol{\omega}))| \quad (\mathbb{E}[|\cdot|] \leqslant \sup |\cdot|) \\
&= \gamma \cdot \sup_{\substack{s' \in \mathcal{S} \\ a' \in \mathcal{A} \\ \boldsymbol{\omega} \in \Omega}} |\boldsymbol{\omega}^\intercal (\mathbf{Q}(s', a', \boldsymbol{\omega}) - \mathbf{Q}'(s', a', \boldsymbol{\omega}))| = \gamma \cdot \mathrm{d}(\mathbf{Q}, \mathbf{Q}') \\
\mathrm{d}(\mathcal{T}_\pi \mathbf{Q}, \mathcal{T}_\pi \mathbf{Q}') &\leqslant \gamma \cdot \mathrm{d}(\mathbf{Q}, \mathbf{Q}')
\end{aligned}
$$

To avoid confusion, the last step (Step 7) has only one supremum norm since the outside supremum does not see any variable. The definition of the variables ends with the inside supremum. This completes our proof that multi-objective Bellman's evaluation operator, $\mathcal{T}_\pi$, is contraction. □

We define a *preference-driven optimality operator* $\mathcal{T}$ by adding a cosine similarity term between preference vectors and state-action values to the multi-objective

Bellman's optimality operator as:

$$(\mathcal{T}\mathbf{Q})(s,a,\boldsymbol{\omega}) := \mathbf{r}(s,a)+\gamma\mathbb{E}_{s'\sim\mathcal{P}(\cdot|s,a)}\mathbf{Q}(s', \sup_{a'\in\mathcal{A}}(S_c(\boldsymbol{\omega},\mathbf{Q}(s',a',\boldsymbol{\omega}))\cdot(\boldsymbol{\omega}^\top\mathbf{Q}(s',a',\boldsymbol{\omega}))),\boldsymbol{\omega}) \tag{A.3}$$

where $S_c(\boldsymbol{\omega},\mathbf{Q}(s',a',\boldsymbol{\omega}))$ denotes the cosine similarity between preference vector and Q-value. $\sup_{a'\in\mathcal{A}}(S_c(\boldsymbol{\omega},\mathbf{Q}(s',a',\boldsymbol{\omega}))\cdot(\boldsymbol{\omega}^\top\mathbf{Q}(s',a',\boldsymbol{\omega})))$ yields the action a' that maximizes the multiplication inside the supremum. This term enables our optimality operator to choose actions that align the preferences with the Q-values and maximize the target value, as elaborated in Section 2.4.2 under the preference alignment subtitle.

Theorem A.3 (Preference-driven Multi-objective Bellman's Optimality Operator is Contraction)**.** *Let* $(\mathcal{Q},d)$ *be a complete metric space. Let* $\mathbf{Q}$ *and* $\mathbf{Q}'$ *be any two multi-objective Q-value functions in this space. The preference-driven multi-objective Bellman's optimality operator is a contraction and* $d(\mathcal{T}\mathbf{Q},\mathcal{T}\mathbf{Q}')\leqslant\gamma d(\mathbf{Q},\mathbf{Q}')$ *holds for the Lipschitz constant* γ *(the discount factor).*

Proof. We use $S_c(\boldsymbol{\omega},\mathbf{Q}(s',a',\boldsymbol{\omega}))$ as S_c and $S_c(\boldsymbol{\omega},\mathbf{Q}'(s',a'',\boldsymbol{\omega}))$ as S_c' in the proof

for notational simplicity. We start by expanding the expression $d(\mathcal{T}\mathbf{Q}, \mathcal{T}\mathbf{Q}')$:

$$
\begin{aligned}
d(\mathcal{T}\mathbf{Q}, \mathcal{T}\mathbf{Q}') &= \sup_{\substack{s\in\mathcal{S},a\in\mathcal{A}\\ \boldsymbol{\omega}\in\Omega}} |\boldsymbol{\omega}^\intercal(\mathcal{T}\mathbf{Q}(s,a,\boldsymbol{\omega}) - \mathcal{T}\mathbf{Q}'(s,a,\boldsymbol{\omega}))| \\
&= \sup_{\substack{s\in\mathcal{S},a\in\mathcal{A}\\ \boldsymbol{\omega}\in\Omega}} \Big| \gamma\boldsymbol{\omega}^\intercal \mathbb{E}_{s'\sim\mathcal{P}(\cdot|s,a)} \mathbf{Q}(s', \sup_{a'\in\mathcal{A}}(S_c)\cdot(\boldsymbol{\omega}^\intercal\mathbf{Q}(s',a',\boldsymbol{\omega})),\boldsymbol{\omega}) \\
&\quad - \gamma\boldsymbol{\omega}^\intercal \mathbb{E}_{s'\sim\mathcal{P}(\cdot|s,a)} \mathbf{Q}'(s', \sup_{a''\in\mathcal{A}}(S'_c)\cdot(\boldsymbol{\omega}^\intercal\mathbf{Q}'(s',a'',\boldsymbol{\omega})),\boldsymbol{\omega}) \Big| \\
&= \gamma\cdot \sup_{\substack{s\in\mathcal{S},a\in\mathcal{A}\\ \boldsymbol{\omega}\in\Omega}} \Big| \mathbb{E}_{s'\sim\mathcal{P}(\cdot|s,a)} \Big[\boldsymbol{\omega}^\intercal \Big(\mathbf{Q}(s', \sup_{a'\in\mathcal{A}}(S_c)\cdot(\boldsymbol{\omega}^\intercal\mathbf{Q}(s',a',\boldsymbol{\omega})),\boldsymbol{\omega}) \\
&\quad - \mathbf{Q}'(s', \sup_{a''\in\mathcal{A}}(S'_c)\cdot(\boldsymbol{\omega}^\intercal\mathbf{Q}'(s',a'',\boldsymbol{\omega})),\boldsymbol{\omega}) \Big) \Big] \Big| \\
&\leqslant \gamma\cdot \sup_{\substack{s\in\mathcal{S},a\in\mathcal{A}\\ \boldsymbol{\omega}\in\Omega}} \mathbb{E}_{s'\sim\mathcal{P}(\cdot|s,a)} \Big[\Big| \boldsymbol{\omega}^\intercal \Big(\mathbf{Q}(s', \sup_{a'\in\mathcal{A}}(S_c)\cdot(\boldsymbol{\omega}^\intercal\mathbf{Q}(s',a',\boldsymbol{\omega})),\boldsymbol{\omega}) \\
&\quad - \mathbf{Q}'(s', \sup_{a''\in\mathcal{A}}(S'_c)\cdot(\boldsymbol{\omega}^\intercal\mathbf{Q}'(s',a'',\boldsymbol{\omega})),\boldsymbol{\omega}) \Big) \Big| \Big] \quad (|\mathbb{E}[\cdot]| \leqslant \mathbb{E}[|\cdot|]) \\
&\leqslant \gamma\cdot \sup_{\substack{s\in\mathcal{S},a\in\mathcal{A}\\ \boldsymbol{\omega}\in\Omega}} \sup_{s'\in\mathcal{S},\boldsymbol{\omega}\in\Omega} \Big| \boldsymbol{\omega}^\intercal \Big(\mathbf{Q}(s', \sup_{a'\in\mathcal{A}}(S_c)\cdot(\boldsymbol{\omega}^\intercal\mathbf{Q}(s',a',\boldsymbol{\omega})),\boldsymbol{\omega}) \\
&\quad - \mathbf{Q}'(s', \sup_{a''\in\mathcal{A}}(S'_c)\cdot(\boldsymbol{\omega}^\intercal\mathbf{Q}'(s',a'',\boldsymbol{\omega})),\boldsymbol{\omega}) \Big) \Big| \quad (\mathbb{E}[|\cdot|] \leqslant \sup|\cdot|) \\
&= \gamma\cdot \sup_{s'\in\mathcal{S},\boldsymbol{\omega}\in\Omega} \Big| \boldsymbol{\omega}^\intercal \Big(\mathbf{Q}(s', \sup_{a'\in\mathcal{A}}(S_c)\cdot(\boldsymbol{\omega}^\intercal\mathbf{Q}(s',a',\boldsymbol{\omega})),\boldsymbol{\omega}) \\
&\quad - \mathbf{Q}'(s', \sup_{a''\in\mathcal{A}}(S'_c)\cdot(\boldsymbol{\omega}^\intercal\mathbf{Q}'(s',a'',\boldsymbol{\omega})),\boldsymbol{\omega}) \Big) \Big| \quad (\textit{Rearrange supremums})
\end{aligned}
$$

Let a' be the action that maximizes $(S_c)\cdot(\boldsymbol{\omega}^\intercal\mathbf{Q}(s',a',\boldsymbol{\omega}))$ for state s' and preference $\boldsymbol{\omega}$, then we have :

$$
d(\mathcal{T}\mathbf{Q}, \mathcal{T}\mathbf{Q}') \leqslant \gamma\cdot \sup_{s'\in\mathcal{S},\boldsymbol{\omega}\in\Omega} \Big| \boldsymbol{\omega}^\intercal \Big(\mathbf{Q}(s',a',\boldsymbol{\omega}) - \mathbf{Q}'(s', \sup_{a''\in\mathcal{A}}(S'_c)\cdot(\boldsymbol{\omega}^\intercal\mathbf{Q}'(s',a'',\boldsymbol{\omega})),\boldsymbol{\omega}) \Big) \Big|
$$

W.l.o.g we assume $\boldsymbol{\omega}^\intercal\mathbf{Q}(s',a',\boldsymbol{\omega}) - \boldsymbol{\omega}^\intercal\mathbf{Q}'(s', \sup_{a''\in\mathcal{A}}(S'_c)\cdot(\boldsymbol{\omega}^\intercal\mathbf{Q}'(s',a'',\boldsymbol{\omega})),\boldsymbol{\omega}) \geqslant 0$. Proof is similar for the other inequality case. Since $\boldsymbol{\omega}^\intercal\mathbf{Q}'(s',a',\boldsymbol{\omega}) \leqslant \boldsymbol{\omega}^\intercal\mathbf{Q}'(s', \sup_{a''\in\mathcal{A}}(S'_c)\cdot$

$(\boldsymbol{\omega}^\intercal \mathbf{Q}'(s', a'', \boldsymbol{\omega})), \boldsymbol{\omega})$, we have:

$$d(\mathcal{T}\mathbf{Q}, \mathcal{T}\mathbf{Q}') \leqslant \gamma \cdot \sup_{\substack{s' \in \mathcal{S}, a \in \mathcal{A} \\ \boldsymbol{\omega} \in \Omega}} \left| \boldsymbol{\omega}^\intercal \Big(\mathbf{Q}(s', a', \boldsymbol{\omega}) - \mathbf{Q}'(s', a', \boldsymbol{\omega}) \Big) \right| = \gamma \cdot d(\mathbf{Q}, \mathbf{Q}')$$

$$d(\mathcal{T}\mathbf{Q}, \mathcal{T}\mathbf{Q}') \leqslant \gamma \cdot d(\mathbf{Q}, \mathbf{Q}')$$

This completes our proof that preference-driven multi-objective Bellman's optimality operator, $\mathcal{T}$, is contraction. □

Theorem A.2 and Theorem A.3 state that multi-objective evaluation and optimality operators are contractions. They ensure that we can apply our optimality operator in Equation A.3 iteratively to obtain the optimal multi-objective value function given by Theorem A.4 and Theorem A.5 below, respectively.

Theorem A.4 (Preference-driven Multi-objective Optimality Operator Converges to a Fixed-Point)**.** *Let $(\mathcal{Q}, d)$ be a complete metric space which is proved above and let $\mathcal{T} : \mathcal{Q} \to \mathcal{Q}$ be a contraction on $\mathcal{Q}$ with modulus γ. Then, $\mathcal{T}$ has a unique fixed-point $\mathbf{Q}^* \in \mathcal{Q}$ such that $\mathcal{T}(\mathbf{Q}) = \mathbf{Q}^*$.*

Proof. Let $\mathbf{Q}_0 \in \mathcal{Q}$ and define a sequence $(\mathbf{Q}_n)$ where $\mathbf{Q}_{n+1} = \mathcal{T}(\mathbf{Q}_n), n = 1, 2, \ldots$

$$\begin{aligned} d(\mathbf{Q}_{n+1}, \mathbf{Q}_n) &= d(\mathcal{T}(\mathbf{Q}_n), \mathcal{T}(\mathbf{Q}_{n-1})) \\ &\leqslant \gamma d(\mathbf{Q}_n, \mathbf{Q}_{n-1}) = \gamma d(\mathcal{T}(\mathbf{Q}_{n-1}), \mathcal{T}(\mathbf{Q}_{n-2})) \\ &\leqslant \gamma^2 d(\mathbf{Q}_{n-1}, \mathbf{Q}_{n-2}) \\ &\vdots \\ &\leqslant \gamma^m d(\mathbf{Q}_1, \mathbf{Q}_0) \end{aligned}$$

Hence, for $m > n \; \forall \; m, n \in \mathbb{N}$ by the triangle inequality we have

$$\begin{aligned} d(\mathbf{Q}_m, \mathbf{Q}_n) &\leqslant d(\mathbf{Q}_m, \mathbf{Q}_{m-1}) + d(\mathbf{Q}_{m-1}, \mathbf{Q}_{m-2}) + \ \ldots \ + d(\mathbf{Q}_{n+1}, \mathbf{Q}_n) \\ &\leqslant (\gamma^{m-1} + \gamma^{m-2} + \ \ldots \ + \gamma^n) d(\mathbf{Q}_1, \mathbf{Q}_0) \\ &\leqslant \frac{\gamma^n}{1-\gamma} d(\mathbf{Q}_1, \mathbf{Q}_0) \end{aligned}$$

Therefore, $\mathbf{Q}_n$ is Cauchy: for $\epsilon > 0$, let N be large enough that $\frac{\gamma^n}{1-\gamma} d(\mathbf{Q}_1, \mathbf{Q}_0) < \epsilon$, which ensures that $n, m > N \Rightarrow d(\mathbf{Q}_n, \mathbf{Q}_m) < \epsilon$. Since $(\mathcal{Q}, d)$ is complete, $\mathbf{Q}_n$ converges to $\mathbf{Q}^* \in \mathcal{Q}$. Now, we show that $\mathbf{Q}^*$ is indeed a fixed point since $\mathbf{Q}_n$ converges to $\mathbf{Q}^*$ and $\mathcal{T}$ is continuous.

$$\mathbf{Q}^* = \lim_{n\to\infty} \mathbf{Q}_n = \lim_{n\to\infty} \mathcal{T}(\mathbf{Q}_{n-1}) = \mathcal{T}(\lim_{n\to\infty} \mathbf{Q}_{n-1}) = \mathcal{T}(\mathbf{Q}^*)$$

Finally, $\mathcal{T}$ cannot have more than one fixed point in $(\mathcal{Q}, d)$, since any pair of distinct fixed points $\mathbf{Q}_1^*$ and $\mathbf{Q}_2^*$ would contradict the contraction of $\mathcal{T}$:

$$d(\mathcal{T}(\mathbf{Q}_1^*), \mathcal{T}(\mathbf{Q}_2^*)) = d(\mathbf{Q}_1^*, \mathbf{Q}_2^*) > \gamma d(\mathbf{Q}_1^*, \mathbf{Q}_2^*)$$

This completes our proof that multi-objective optimality operator converges to $\mathbf{Q}^*$. □

Theorem A.5 (Optimal Fixed Point of Optimality Operator)**.** *Let $\mathbf{Q}^* \in \mathcal{Q}$ be the optimal multi-objective value function, such that it takes multi-objective Q-value corresponding to the supremum of expected discounted rewards under a policy π then $\mathbf{Q}^* = \mathcal{T}(\mathbf{Q}^*)$.*

Proof. We start by defining the optimal Q-function, $\mathbf{Q}^*$ as:

$$\mathbf{Q}^*(s, a, \boldsymbol{\omega}) = \arg_{\mathbf{Q}} \sup_{\pi\in\Pi} \boldsymbol{\omega}^\intercal \mathbb{E}_{\mathcal{T}\sim(\mathcal{P},\pi)|s_0=s,a_0=a} \Big[\sum_{t=0}^{\infty} \gamma^t \mathbf{r}(s_t, a_t) \Big].$$

From Theorem A.4 we observe that $\lim_{n\to\infty} d(\mathcal{T}^n(\mathbf{Q}), \mathbf{Q}^*) = 0$ for any $\mathbf{Q} \in \mathcal{Q}$. This suggests that $\boldsymbol{\omega}^\intercal \mathcal{T}(\mathbf{Q}^*)(s, a, \boldsymbol{\omega}) = \boldsymbol{\omega}^\intercal \mathbf{Q}^*(s, a, \boldsymbol{\omega})$ from the definition of metric

d. Expanding the $\boldsymbol{\omega}^T \mathcal{T}(\mathbf{Q}^*)(s, a, \boldsymbol{\omega})$, we have:

$$\boldsymbol{\omega}^T \mathcal{T}(\mathbf{Q}^*)(s, a, \boldsymbol{\omega}) = \boldsymbol{\omega}^T \mathbf{r}(s, a) + \gamma \cdot \boldsymbol{\omega}^T \mathbb{E}_{s' \sim \mathcal{P}(\cdot|s,a)} \mathbf{Q}^*(s', \sup_{a' \in \mathcal{A}} (S_c(\boldsymbol{\omega}, \mathbf{Q}^*(s', a', \boldsymbol{\omega})) \cdot (\boldsymbol{\omega}^T \mathbf{Q}^*(s', a', \boldsymbol{\omega}))), \boldsymbol{\omega}).$$

Let a' is the action that maximizes the $\sup_{a' \in \mathcal{A}} (S_c(\boldsymbol{\omega}, \mathbf{Q}^*(s', a', \boldsymbol{\omega})) \cdot (\boldsymbol{\omega}^T \mathbf{Q}^*(s', a', \boldsymbol{\omega}))$ for state s' and preference $\boldsymbol{\omega}$. Then, by substituting Q^* into $\mathcal{T}(\mathbf{Q}^*)$ we have:

$$\begin{aligned}
\boldsymbol{\omega}^T \mathcal{T}(\mathbf{Q}^*)(s, a, \boldsymbol{\omega}) &= \boldsymbol{\omega}^T \mathbf{r}(s, a) + \gamma \cdot \boldsymbol{\omega}^T \mathbb{E}_{s' \sim \mathcal{P}(\cdot|s,a)} \Big[\arg_{\mathbf{Q}} \sup_{\pi \in \Pi} \boldsymbol{\omega}^T \mathbb{E}_{\mathcal{T} \sim (\mathcal{P}, \pi) | s_0 = s', a_0 = a'} \Big[\sum_{t=0}^{\infty} \gamma^t \mathbf{r}(s_t, a_t) \Big] \Big] \\
&= \boldsymbol{\omega}^T \mathbf{r}(s, a) + \gamma \cdot \boldsymbol{\omega}^T \arg_{\mathbf{Q}} \sup_{\pi \in \Pi} \boldsymbol{\omega}^T \mathbb{E}_{\substack{\mathcal{T} \sim (\mathcal{P}, \pi) \\ s_0 \sim \mathcal{P}(\cdot|s,a)}} \sum_{t=0}^{\infty} \gamma^t \mathbf{r}(s_t, a_t)
\end{aligned}$$

Now, we merge $\mathbf{r}(s, a)$ with the summation operation by rearranging the conditions of the expectation operator and obtain:

$$= \boldsymbol{\omega}^T \Big(\arg_{\mathbf{Q}} \sup_{\pi \in \Pi} \boldsymbol{\omega}^T \mathbb{E}_{\mathcal{T} \sim (\mathcal{P}, \pi) | s_0 = s, a_0 = a} \Big[\sum_{t=0}^{\infty} \gamma^t \mathbf{r}(s_t, a_t) \Big] \Big) = \boldsymbol{\omega}^T \mathbf{Q}^*(s, a, \boldsymbol{\omega})$$

$$\boldsymbol{\omega}^T \mathcal{T}(\mathbf{Q}^*)(s, a, \boldsymbol{\omega}) = \boldsymbol{\omega}^T \mathbf{Q}^*(s, a, \boldsymbol{\omega})$$

This completes our proof that optimal multi-objective value function $\mathbf{Q}^*$ is the fixed point of the preference-driven multi-objective optimality operator $\mathcal{T}$. □

B APPENDIX B: PD-MORL IMPLEMENTATION-TRAINING DETAILS AND ADDITIONAL EXPERIMENTAL RESULTS

This section provides details on the Preference-Driven MO-DDQN-HER and MO-TD3-HER algorithms. It expands the explanation of the algorithms with implementation details. Then, it explains the training details for all benchmarks. Furthermore, we provide additional experimental results for discrete MORL and multi-objective continuous control benchmarks.

B.1 Preference-Driven MO-DDQN-HER and MO-TD3-HER

B.1.1 Hindsight Experience Replay

We randomly sample a preference vector $(\boldsymbol{\omega} \in \Omega : \sum_{i=0}^{L} \omega_i = 1)$ from a uniform distribution in each episode during training. Depending on the problem, the number of transitions observed by interacting with the environment for some preferences may be underrepresented. This behavior creates a bias towards over-represented preferences, and the network cannot learn to cover the entire preference space. Therefore, we employ hindsight experience replay buffer (HER) [31], where every transition is also stored with N_ω randomly sampled preferences $(\boldsymbol{\omega}' \in \Omega : \sum_{i=0}^{L} \omega'_i = 1)$ from a uniform distribution different than the original preference of the transition. Specifically, for each transition $(s, a, \mathbf{r}, s', \boldsymbol{\omega}, done)$, additional N_ω transitions $(s, a, \mathbf{r}, s', \boldsymbol{\omega'}, done)$ are also stored in the experience replay buffer. This strategy provides efficient exploration and generalizability to the agent. Using HER, the agent learns to recover from undesired states and continue to align itself with its original preference.

B.1.2 Exploration in Parallel

To further increase the sample efficiency of our algorithm, we equally divide the preference space into C_p sub-spaces ($\tilde{\Omega}$), where C_p denotes the number of child processes. The number of child processes C_p is set to 10 for all benchmarks. The agent is shared among these child processes and the main process. In each child process, for each episode, we randomly sample a preference vector from child's preference sub-space ($\boldsymbol{\omega} \in \tilde{\Omega} : \sum_{i=0}^{L} \omega_i = 1$). These C_p child processes run in parallel to collect transitions. After collecting a transition, each child process transfers it to the main process and waits for other child processes. After the main process receives transitions from every child process and stores them using HER, the child processes continue to collect transitions. Since we collect and store C_p transitions at every child-main process loop, networks are also updated C_p times. Parallel exploration and HER together provides efficient exploration.

B.1.3 Preference Alignment with Interpolation

A solution in the Pareto front may not align perfectly with its preference vector. To mitigate this adverse effect, we fit a multi-dimensional interpolator to project the original preference vectors ($\boldsymbol{\omega} \in \Omega$) to normalized solution space to align preferences with the multi-objective solutions. Here, normalization is the process of obtaining unit vectors for these solutions. For example, the normalized vector for a solution f is described as $\hat{f} = \frac{f}{|f|}$. We identify the key preferences and obtain solutions for each of these preferences without HER. We set key preference points as $\boldsymbol{\omega}_j = 1 : j = i, \boldsymbol{\omega}_j = 0 : j \neq i \; \forall \; i, j \in \{0, \ldots, L\}$ where i^{th} element corresponds to the objective we try to maximize. A preference vector of $\boldsymbol{\omega} = \{\frac{1}{L}, \ldots, \frac{1}{L}\}$ is also added to this key preference set. We first obtain solutions for each preference in this key preference set by training the agent with a fixed preference vector and without using HER. We use these solutions first to obtain a normalized solution space. Then, we use the normalized solution space and key preference vectors to fit a multi-dimensional interpolator $I(\boldsymbol{\omega})$ to project the original preference vectors ($\boldsymbol{\omega} \in \Omega$) to the normalized solution space. We use radial basis function interpolation with

linear kernel in this work [162]. As a result, we obtain projected preference vectors ($\boldsymbol{\omega}_p$) that are incorporated in the cosine similarity term of the preference-driven optimality operator for the MO-DDQN-HER algorithm and in the directional angle term for the MO-TD3-HER algorithm. The interpolator is also updated during training as PD-MORL may find new non-dominated solutions for identified key preferences. To this end, at every episode, we evaluate our agent using the key preference set.

B.1.4 Details on MO-DDQN-HER

The main objective of this work is to obtain a single policy network to approximate the Pareto front that covers the entire preference space. For this purpose, we extend double deep Q-network (DDQN) [43] to a multi-objective version (MO-DDQN) with the preference-driven optimality operator. Algorithm 10 describes the training using the proposed MO-DDQN-HER.

We first initialize an empty buffer $\mathcal{D}$, Q-network and target Q-network with parameters θ and θ'. The preference space is then equally divided into C_p sub-spaces, where we initialize a child process for each sub-space. In each child process, for each episode, we randomly sample a preference vector from child's preference sub-space ($\boldsymbol{\omega} \in \tilde{\Omega} : \sum_{i=0}^{L} \omega_i = 1$). The agent interacts with the environment using ϵ-greedy policy and collect transition $(s, a, \mathbf{r}, s', \boldsymbol{\omega}, done)$. This transition is then transferred to the main process. After main process receives transitions from every child process, it stores transitions inside $\mathcal{D}$. For each transition, we also store N_ω transitions where a different preference ($\boldsymbol{\omega}' \in \Omega : \sum_{i=0}^{L} \omega'_i = 1$) is sampled. Specifically, for each transition $(s, a, \mathbf{r}, s', \boldsymbol{\omega}, done)$, additional N_ω transitions $(s, a, \mathbf{r}, s', \boldsymbol{\omega'}, done)$ are also stored in $\mathcal{D}$. We then sample a minibatch of transitions from $\mathcal{D}$ to update the network. To calculate the cosine similarity metric, we first project the preferences $\boldsymbol{\omega}$ to normalized solution space and obtain projected preferences $\boldsymbol{\omega}_p$. MO-DDQN-HER minimizes the following loss function at each step k:

$$L_k(\theta) = \mathbb{E}_{(s,a,\mathbf{r},s',\boldsymbol{\omega})\sim\mathcal{D}}\left[\left(\mathbf{y} - \mathbf{Q}(s, a, \boldsymbol{\omega}; \theta)\right)^2\right] \tag{B.1}$$

where $\mathbf{y} = \mathbf{r} + \gamma \mathbf{Q}(s', \sup_{a' \in A}(S_c(\boldsymbol{\omega}, \mathbf{Q}(s', a', \boldsymbol{\omega})) \cdot (\boldsymbol{\omega}^T \mathbf{Q}(s', a', \boldsymbol{\omega}))); \theta')$ denotes the preference-driven target value which is obtained using target Q-network's parameters θ'. Instead of MSE, Smooth L1 loss may also be used for this loss function. We soft update the target network at every time step k. Note that each child process runs for N time steps. Hence, in total, we collect $N \times C_p$ transitions.

Algorithm 10: Preference Driven MO-DDQN-HER

Input: Minibatch size N_m, Number of time steps N,
Discount factor γ, Target network update coefficient τ,
Multi-dimensional interpolator $I(\boldsymbol{\omega})$.
Initialize: Replay buffer $\mathcal{D}$, Current $\mathbf{Q}_\theta$ and target network $\mathbf{Q}_{\theta'} \leftarrow \mathbf{Q}_\theta$ with parameters θ and θ'.
for $n = 0$: N **do**
    `// Child Process`
    Initialize $t = 0$ and $done = False$.
    Reset the environment to randomly initialized state s_0.
    Sample a preference vector $\boldsymbol{\omega}$ from the subspace $\tilde{\Omega}$.
    while $done = False$ **do**
        Observe state s_t and select an action a_t ϵ-greedily:
        $$a_t = \begin{cases} a \in A & w.p. \quad \epsilon \\ \max_{a \in \mathcal{A}} \boldsymbol{\omega} \mathbf{Q}(s_t, a, \boldsymbol{\omega}; \theta), & w.p. \quad 1 - \epsilon \end{cases}$$
        Observe $\mathbf{r}$, s', and $done$.
        Transfer $(s_t, a_t, \mathbf{r}_t, s', \boldsymbol{\omega}, done)$ to main process.
    `// Main Process`
    Store the transition $(s_t, a_t, \mathbf{r}_t, s', \boldsymbol{\omega}, done)$ obtained from every child process in $\mathcal{D}$.
    Sample N_ω preferences $\boldsymbol{\omega}'$
    for $j = 1$: N_ω **do**
        Store transition $(s_t, a_t, \mathbf{r}_t, s', \boldsymbol{\omega}'_j, done)$ in $\mathcal{D}$
    Sample N_m transitions from $\mathcal{D}$.
    $\boldsymbol{\omega}_p \leftarrow I(\boldsymbol{\omega})$
    $\mathbf{y} \leftarrow \mathbf{r} + \gamma \mathbf{Q}(s', \sup_{a' \in A}(S_c(\boldsymbol{\omega}_p, \mathbf{Q}(s', a', \boldsymbol{\omega})) \cdot (\boldsymbol{\omega}^T \mathbf{Q}(s', a', \boldsymbol{\omega})); \theta')$
    $L_k(\theta) = \mathbb{E}_{(s,a,\mathbf{r},s',\boldsymbol{\omega}) \sim \mathcal{D}} \left[\left(\mathbf{y} - \mathbf{Q}(s, a, \boldsymbol{\omega}; \theta) \right)^2 \right]$
    Update θ_k by applying SGD to $L_k(\theta)$.
    Update target network parameters $\theta'_k \leftarrow \tau \theta_k + (1 - \tau) \theta'_k$

B.1.5 Details on MO-TD3-HER

We extend the TD3 algorithm to a multi-objective version using PD-MORL for problems with continuous action space. To incorporate the relation between preference vectors and Q-values for the multi-objective version of TD3, we include a directional angle term $g(\boldsymbol{\omega}, \mathbf{Q}(s, a, \boldsymbol{\omega}; \theta))$ to both actor's and critic's loss function. This directional angle term $g(\boldsymbol{\omega}, \mathbf{Q}(s, a, \boldsymbol{\omega}; \theta)) = \cos^{-1}(\frac{\boldsymbol{\omega}^T \mathbf{Q}(s,a,\boldsymbol{\omega};\theta)}{||\boldsymbol{\omega}_p|| \, ||\mathbf{Q}(s,a,\boldsymbol{\omega};\theta)||})$ denotes the angle between the preference vector $(\boldsymbol{\omega})$ and multi-objective Q-value $\mathbf{Q}(s, a, \boldsymbol{\omega}; \theta)$ and provides an alignment between preferences and the Q-values.

Similar to MO-DDQN-HER, we first initialize an empty buffer $\mathcal{D}$, critic networks $\mathbf{Q}_{\theta_1}$,$\mathbf{Q}_{\theta_2}$ and actor network π_ϕ with parameters θ_1, θ_2, and ϕ. We also initialize critic and actor target networks. The process of initialization of child processes is the same with MO-DDQN-HER. The agent interacts with the environment to collect transition $(s, a, \mathbf{r}, s', \boldsymbol{\omega}, \texttt{done})$ by selecting actions according to a policy with an exploration noise term ϵ. The utilization of HER for collected transitions is also the same as the MO-DDQN-HER. We then sample a minibatch of transitions from $\mathcal{D}$. The algorithm computes actions for the next state using the target actor network plus a target smoothing noise. In the conventional TD3 algorithm, both critics use a single target value to update their parameters calculated using whichever of the two critics gives a smaller Q-value. In multi-objective version of TD3, we modified this clipped double-Q-learning approach such that target $\mathbf{Q}$-value is calculated using whichever of the two critics gives a smaller $\boldsymbol{\omega}^T \mathbf{Q}$. To calculate the directional angle term, similar to MO-DDQN-HER, we project the preferences $\boldsymbol{\omega}$ to normalized solution space and obtain projected preferences $\boldsymbol{\omega}_p$. MO-TD3-HER minimizes the following loss function to update both critic networks at each step k:

$$L_{\texttt{critic}_k}(\theta_i) = \mathbb{E}_{(s,a,\mathbf{r},s',\boldsymbol{\omega})\sim\mathcal{D}}\left[\left(\mathbf{y} - \mathbf{Q}(s, a, \boldsymbol{\omega}; \theta_i)\right)^2\right] + \mathbb{E}_{(s,a,\boldsymbol{\omega})\sim\mathcal{D}}\left[g(\boldsymbol{\omega}_p, \mathbf{Q}(s, a, \boldsymbol{\omega}; \theta_i))\right] \tag{B.2}$$

where $\mathbf{y} = \mathbf{r} + \gamma \arg_{\mathbf{Q}} \min_{i=1,2} \boldsymbol{\omega}^T \mathbf{Q}(s', \tilde{a}, \boldsymbol{\omega}; \theta'_i)$ denotes the target value which is obtained using target critic network's parameters. Instead of MSE, Smooth L1 loss may also be used for this loss function. One of the essential tricks that TD3 has is

that it updates actor and target networks less frequently than the critics. The actor network is updated every p_{delay} step by maximizing the $\boldsymbol{\omega}^{\mathsf{T}}\mathbf{Q}$ while minimizing the directional angle term using the following loss:

$$\nabla_{\phi} L_{actor_k}(\phi) = \mathbb{E}_{(s,a,r,s',\boldsymbol{\omega})\sim\mathcal{D}}\Big[\nabla_a \; \boldsymbol{\omega}^{\mathsf{T}}\mathbf{Q}(s,a,\boldsymbol{\omega};\theta_1)|_{a=\pi(s,\boldsymbol{\omega};\phi)}\nabla_{\phi}\pi(s,\boldsymbol{\omega};\phi)\Big] + \alpha \cdot \mathbb{E}_{(s,a,\boldsymbol{\omega})\sim\mathcal{D}}\Big[\nabla_a \; g(\boldsymbol{\omega}_p, \mathbf{Q}(s,a,\boldsymbol{\omega};\theta_1)|_{a=\pi(s,\boldsymbol{\omega};\phi)}\nabla_{\phi}\pi(s,\boldsymbol{\omega};\phi)\Big] \tag{B.3}$$

where α denotes the loss coefficient to scale up the directional angle term to match the $\boldsymbol{\omega}^{\mathsf{T}}\mathbf{Q}$ term. Here, we also soft update the target networks at every time step k. Similar to MO-DDQN-HER, each child process runs for N time steps. Hence, in total, we collect $N \times C_p$ transitions during training.

Algorithm 11: Preference-Driven MO-TD3-HER

Input: Minibatch size N_m, Number of time steps N, Discount factor γ, Target network update coefficient τ, Multi-dimensional interpolator $I(\boldsymbol{\omega})$, Policy update delay p_{delay}, Standard deviation for Gaussian exploration noise added to the policy σ, Standard deviation for smoothing noise added to target policy σ', Limit for absolute value of target policy smoothing noise c.

Initialize: Replay buffer $\mathcal{D}$, Critic networks $\mathbf{Q}_{\theta_1}$,$\mathbf{Q}_{\theta_2}$ and actor network π_ϕ with parameters θ_1, θ_2, and ϕ, Target networks $\mathbf{Q}_{\theta'_1} \leftarrow \mathbf{Q}_{\theta_1}$, $\mathbf{Q}_{\theta'_2} \leftarrow \mathbf{Q}_{\theta_2}$, $\pi_{\phi'} \leftarrow \pi_\phi$,

for $n = 0$: N **do**

`// Child Process`

Initialize $t = 0$ and $done = False$. Reset the environment to randomly initialized state s_0.

Sample a preference vector $\boldsymbol{\omega}$ from the subspace $\tilde{\Omega}$

while $done = False$ **do**

Observe state s_t and select an action with exploration noise $a_t \sim \pi(s_t, \boldsymbol{\omega}; \phi) + \epsilon : \epsilon \sim \mathcal{N}(0, \sigma)$

Observe reward $\mathbf{r}$, s', and $done$. Transfer $(s_t, a_t, \mathbf{r}_t, s', \boldsymbol{\omega}, done)$ to main process.

`// Main Process`

Store the transition $(s_t, a_t, \mathbf{r}_t, s', \boldsymbol{\omega}, done)$ obtained from every child process in $\mathcal{D}$

Sample N_ω preferences $\boldsymbol{\omega}'$

for $j = 1$: N_ω **do**

Store transition $(s_t, a_t, \mathbf{r}_t, s', \boldsymbol{\omega}'_j, done)$ in $\mathcal{D}$

Sample N_m transitions from $\mathcal{D}$.

$\tilde{a} \leftarrow \pi(s', \boldsymbol{\omega}; \phi') + \epsilon : \epsilon \sim clip(\mathcal{N}(0, \sigma'), -c, c)$

$\mathbf{y} \leftarrow \mathbf{r} + \gamma \arg_{\mathbf{Q}} \min_{i=1,2} \boldsymbol{\omega}^T \mathbf{Q}(s', \tilde{a}, \boldsymbol{\omega}; \theta'_i)$

$\boldsymbol{\omega}_p \leftarrow I(\boldsymbol{\omega})$

$g(\boldsymbol{\omega}_p, \mathbf{Q}(s, a, \boldsymbol{\omega}; \theta_i)) \leftarrow \cos^{-1}\left(\frac{\boldsymbol{\omega}_p^T \mathbf{Q}(s,a,\boldsymbol{\omega};\theta_i)}{||\boldsymbol{\omega}_p|| \, ||\mathbf{Q}(s,a,\boldsymbol{\omega};\theta_i)||}\right)$

$L_{critic_k}(\theta_i) = \mathbb{E}_{(s,a,\mathbf{r},s',\boldsymbol{\omega})\sim\mathcal{D}}\left[\left(\mathbf{y} - \mathbf{Q}(s, a, \boldsymbol{\omega}; \theta_i)\right)^2\right] + \mathbb{E}_{(s,a,\boldsymbol{\omega})\sim\mathcal{D}}\left[g(\boldsymbol{\omega}_p, \mathbf{Q}(s, a, \boldsymbol{\omega}; \theta_i))\right]$

Update θ_{i_k} by applying SGD to $L_{critic_k}(\theta_i)$.

if $n \bmod p_{delay}$ **then**

$\nabla_\phi L_{actor_k}(\phi) = \mathbb{E}_{(s,a,\mathbf{r},s',\boldsymbol{\omega})\sim\mathcal{D}}\left[\nabla_a \, \boldsymbol{\omega}^T \mathbf{Q}(s, a, \boldsymbol{\omega}; \theta_1)|_{a=\pi(s,\boldsymbol{\omega};\phi)} \nabla_\phi \pi(s, \boldsymbol{\omega}; \phi)\right] +$ $\alpha \cdot \mathbb{E}_{(s,a,\boldsymbol{\omega})\sim\mathcal{D}}\left[\nabla_a \, g(\boldsymbol{\omega}_p, \mathbf{Q}(s, a, \boldsymbol{\omega}; \theta_1)|_{a=\pi(s,\boldsymbol{\omega};\phi)} \nabla_\phi \pi(s, \boldsymbol{\omega}; \phi)\right]$

Update target critics parameters $\theta'_{i_k} \leftarrow \tau\theta_{i_k} + (1 - \tau)\theta'_{i_k}$

Update target actor parameters $\phi'_k \leftarrow \tau\phi_k + (1 - \tau)\phi'_k$

B.2 Benchmarks

We first evaluate PD-MORL's performance on two commonly used discrete MORL benchmarks: Deep Sea Treasure and Fruit Tree Navigation. For these benchmarks, we make use of the codebase provided by [1]. We further evaluate PD-MORL on multi-objective continuous control tasks such as MO-Walker2d-v2, MO-HalfCheetah-v2, MO-Ant-v2, MO-Swimmer-v2, and MO-Hopper-v2. These benchmarks are presented by [4] and are licensed under the terms of the MIT license. The details for all benchmarks are provided below:

Deep Sea Treasure (DST): A well-studied MORL benchmark[23, 29] where an agent is a submarine trying to collect treasures in a 10×11 grid-world. The treasure values increase as their distance from the starting point $s_0 = (0, 0)$ increase. The submarine has two objectives: the time penalty and the treasure value. The actions are navigation in four directions and are discrete. The reward is a two-element vector. The first element shows the treasure value, and the second element shows the time penalty.

Fruit Tree Navigation (FTN): A recent MORL benchmark presented by Yang et al [1]. It is a binary tree of depth d with randomly assigned reward $\mathbf{r} \in \mathbb{R}^6$ on the leaf nodes. These rewards show the amounts of six different nutrition facts of the fruits on the tree: $\{Protein, Carbs, Fats, Vitamins, Minerals, Water\}$. The goal of the agent is to find a path on the tree to collect the fruit that maximizes the nutrition facts for a given preference.

MO-Walker2d-v2: The state space and action space is defined as $\mathcal{S} \subseteq \mathbb{R}^{17}, \mathcal{A} \subseteq \mathbb{R}^6$. The agent is a two-dimensional two-legged figure. It has two objectives to consider: forward speed and energy efficiency. The goal is to tune the amount of torque applied on the hinges for a given preference $\boldsymbol{\omega}$ while moving in the forward direction.

MO-HalfCheetah-v2: The state space and action space is defined as $\mathcal{S} \subseteq \mathbb{R}^{17}, \mathcal{A} \subseteq \mathbb{R}^6$. The agent is a two-dimensional robot that resembles a cheetah. It has two objectives to consider: forward speed and energy efficiency. The goal is to tune the amount of torque applied on the joints for a given preference $\boldsymbol{\omega}$ while running

in the forward direction.

MO-Ant-v2: The state space and action space is defined as $\mathcal{S} \subseteq \mathbb{R}^{27}, \mathcal{A} \subseteq \mathbb{R}^{8}$. The agent is a 3D robot that resembles an ant. It has two objectives to consider: x-axis speed and y-axis speed. The goal is to tune the amount of torque applied on the hinges connecting the legs and the torso for a given preference $\boldsymbol{\omega}$.

MO-Swimmer-v2: The state space and action space is defined as $\mathcal{S} \subseteq \mathbb{R}^{8}, \mathcal{A} \subseteq \mathbb{R}^{2}$. The agent is a two-dimensional robot. It has two objectives to consider: forward speed and energy efficiency. The goal is to tune the amount of torque applied on the rotors for a given preference $\boldsymbol{\omega}$ while swimming in the forward direction inside a two-dimensional pool.

MO-Hopper-v2: The state space and action space is defined as $\mathcal{S} \subseteq \mathbb{R}^{11}, \mathcal{A} \subseteq \mathbb{R}^{3}$. The agent is a two-dimensional one-legged figure. It has two objectives to consider: forward speed and jumping height. The goal is to tune the amount of torque applied on the hinges for a given preference $\boldsymbol{\omega}$ while moving in the forward direction by making hops.

B.3 Training Details

We run all our experiments on a local server including Intel Xeon Gold 6242R. We do not use any GPU in our implementation. For Deep Sea Treasure and Fruit Tree Navigation benchmarks, we employ the proposed MO-DDQN-HER algorithm. The network here takes state s and preference $\boldsymbol{\omega}$ as inputs and outputs $|\mathcal{A}| \times L$ Q-values. The number of hidden layers and hidden neurons among other hyperparameters for MO-DDQN-HER are given in Table B.1.

For continuous control benchmarks, we employ the proposed MO-TD3-HER algorithm. The critic network takes state s, preference $\boldsymbol{\omega}$, and action a as input and outputs L Q-values for each objective. The actor network takes state s and preference $\boldsymbol{\omega}$ as inputs and outputs $|\mathcal{A}|$ actions. The number of hidden layers and hidden neurons among other hyperparameters for MO-TD3-HER for each continuous benchmark are given in Table B.2. The hyperparameters are the same

for all benchmarks except the policy update delay for MO-Hopper-v2, which is set to 20.

As it is elaborated in Section B.1.3, we first obtain key solutions for each of the preferences in the key preference set to fit a multi-dimensional interpolator. Since we already know the Pareto front solutions for discrete benchmarks, we directly use key solutions in the Pareto front to fit this interpolator. For continuous control benchmarks, we train an agent utilizing the multi-objective version of TD3 with a fixed key preference vector and without using HER. The number of hidden layers and hidden neurons among other hyperparameters for this approach on each continuous benchmark are given in Table B.3.

Table B.1: Hyperparameters for MO-DDQN-HER

	Deep Sea Treasure	**Fruit Tree Navigation**
Total number of steps (N)	1×10^5	1×10^5
Minibatch size (N_m)	32	32
Discount factor (γ)	0.99	0.99
Soft update coefficient (τ)	0.005	0.005
Buffer size	1×10^4	1×10^4
Number of child processes (C_p)	10	10
Number of preferences sampled for HER (N_ω)	3	3
Learning rate	3×10^{-4}	3×10^{-4}
Number of hidden layers	3	3
Number of hidden neurons	256	512

Table B.2: Hyperparameters for MO-TD3-HER

	MO-Walker2d-v2	**MO-HalfCheetah-v2**	**MO-Ant-v2**	**MO-Swimmer-v2**	**MO-Hopper-v2**
Total number of steps	1×10^6	1×10^6	1×10^6	1×10^6	1×10^6
Minibatch size	256	256	256	256	256
Discount factor	0.995	0.995	0.995	0.995	0.995
Soft update coefficient	0.005	0.005	0.005	0.005	0.005
Buffer size	2×10^6	2×10^6	2×10^6	2×10^6	2×10^6
Number of child processes	10	10	10	10	10
Number of preferences sampled for HER	3	3	3	3	3
Learning rate-Critic	3×10^{-4}	3×10^{-4}	3×10^{-4}	3×10^{-4}	3×10^{-4}
Number of hidden layers - Critic	1	1	1	1	1
Number of hidden neurons - Critic	400	400	400	400	400
Learning rate-Actor	3×10^{-4}	3×10^{-4}	3×10^{-4}	3×10^{-4}	3×10^{-4}
Number of hidden layers - Actor	1	1	1	1	1
Number of hidden neurons - Actor	400	400	400	400	400
Policy update delay	10	10	10	10	20
Exploration noise std.	0.1	0.1	0.1	0.1	0.1
Target policy smoothing noise std.	0.2	0.2	0.2	0.2	0.2
Noise clipping limit	0.5	0.5	0.5	0.5	0.5
Loss coefficient	10	10	10	10	10

Table B.3: Hyperparameters for MO-TD3 algorithm to obtain key solutions

	MO-Walker2d-v2	MO-HalfCheetah-v2	MO-Ant-v2	MO-Swimmer-v2	MO-Hopper-v2
Total number of steps	2×10^6	2×10^6	1×10^6	1×10^6	1×10^6
Minibatch size	100	100	100	100	100
Discount factor	0.99	0.99	0.99	0.99	0.99
Soft update coefficient	0.005	0.005	0.005	0.005	0.005
Buffer size	5×10^5	5×10^5	5×10^5	1×10^6	1×10^6
Learning rate-Critic	3×10^{-4}	3×10^{-4}	3×10^{-4}	3×10^{-4}	3×10^{-4}
Number of hidden layers - Critic	1	1	1	1	1
Number of hidden neurons - Critic	400	400	400	400	400
Learning rate-Actor	3×10^{-4}	3×10^{-4}	3×10^{-4}	3×10^{-4}	3×10^{-4}
Number of hidden layers - Actor	1	1	1	1	1
Number of hidden neurons - Actor	400	400	400	400	400
Policy update delay	2	2	2	5	10
Exploration noise std.	0.1	0.1	0.1	0.1	0.1
Target policy smoothing noise std.	0.2	0.2	0.2	0.2	0.2
Noise clipping limit	0.5	0.5	0.5	0.5	0.5

B.4 Additional Experimental Results

Since the main objective of this work is to obtain a single universal network that covers the entire preference space, we first obtain a representative set of preference vectors. For Deep Sea Treasure benchmark, we obtain a preference vector set with a step size of 0.01 $(\{0,1\},\{0.01,0.99\},\ldots,\{0.99,0.01\},\{1,0\})$ Similarly, we obtain preference vector sets with a step size of 0.1 and 0.001 for Fruit Tree Navigation and continuous control benchmarks, respectively.

Since the Envelope algorithm [1] is the superior and more recent approach that learns a unified policy, we first compare PD-MORL with this algorithm using discrete MORL benchmarks. For the Envelope algorithm, we use the authors' codebase with their default hyperparameters mentioned in the paper since we believe that they are already optimized for these two benchmarks. In addition to hypervolume and sparsity metrics that we provide in Section 2.5.1, we also compare PD-MORL with the Envelope algorithm using the coverage ration F1 (CRF1) metric. This metric, which is coined by (**author?**) [1], evaluates the agent's ability to recover optimal solutions in the Pareto front. It is assumed that we have prior knowledge of the optimal solutions for these benchmarks. Let B be the set of solutions obtained by the agent for various preferences, and let P be the Pareto front set of solutions. The intersection between B and P with a tolerance of ϵ is defined as $B \cap P := b \in B | \exists\, p \in P : \|b - p\|_1 / \|p\|_1 \leqslant \epsilon$. The precision in this context

is defined as $Precision = \frac{|B \cap P|}{|B|}$ and recall is defined as $Recall = \frac{|B \cap P|}{|P|}$. Then CRF1 is computed as $CRF1 = 2\frac{Precision] \times Recall}{Precision + Recall}$. Table B.4 summarizes the comparison of PD-MORL with the Envelope algorithm in terms of CRF1, hypervolume, and sparsity metrics on discrete MORL benchmarks. We emphasize that our evaluation process is more extensive than the work proposed by (**author?**) [1]. In their implementation, they report an average of 2000 evaluation episodes where a random preference is sampled uniformly in each episode. In contrast, evaluation of PD-MORL sweeps the entire preference space. The hypervolume and sparsity values for both approaches are obtained through PD-MORL's evaluation process and are discussed in Section 2.5.1. In this table, we provide the CRF1 values reported in [1] for the Envelope algorithm. PD-MORL achieves larger or the same CRF1 values compared to the Envelope algorithm for all benchmarks. Specifically, it achieves up to 12% higher CRF1 for the Fruit Tree Navigation task with a depth of $d = 7$. Additionally, a comparison with [3] is given in Table B.4 which shows the superiority of both the Envelope algorithm and PD-MORL over the proposed algorithm by (**author?**) [3] which also learns a unified policy.

Table B.4: Comparison of our approach with prior works [1, 3] using discrete MORL benchmarks in terms CRF1, hypervolume, and sparsity metrics.* are the reported values in [1].

	Deep Sea Treasure			Fruit Tree Navigation (d=5)			Fruit Tree Navigation (d=6)			Fruit Tree Navigation (d=7)		
	CRF1	**Hypervolume**	**Sparsity**	**CRF1**	**Hypervolume**	**Sparsity**	**CRF1**	**Hypervolume**	**Sparsity**	**CRF1**	**Hypervolume**	**Sparsity**
Envelope [1]	0.994*	227.89	2.62	1	6920.58	N/A	0.995*	8427.51	N/A	0.819*	6395.27	N/A
CN+DER [3]	0.989*	–	–	1	–	N/A	0.9258*	–	N/A	0.6719*	–	N/A
Ours	1	241.73	1.14	1	6920.58	N/A	1	9299.15	N/A	0.92	11419.58	N/A

For the continuous control benchmarks, we provide additional plots on the Pareto front and hypervolume progression. Figure B.1(a)-(e) shows the progression of the Pareto front solutions during training. We plot the Pareto front solutions at early stages, mid stages, and late stages of the training process. As the training progresses, the Pareto front expands to a broader and denser curve. This behavior is supported by Figure B.2(a)-(e), which shows the progression of the hypervolume for all benchmarks. This figure shows that PD-MORL effectively pushes the hypervolume by discovering new Pareto solutions with the help of efficient and robust

exploration. For the MO-HalfCheetah-v2 problem, a broad and dense Pareto front is already achieved at the early stages of the training. Therefore, we do not observe the progression of the Pareto front and hypervolume for this specific benchmark.

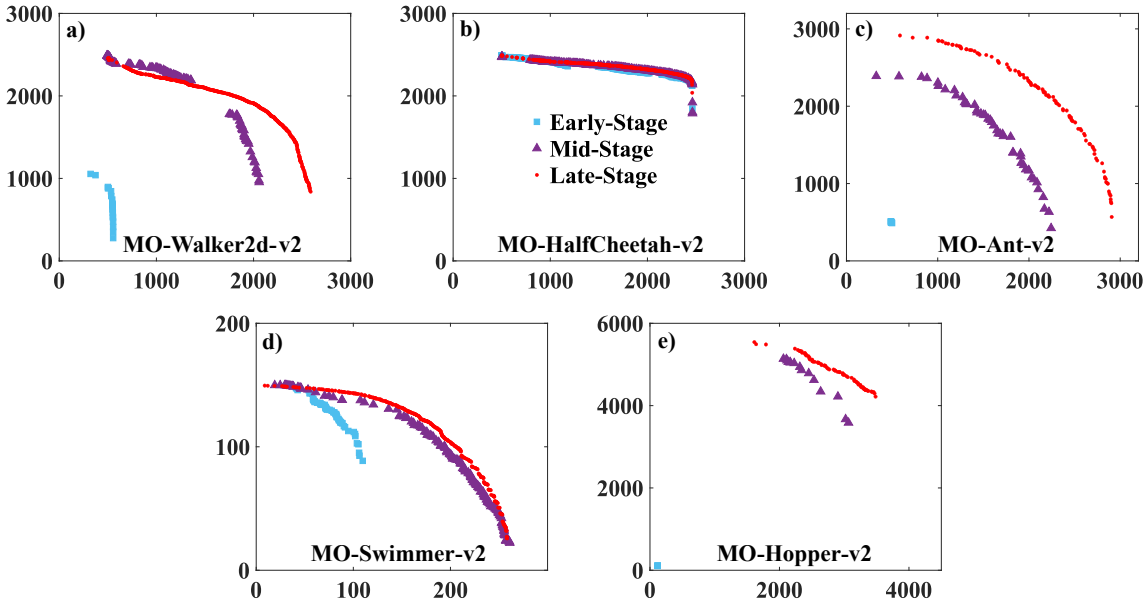


Figure B.1: Pareto front progression for (a) MO-Walker-v2, (b) MO-HalfCheetah-v2, (c) MO-Ant-v2, (d) MO-Swimmer-v2, (e) MO-Hopper-v2.

We also report standard deviations for all benchmarks. As explained in the main manuscript, since META [40] and PG-MORL [4] report the average of six runs, we also ran each benchmark six times with PD-MORL. Table B.5 reports average metrics as well as standard deviations obtained from these six runs. The standard deviations obtained from six runs are not reported in the prior work. This table shows that the individual differences of different runs are mostly three orders of magnitudes less than the average of these runs. This also suggests that PD-MORL achieves a generalizable network independent of starting state of the agent.

Additionally, we conduct an ablation study to show the effects of the proposed angle term in the loss function and the proposed parallel exploration in our algorithm on continuous control benchmarks. Table B.6 reports average metrics and standard deviations obtained from six runs. Since we fit a multi-dimensional

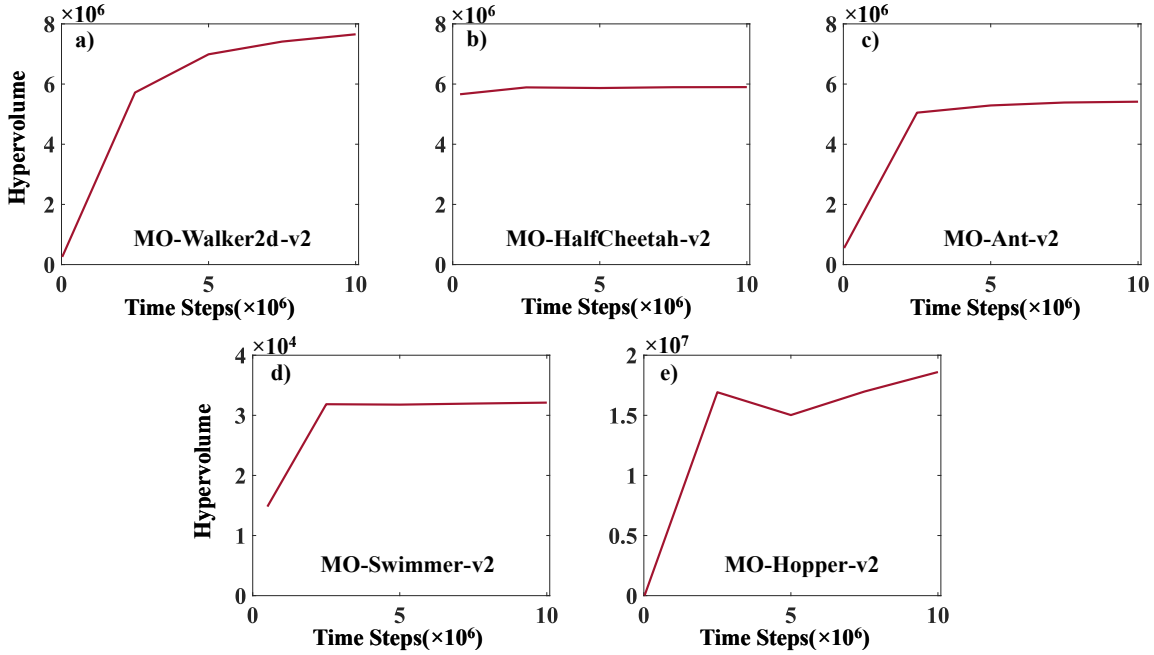


Figure B.2: Hypervolume progression for (a) MO-Walker-v2, (b) MO-HalfCheetah-v2, (c) MO-Ant-v2, (d) MO-Swimmer-v2, (e) MO-Hopper-v2.

interpolator using solutions with key preferences, the interpolator may introduce a bias at the beginning of the training depending on how representative these key solutions are. Hence, this bias may have a negative effect on the training. For tasks with a broad and more convex Pareto front, the probability of this bias increases as we choose key preferences from corner cases.

Therefore, by removing the angle term in the loss function and thus, removing the bias, the performance of our algorithm slightly degrades due to the convex and broad nature of the Pareto front in MO-Walker2d-v2 and MO-Ant-v2 problems. However, removing the angle term has a negative effect for MO-HalfCheetah-v2 and MO-Hopper-v2, where the Pareto front is dense. Figure B.3 shows the obtained Pareto front with and without the directional angle term in the loss function. The solutions obtained without the directional angle term are sparse. Specifically, for MO-HalfCheetah-v2, Table B.6 shows that both the hypervolume and the sparsity metrics are negatively affected. For MO-Hopper-v2, although

Table B.5: Performance comparison of the proposed approach and state-of-the-art algorithms on the continuous control benchmarks in terms of hypervolume and sparsity metrics. Reference point for hypervolume calculation is set to $(0,0)$ point. HV*: Hypervolume

		PG-MORL [4]	META [40]	**PD-MORL (Ours)**
MO-Walker2d-v2	HV*	4.82×10^6	2.10×10^6	$\mathbf{5.41 \pm 0.004 \times 10^6}$
	Sparsity	0.04×10^4	2.10×10^4	$\mathbf{0.03 \pm 0.005 \times 10^4}$
MO-HalfCheetah-v2	HV*	5.77×10^6	5.18×10^6	$\mathbf{5.89 \pm 0.002 \times 10^6}$
	Sparsity	$\mathbf{0.44 \times 10^3}$	2.13×10^3	$0.49 \pm 0.041 \times 10^3$
MO-Ant-v2	HV*	6.35×10^6	2.40×10^4	$\mathbf{7.48 \pm 0.019 \times 10^6}$
	Sparsity	$\mathbf{0.37 \times 10^4}$	1.56×10^4	$0.78 \pm 0.2 \times 10^4$
MO-Swimmer-v2	HV*	2.57×10^4	1.23×10^4	$\mathbf{3.21 \pm 0.001 \times 10^4}$
	Sparsity	9.9	24.4	$\mathbf{5.7 \pm 0.7}$
MO-Hopper-v2	HV*	$\mathbf{2.02 \times 10^7}$	1.25×10^7	$1.88 \pm 0.005 \times 10^7$
	Sparsity	0.5×10^4	4.84×10^4	$\mathbf{0.3 \pm 0.09 \times 10^4}$

the hypervolume metric seems to be increased, the change in the sparsity metric suggests that the algorithm cannot find a dense Pareto front. This, in fact, supports that the hypervolume alone is insufficient to assess the quality of the Pareto front. We further investigate the effects of using parallel exploration. To this end, we collect transitions using a single preference space instead of dividing them into sub-spaces. Parallel exploration guarantees that obtained transitions are not biased towards a preference sub-space. Hence, it increases the sample efficiency of the algorithm. Table B.6 shows that removing the parallel exploration significantly reduces the performance of the algorithm. For all tasks, the hypervolume metric decreases, and the sparsity metric increases. We also observe that both PG-MORL and PD-MORL with proposed novel aspects are superior to PD-MORL without the novel aspects. This observation is crucial since the prior work uses PPO, whereas PD-MORL uses TD3 in their algorithm pipeline. Using this observation, we conclude that the superiority of PD-MORL is due to the proposed novel aspects.

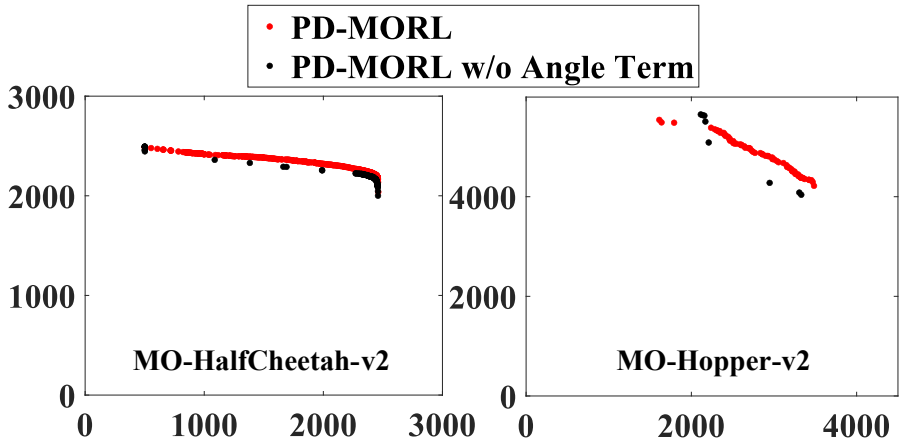


Figure B.3: Pareto front with and without the directional angle term in the loss function for MO-HalfCheetah-v2 and MO-Hopper-v2 problems. In the latter case, the solutions are sparse and represented by few points.

Table B.6: Ablation study of the proposed approach with and without proposed terms and improvements on continuous control benchmarks in terms of hypervolume and sparsity metrics. Reference point for hypervolume calculation is set to (0,0) point. HV*: Hypervolume

		PG-MORL [4]	PD-MORL w/o Angle Term	PD-MORL w/o Parallel Exploration	PD-MORL
MO-Walker2d-v2	HV*	4.82×10^6	$\mathbf{5.51 \pm 0.004 \times 10^6}$	$2.81 \pm 0.009 \times 10^6$	$5.41 \pm 0.004 \times 10^6$
	Sparsity	0.04×10^4	$\mathbf{0.02 \pm 0.004 \times 10^4}$	$0.22 \pm 0.026 \times 10^4$	$0.03 \pm 0.005 \times 10^4$
MO-HalfCheetah-v2	HV*	5.77×10^6	$5.61 \pm 0.003 \times 10^6$	$5.84 \pm 0.001 \times 10^6$	$\mathbf{5.89 \pm 0.002 \times 10^6}$
	Sparsity	$\mathbf{0.44 \times 10^3}$	$49.73 \pm 3.917 \times 10^3$	$1.218 \pm 0.172 \times 10^3$	$0.49 \pm 0.041 \times 10^3$
MO-Ant-v2	HV*	6.35×10^6	$\mathbf{9.12 \pm 0.017 \times 10^6}$	$4.98 \pm 0.237 \times 10^6$	$7.48 \pm 0.019 \times 10^6$
	Sparsity	$\mathbf{0.37 \times 10^4}$	$\mathbf{0.58 \pm 0.09 \times 10^4}$	$0.91 \pm 0.25 \times 10^4$	$0.78 \pm 0.2 \times 10^4$
MO-Swimmer-v2	HV*	2.57×10^4	$2.78 \pm 0.001 \times 10^4$	$3.21 \pm 0.001 \times 10^4$	$\mathbf{3.21 \pm 0.001 \times 10^4}$
	Sparsity	9.9	6.3 ± 0.9	9.4 ± 1.1	$\mathbf{5.7 \pm 0.7}$
MO-Hopper-v2	HV*	$\mathbf{2.02 \times 10^7}$	$1.92 \pm 0.023 \times 10^7$	$0.63 \pm 0.005 \times 10^7$	$1.88 \pm 0.005 \times 10^7$
	Sparsity	0.5×10^4	$6.1 \pm 2.30 \times 10^4$	$20.53 \pm 33.20 \times 10^4$	$\mathbf{0.3 \pm 0.09 \times 10^4}$

C APPENDIX C: PARAMETER SWEEP FOR THE HEURISTIC USED IN CHAPTER 5

C.1 Parameter sweep for the heuristic

Table C.1: All 285 configurations for c_1, c_2, c_3.

c_3	c_2	c_1	c_3	c_2	c_1	c_3	c_2	c_1	c_3	c_2	c_1	c_3	c_2	c_1	c_3	c_2	c_1	c_3	c_2	c_1	c_3	c_2	c_1
15	15	15	30	15	15	33	27	27	36	33	15	39	30	30	42	27	15	42	42	15	45	33	27	45	45	30
18	15	15	30	18	15	33	30	15	36	33	18	39	33	15	42	27	18	42	42	18	45	33	30	45	45	33
18	18	15	30	18	18	33	30	18	36	33	21	39	33	18	42	27	21	42	42	21	45	33	33	45	45	36
18	18	18	30	21	15	33	30	21	36	33	24	39	33	21	42	27	24	42	42	24	45	36	15	45	45	39
21	15	15	30	21	18	33	30	24	36	33	27	39	33	24	42	27	27	42	42	27	45	36	18	45	45	42
21	18	15	30	21	21	33	30	27	36	33	30	39	33	27	42	30	15	42	42	30	45	36	21	45	45	45
21	18	18	30	24	15	33	30	30	36	33	33	39	33	30	42	30	18	42	42	33	45	36	24			
21	21	15	30	24	18	33	33	15	36	36	15	39	33	33	42	30	21	42	42	36	45	36	27			
21	21	18	30	24	21	33	33	18	36	36	18	39	36	15	42	30	24	42	42	39	45	36	30			
21	21	21	30	24	24	33	33	21	36	36	21	39	36	18	42	30	27	42	42	42	45	36	33			
24	15	15	30	27	15	33	33	24	36	36	24	39	36	21	42	30	30	45	15	15	45	36	36			
24	18	15	30	27	18	33	33	27	36	36	27	39	36	24	42	33	15	45	18	15	45	39	15			
24	18	18	30	27	21	33	33	30	36	36	30	39	36	27	42	33	18	45	18	18	45	39	18			
24	21	15	30	27	24	33	33	33	36	36	33	39	36	30	42	33	21	45	21	15	45	39	21			
24	21	18	30	27	27	36	15	15	36	36	36	39	36	33	42	33	24	45	21	18	45	39	24			
24	21	21	30	30	15	36	18	15	39	15	15	39	36	36	42	33	27	45	21	21	45	39	27			
24	24	15	30	30	18	36	18	18	39	18	15	39	39	15	42	33	30	45	24	15	45	39	30			
24	24	18	30	30	21	36	21	15	39	18	18	39	39	18	42	33	33	45	24	18	45	39	33			
24	24	21	30	30	24	36	21	18	39	21	15	39	39	21	42	36	15	45	24	21	45	39	36			
24	24	24	30	30	27	36	21	21	39	21	18	39	39	24	42	36	18	45	24	24	45	39	39			
27	15	15	30	30	30	36	24	15	39	21	21	39	39	27	42	36	21	45	27	15	45	42	15			
27	18	15	33	15	15	36	24	18	39	24	15	39	39	30	42	36	24	45	27	18	45	42	18			
27	18	18	33	18	15	36	24	21	39	24	18	39	39	33	42	36	27	45	27	21	45	42	21			
27	21	15	33	18	18	36	24	24	39	24	21	39	39	36	42	36	30	45	27	24	45	42	24			
27	21	18	33	21	15	36	27	15	39	24	24	39	39	39	42	36	33	45	27	27	45	42	27			
27	21	21	33	21	18	36	27	18	39	27	15	42	15	15	42	36	36	45	30	15	45	42	30			
27	24	15	33	21	21	36	27	21	39	27	18	42	18	15	42	39	15	45	30	18	45	42	33			
27	24	18	33	24	15	36	27	24	39	27	21	42	18	18	42	39	18	45	30	21	45	42	36			
27	24	21	33	24	18	36	27	27	39	27	24	42	21	15	42	39	21	45	30	24	45	42	39			
27	24	24	33	24	21	36	30	15	39	27	27	42	21	18	42	39	24	45	30	27	45	42	42			
27	27	15	33	24	24	36	30	18	39	30	15	42	21	21	42	39	27	45	30	30	45	45	15			
27	27	18	33	27	15	36	30	21	39	30	18	42	24	15	42	39	30	45	33	15	45	45	18			
27	27	21	33	27	18	36	30	24	39	30	21	42	24	18	42	39	33	45	33	18	45	45	21			
27	27	24	33	27	21	36	30	27	39	30	24	42	24	21	42	39	36	45	33	21	45	45	24			
27	27	27	33	27	24	36	30	30	39	30	27	42	24	24	42	39	39	45	33	24	45	45	27			

D APPENDIX D: ADDITIONAL EXPERIMENTAL RESULTS - DTRL

In this section, we assess the generalizability of DTRL by evaluating it using a different DSSoC configuration. Specifically, we divide the number of accelerator cores by two to demonstrate DTRL's performance across different configurations. This configuration comprises thirteen PEs classified into five clusters based on their functionalities. These clusters comprise four LITTLE Arm A57 cores, four big Arm Cortex-A53 cores, and fixed-function accelerators, which include one matrix multiplication (MM) core and two fast Fourier transform (FFT) cores, and one Viterbi decoding core. Besides the difference in the DSSoC configuration, the experimental setup remains the same for this evaluation.

Fig. D.1(a) compares the average frame execution time of DTRL with baseline and state-of-the-art schedulers. The DTRL policy in Fig. D.1(a) uses a preference vector of $\{1, 0\}$, whereas the results for Scalarized-MOPPO are obtained using the policy that is separately trained for the same preference vector. DTRL achieves an execution time speedup of 1.25×, 1.05×, 1.2×, 1.8×, and 8.5× than ILS, Scalarized-

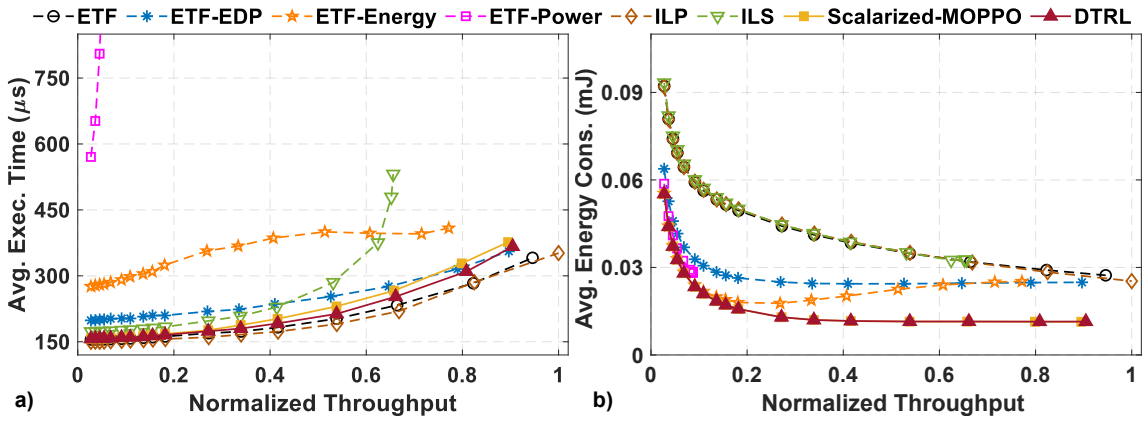


Figure D.1: Comparison of a) average frame execution time and b) average energy consumption between ETF [5], ETF-EDP, ETF-Energy, ETF-Power, ILP solution, ILS [6], Scalarized-MOPPO ([7]-Adapted) and DTRL to schedule a workload comprising six streaming applications. The x-axis in Fig. 6(a) and 6(b) is normalized to the throughput achieved by the ILP solution.

MOPPO, ETF-EDP, ETF-Energy, and ETF-Power, respectively. Although DTRL is trained with multiple objectives, it still achieves an average execution time within 5% and 9% of ETF and ILP, respectively. We emphasize that the runtime overhead of ETF is 2× and 10× higher than that of DTRL, as evaluated in Section **??**. It is important to highlight that ETF, ILP, and ILS are designed to optimize a single specific objective, whereas Scalarized-MOPPO and DTRL are specifically trained to handle multiple objectives. *It is important to highlight that DTRL learns a Pareto front set of solutions for execution time and power consumption objectives using a single policy for various DSSoC configurations*.

DTRL's energy consumption is evaluated by using a preference vector of $\{0, 1\}$ to the global DDT policy, as shown in Fig. D.1(b). DTRL achieves 3.1×, 3.1×, 3.6×, 2×, 1.7×, and 1.06× lower energy consumption compared to ETF, ILP, ILS, ETF-EDP, ETF-Energy, and ETF-Power, respectively. It achieves very similar energy consumption values compared to Scalarized-MOPPO.

Fig. D.2(a)-(c) illustrates the average frame execution time versus energy efficiency curves for low, medium, and high throughput workloads using multiple preference vectors. The evaluation employs preference vectors (ω) separated by a

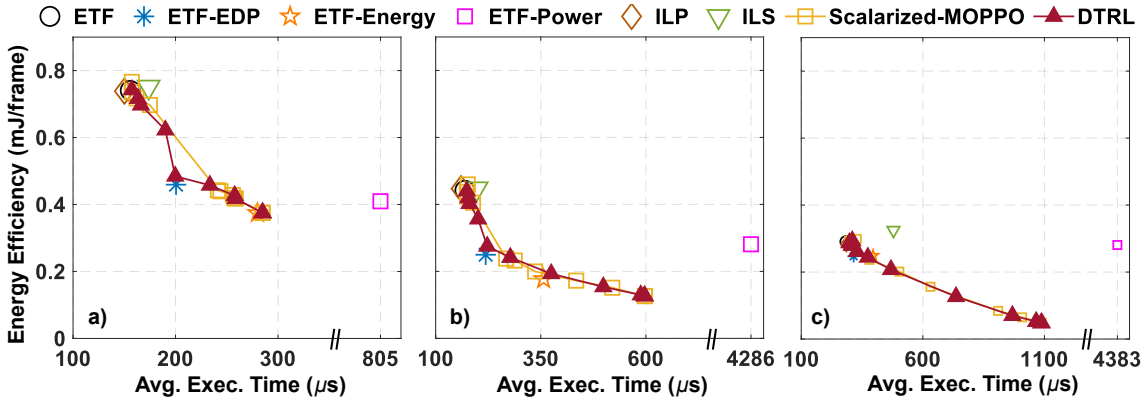


Figure D.2: Average frame execution time (µs) vs. Energy efficiency (mJ / frame) for a) low b) medium c) high target throughputs. Comparison between ETF [5], ETF-EDP, ETF-Energy, ETF-Power, ILP solution, ILS [6], Scalarized-MOPPO ([7]-Adapted), and DTRL are presented. DTRL achieves multiple near-optimal policies using various preferences.

step size of 0.1, $\boldsymbol{\omega} \in \{\{1,0\},\{0.9,0.1\},\ldots,\{0.1,0.9\},\{0,1\}\}$. Fig. D.2(a)-(c) also shows the energy efficiency (in milli-Joules per frame) of the baseline and state-of-the-art schedulers. It should be noted that ETF, ILP, and ILS represent a single point on the plot since they are designed for a single objective. However, ETF's variants correspond to different comparison points in the objective space. In contrast, DTRL covers the entire preference space and generates solutions that can be compared to ETF and ILP by utilizing a single global DDT policy. DTRL outperforms other schedulers and offers the flexibility to generate near-optimal scheduling decisions for any preference vector specified at runtime.

BIBLIOGRAPHY

[1] Runzhe Yang, Xingyuan Sun, and Karthik Narasimhan. A generalized algorithm for multi-objective reinforcement learning and policy adaptation. *Advances in Neural Information Processing Systems*, 32:14636–14647, 2019.

[2] Taoufik Bouguera, Jean-François Diouris, Jean-Jacques Chaillout, Randa Jaouadi, and Guillaume Andrieux. Energy consumption model for sensor nodes based on lora and lorawan. *Sensors*, 18(7):2104, 2018.

[3] Axel Abels, Diederik Roijers, Tom Lenaerts, Ann Nowé, and Denis Steckelmacher. Dynamic weights in multi-objective deep reinforcement learning. In *International Conference on Machine Learning*, pages 11–20. PMLR, 2019.

[4] Jie Xu, Yunsheng Tian, Pingchuan Ma, Daniela Rus, Shinjiro Sueda, and Wojciech Matusik. Prediction-guided multi-objective reinforcement learning for continuous robot control. In *International Conference on Machine Learning*, pages 10607–10616. PMLR, 2020.

[5] Jing-Jang Hwang, Yuan-Chieh Chow, Frank D Anger, and Chung-Yee Lee. Scheduling Precedence Graphs in Systems with Interprocessor Communication Times. *SIAM Journal on Computing*, 18(2):244–257, 1989.

[6] Anish Krishnakumar et al. Runtime Task Scheduling using Imitation Learning for Heterogeneous Many-core Systems. *IEEE Transactions on CAD of Integrated Circuits and Systems*, 39(11):4064–4077, 2020.

[7] Hossam Mossalam, Yannis M Assael, Diederik M Roijers, and Shimon Whiteson. Multi-objective deep reinforcement learning. *arXiv preprint arXiv:1610.02707*, 2016.

[8] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[9] Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Dębiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019.

[10] Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.

[11] Hai Nguyen and Hung La. Review of deep reinforcement learning for robot manipulation. In *2019 Third IEEE International Conference on Robotic Computing (IRC)*, pages 590–595. IEEE, 2019.

[12] Saurabh Gupta, Siddhant Bhambri, Karan Dhingra, Arun Balaji Buduru, and Ponnurangam Kumaraguru. Multi-objective reinforcement learning based approach for user-centric power optimization in smart home environments. In *2020 IEEE International Conference on Smart Data Services (SMDS)*, pages 89–96. IEEE, 2020.

[13] Liang Yu, Shuqi Qin, Meng Zhang, Chao Shen, Tao Jiang, and Xiaohong Guan. A review of deep reinforcement learning for smart building energy management. *IEEE Internet of Things Journal*, 2021.

[14] Szilárd Aradi. Survey of deep reinforcement learning for motion planning of autonomous vehicles. *IEEE Transactions on Intelligent Transportation Systems*, 23(2):740–759, 2020.

[15] Joris Dinneweth, Abderrahmane Boubezoul, René Mandiau, and Stéphane Espié. Multi-agent reinforcement learning for autonomous vehicles: A survey. *Autonomous Intelligent Systems*, 2(1):27, 2022.

[16] Antonio Coronato, Muddasar Naeem, Giuseppe De Pietro, and Giovanni Paragliola. Reinforcement learning for intelligent healthcare applications: A survey. *Artificial Intelligence in Medicine*, 109:101964, 2020.

[17] Arjun Reddy Kunduru. Machine learning in drug discovery: A comprehensive analysis of applications, challenges, and future directions. 2023.

[18] Ben Hambly, Renyuan Xu, and Huining Yang. Recent advances in reinforcement learning in finance. *Mathematical Finance*, 33(3):437–503, 2023.

[19] Xiao-Yang Liu, Hongyang Yang, Jiechao Gao, and Christina Dan Wang. Finrl: Deep reinforcement learning framework to automate trading in quantitative finance. In *Proceedings of the second ACM international conference on AI in finance*, pages 1–9, 2021.

[20] Rui Nian, Jinfeng Liu, and Biao Huang. A review on reinforcement learning: Introduction and applications in industrial process control. *Computers & Chemical Engineering*, 139:106886, 2020.

[21] Hao Zheng and Ahmed Louri. An energy-efficient network-on-chip design using reinforcement learning. In *Proceedings of the 56th Annual Design Automation Conference 2019*, pages 1–6, 2019.

[22] Azalia Mirhoseini, Anna Goldie, Mustafa Yazgan, Joe Jiang, Ebrahim Songhori, Shen Wang, Young-Joon Lee, Eric Johnson, Omkar Pathak, Sungmin Bae, et al. Chip placement with deep reinforcement learning. *arXiv preprint arXiv:2004.10746*, 2020.

[23] Conor F Hayes, Roxana Rădulescu, Eugenio Bargiacchi, Johan Källström, Matthew Macfarlane, Mathieu Reymond, Timothy Verstraeten, Luisa M Zintgraf, Richard Dazeley, Fredrik Heintz, et al. A practical guide to multi-objective reinforcement learning and planning. *Autonomous Agents and Multi-Agent Systems*, 36(1):1–59, 2022.

[24] Aviv Navon, Aviv Shamsian, Gal Chechik, and Ethan Fetaya. Learning the pareto front with hypernetworks. *arXiv preprint arXiv:2010.04104*, 2020.

[25] Toygun Basaklar, Suat Gumussoy, and Umit Y Ogras. Pd-morl: Preference-driven multi-objective reinforcement learning algorithm. *arXiv preprint arXiv:2208.07914*, 2022.

[26] Toygun Basaklar, Yigit Tuncel, Suat Gumussoy, and Umit Ogras. Gem-rl: Generalized energy management of wearable devices using reinforcement learning. In *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1–6. IEEE, 2023.

[27] Yigit Tuncel, Toygun Basaklar, Dina Carpenter-Graffy, and Umit Ogras. A self-sustained cps design for reliable wildfire monitoring. *ACM Transactions on Embedded Computing Systems*, 22(5s):1–23, 2023.

[28] Toygun Basaklar, A Alper Goksoy, Anish Krishnakumar, Suat Gumussoy, and Umit 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):1–22, 2023.

[29] Chunming Liu, Xin Xu, and Dewen Hu. Multiobjective reinforcement learning: A comprehensive overview. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 45(3):385–398, 2014.

[30] Abbas Abdolmaleki, Sandy Huang, Leonard Hasenclever, Michael Neunert, Francis Song, Martina Zambelli, Murilo Martins, Nicolas Heess, Raia Hadsell, and Martin Riedmiller. A distributional view on multi-objective policy optimization. In *International Conference on Machine Learning*, pages 11–22. PMLR, 2020.

[31] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. *Advances in neural information processing systems*, 30, 2017.

[32] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

[33] Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*, pages 1587–1596. PMLR, 2018.

[34] Diederik M Roijers, Peter Vamplew, Shimon Whiteson, and Richard Dazeley. A survey of multi-objective sequential decision-making. *Journal of Artificial Intelligence Research*, 48:67–113, 2013.

[35] Kristof Van Moffaert, Madalina M Drugan, and Ann Nowé. Scalarized multi-objective reinforcement learning: Novel design techniques. In *2013 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*, pages 191–199. IEEE, 2013.

[36] Diederik M Roijers, Shimon Whiteson, Frans A Oliehoek, et al. Linear support for multi-objective coordination graphs. In *AAMAS'14: Proceedings of the 2014 International Conference on Autonomous Agents & Multiagent Systems*, pages 1297–1304, 2014.

[37] Marcela Zuluaga, Andreas Krause, and Markus Püschel. ε-pal: an active learning approach to the multi-objective optimization problem. *The Journal of Machine Learning Research*, 17(1):3619–3650, 2016.

[38] Matteo Pirotta, Simone Parisi, and Marcello Restelli. Multi-objective reinforcement learning with continuous pareto frontier approximation. In *Twenty-ninth AAAI conference on artificial intelligence*, 2015.

[39] Simone Parisi, Matteo Pirotta, and Jan Peters. Manifold-based multi-objective policy search with sample reuse. *Neurocomputing*, 263:3–14, 2017.

[40] Xi Chen, Ali Ghadirzadeh, Mårten Björkman, and Patric Jensfelt. Meta-learning for multi-objective reinforcement learning. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 977–983. IEEE, 2019.

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

[42] Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pages 1587–1596. PMLR, 2018.

[43] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.

[44] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, pages 5026–5033. IEEE, 2012.

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

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

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

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

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

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

[51] Billy Pik Lik Lau, Sumudu Hasala Marakkalage, Yuren Zhou, Naveed Ul Hassan, Chau Yuen, Meng Zhang, and U-Xuan Tan. A survey of data fusion in smart city applications. *Information Fusion*, 52:357–374, 2019.

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

[53] Toygun Basaklar, Yigit Tuncel, Shruti Yadav Narayana, Suat Gumussoy, and Umit Y Ogras. Hypervector design for efficient hyperdimensional computing on edge devices. *arXiv preprint arXiv:2103.06709*, 2021.

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

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

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

[57] Yigit Tuncel, Ganapati Bhat, Jaehyun Park, and Umit Ogras. Eco: Enabling energy-neutral iot devices through runtime allocation of harvested energy. *IEEE Internet of Things Journal*, 2021.

[58] IBM ILOG Cplex. V12. 1: User's manual for cplex. *International Business Machines Corporation*, 46(53):157, 2009.

[59] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2023.

[60] Quentin Cappart, Thierry Moisan, Louis-Martin Rousseau, Isabeau Prémont-Schwarz, and Andre A Cire. Combining reinforcement learning and constraint programming for combinatorial optimization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 3677–3687, 2021.

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

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

[63] Dina Hussein, Ganapati Bhat, and Janardhan Rao Doppa. Adaptive energy management for self-sustainable wearables in mobile health. In *Proc. AAAI*, 2022.

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

[65] Nuzhat Yamin and Ganapati Bhat. Uncertainty-aware energy harvest prediction and management for iot devices. *ACM Transactions on Design Automation of Electronic Systems*, 28(5):1–33, 2023.

[66] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[67] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[68] DEKRA Testing and Certification. FCCID - 2AJ2X-WS40, 2024.

[69] Amazon Halo. Amazon Halo - Health & wellness band, 2024.

[70] DEKRA Testing and Certification. FCCID - Test Report USA Part 15.247, 15.209., 2024.

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

[72] Toygun Basaklar, Yigit Tuncel, Sizhe An, and Umit Ogras. Wearable devices and low-power design for smart health applications: challenges and opportunities. In *Intl. Symp. on Low Power Electronics and Design*, pages 1–1, 2021.

[73] G Sucharitha, Bodepu Tannmayee, and Kanagala Dwarakamai. Revolution in iot: Smart wearable technology. In *Internet of Things and Its Applications*, pages 407–425. 2022.

[74] Ganapati Bhat, Ranadeep Deb, and Umit Y Ogras. OpenHealth: Open source platform for wearable health monitoring. *IEEE Design & Test*, 2019.

[75] Nuzhat Yamin, Ganapati Bhat, and Janardhan Rao Doppa. Diet: a dynamic energy management approach for wearable health monitoring devices. In *Design, Automation & Test in Europe Conf. & Exhibition*, pages 1365–1370, 2022.

[76] Piotr Mirowski et al. Learning to navigate in cities without a map. In *Neural Information Processing Systems* (*NeurIPS*), 2018.

[77] Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In *Field and Service Robotics*, 2017.

[78] Richard Bellman. Dynamic programming. *Science*, 153(3731):34–37, 1966.

[79] Michael Grant and Stephen Boyd. CVX: Matlab software for disciplined convex programming, version 2.1. `http://cvxr.com/cvx`, March 2014.

[80] The European Commission. Data from the EFFIS database. Online; accessed March 2023.

[81] National Interagency Fire Center. Wildfires and Acres. https://www.nifc.gov/fire-information/statistics/wildfires, accessed Mar 2023.

[82] Vanessa Romo. PG&E Pleads Guilty On 2018 California Camp Fire: "Our Equipment Started That Fire". https://www.npr.org/2020/06/16/879008760/pg-e-pleads-guilty-on-2018-california-camp-fire-our-equipment-started-that-fir accessed Mar 2023.

[83] Fire Danger Forecast. Wildland Fire Potential Index (WFPI). https://www.usgs.gov/fire-danger-forecast/wildland-fire-potential-index-wfpi, accessed Mar 2023.

[84] Forest Service. National Fire Danger Rating System. https://www.fs.usda.gov/detail/scnfs/home/?cid=fseprd634957, accessed Mar 2023.

[85] W Matt Jolly, Patrick H Freeborn, Wesley G Page, and Bret W Butler. Severe fire danger index: A forecastable metric to inform firefighter and community wildfire risk management. *Fire*, 2(3):47, 2019.

[86] Ankita Mohapatra and Timothy Trinh. Early wildfire detection technologies in practice—a review. *Sustainability*, 14(19):12270, 2022.

[87] Yaokun Pang et al. Multilayered cylindrical triboelectric nanogenerator to harvest kinetic energy of tree branches for monitoring environment condition and forest fire. *Advanced functional materials*, 30(32):2003598, 2020.

[88] Chi Yuan, Zhixiang Liu, and Youmin Zhang. Uav-based forest fire detection and tracking using image processing techniques. In *IEEE International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 639–643, 2015.

[89] Huy Xuan Pham, Hung Manh La, David Feil-Seifer, and Matthew C Deans. A distributed control framework of multiple unmanned aerial vehicles for dynamic wildfire tracking. *IEEE Transactions on Systems, Man, and Cybernetics*, 50(4):1537–1548, 2018.

[90] Joseph W Gangestad, Darren W Rowen, and Brian S Hardy. Forest fires, sunglint, and a solar eclipse: Responsive remote sensing with aerocube-4. In *IEEE Geoscience and Remote Sensing Symposium*, pages 3622–3625, 2014.

[91] Muhammad Hasif bin Azami et al. Demonstration of wildfire detection using image classification onboard cubesat. In *IEEE International Geoscience and Remote Sensing Symposium IGARSS*, pages 5413–5416, 2021.

[92] Muhammad Hasif bin Azami, Necmi Cihan Orger, Victor Hugo Schulz, Takashi Oshiro, and Mengu Cho. Earth observation mission of a 6u cubesat with a 5-meter resolution for wildfire image classification using convolution neural network approach. *Remote Sensing*, 14(8):1874, 2022.

[93] Udaya Dampage, Lumini Bandaranayake, Ridma Wanasinghe, Kishanga Kottahachchi, and Bathiya Jayasanka. Forest fire detection system using wireless sensor networks and machine learning. *Scientific reports*, 12(1):46, 2022.

[94] Miguel G Cruz and Martin E Alexander. The 10% wind speed rule of thumb for estimating a wildfire's forward rate of spread in forests and shrublands. *Annals of Forest Science*, 76(2):1–11, 2019.

[95] Rachid Zagrouba and Amine Kardi. Comparative study of energy efficient routing techniques in wireless sensor networks. *Information*, 12(1):42, 2021.

[96] Jaewoong Kang, Jongmo Kim, Minhwan Kim, and Mye Sohn. Machine learning-based energy-saving framework for environmental states-adaptive wireless sensor network. *IEEE Access*, 8:69359–69367, 2020.

[97] Abdulaziz Alarifi and Amr Tolba. Optimizing the network energy of cloud assisted internet of things by using the adaptive neural learning approach in wireless sensor networks. *Computers in Industry*, 106:133–141, 2019.

[98] Kunal Shah and Mohan Kumar. Distributed independent reinforcement learning (dirl) approach to resource management in wireless sensor networks. In *IEEE International Conference on Mobile Adhoc and Sensor Systems*, pages 1–9, 2007.

[99] Gabriel Martins Dias, Maddalena Nurchis, and Boris Bellalta. Adapting sampling interval of sensor networks using on-line reinforcement learning. In *IEEE 3rd World Forum on Internet of Things (WF-IoT)*, pages 460–465, 2016.

[100] Francesco Fraternali, Bharathan Balaji, and Rajesh Gupta. Scaling configuration of energy harvesting sensors with reinforcement learning. In *Proceedings of the workshop on energy harvesting & energy-neutral sensing systems*, pages 7–13, 2018.

[101] Mohamed Maalej, Sofiane Cherif, and Hichem Besbes. Qos and energy aware cooperative routing protocol for wildfire monitoring wireless sensor networks. *The Scientific World Journal*, 2013.

[102] Toygun Basaklar, Yigit Tuncel, and Umit Y Ogras. tinyman: Lightweight energy manager using reinforcement learning for energy harvesting wearable iot devices. *arXiv preprint arXiv:2202.09297*, 2022.

[103] Yen Kheng Tan and Sanjib Kumar Panda. Self-autonomous wireless sensor nodes with wind energy harvesting for remote sensing of wind-driven wildfire spread. *IEEE Transactions on Instrumentation and Measurement*, 60(4):1367–1377, 2011.

[104] Sarbajit Paul and Junghwan Chang. Design of novel electromagnetic energy harvester to power a deicing robot and monitoring sensors for transmission lines. *Energy conversion and management*, 197:111868, 2019.

[105] Kevin McGrattan, Randall McDermott, Craig Weinschenk, and Glenn Forney. Fire dynamics simulator, technical reference guide, sixth edition, 2013.

[106] Cristobal Pais, Jaime Carrasco, David L Martell, Andres Weintraub, and David L Woodruff. Cell2fire: A cell-based forest fire growth model to support strategic landscape management planning. *Frontiers in Forests and Global Change*, 4:692706, 2021.

[107] William F Holmgren, Clifford W Hansen, and Mark A Mikofski. pvlib python: A python package for modeling solar energy systems. *Journal of Open Source Software*, 3(29):884, 2018.

[108] David L King, Jay A Kratochvil, and William Earl Boyson. *Photovoltaic array performance model*, volume 8. Citeseer, 2004.

[109] Barani Design. MeteoTemp RH+T. `https://static1.squarespace.com/static/597dc443914e6bed5fd30dcc/t/618a9376bd08d7243005ce49/1636471677532/MeteoTemp-DataSheet.pdf`, accessed Mar 2023.

[110] Cubic. Low Power Particle Sensor PM2105L. `https://en.gassensor.com.cn/IndoorPMSensor/info_itemid_1830.html`, accessed Mar 2023.

[111] NRG Systems. NRG S1 ANEMOMETER. `https://www.nrgsystems.com/blog/anemometer-wind-speed-threshold/`, accessed Mar 2023.

[112] STMicro. Multiprotocol LPWAN 32-bit Arm® Cortex®-M4 MCUs. https://www.st.com/resource/en/datasheet/stm32wle5jb.pdf, accessed Mar 2023.

[113] Texas Instruments Inc. CC2652R . [Online] https://www.ti.com/product/CC2652R, accessed March 2023.

[114] D Green et al. Heterogeneous Integration at DARPA: Pathfinding and Progress in Assembly Approaches. *ECTC, May*, 2018.

[115] John L Hennessy and David A Patterson. A New Golden Age for Computer Architecture. *Communications of the ACM*, 62(2):48–60, 2019.

[116] Joshua Mack, Sahil Hassan, Nirmal Kumbhare, Miguel Castro Gonzalez, and Ali Akoglu. Cedr: A compiler-integrated, extensible dssoc runtime. *ACM Transactions on Embedded Computing Systems*, 22(2):1–34, 2023.

[117] RF Convergence: From the Signals to the Computer by Dr. Tom Rondeau (Microsystems Technology Office, DARPA). https://futurenetworks.ieee.org/images/files/pdf/FirstResponder/Tom-Rondeau-DARPA.pdf. [Online; last accessed 19-March-2023.].

[118] Kasra Moazzemi, Biswadip Maity, Saehanseul Yi, Amir M Rahmani, and Nikil Dutt. HESSLE-FREE: Heterogeneous Systems Leveraging Fuzzy Control for Runtime Resource Management. *ACM Transactions on Embedded Computing Systems (TECS)*, 18(5s):1–19, 2019.

[119] Pradip Bose et al. Secure and Resilient SoCs for Autonomous Vehicles. In *Proceedings of the International Workshop on Domain Specific System Architecture (DOSSA)*, pages 1–6, 2021.

[120] Luiz F Bittencourt, Rizos Sakellariou, and Edmundo RM Madeira. DAG Scheduling using a Lookahead Variant of the Heterogeneous Earliest Finish Time Algorithm. In *IEEE Euromicro Conference on Parallel, Distributed and Network-based Processing*, pages 27–34, 2010.

[121] J.D. Ullman. NP-Complete Scheduling Problems. *Journal of Computer and System Sciences*, 10(3):384 – 393, 1975.

[122] Haluk Topcuoglu, Salim Hariri, and Min-You Wu. Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing. *IEEE Transactions on Parallel and Distributed Systems*, 13(3):260–274, 2002.

[123] Augusto Vega et al. Stomp: agile evaluation of scheduling policies in heterogeneous multi-processors. In *DOSSA-3 Workshop@ HPCA*, 2021.

[124] Junyan Zhou. Real-time Task Scheduling and Network Device Security for Complex Embedded Systems based on Deep Learning Networks. *Microprocessors and Microsystems*, 79:103282, 2020.

[125] Hongzi Mao, Malte Schwarzkopf, Shaileshh Bojja Venkatakrishnan, Zili Meng, and Mohammad Alizadeh. Learning Scheduling Algorithms for Data Processing Clusters. In *ACM Special Interest Group on Data Communication*, pages 270–288. 2019.

[126] OpenAI. GPT-4 Technical Report, 2023.

[127] Chao Yu, Jiming Liu, Shamim Nemati, and Guosheng Yin. Reinforcement Learning in Healthcare: A Survey. *ACM Computing Surveys (CSUR)*, 55(1):1–36, 2021.

[128] Xinlei Pan, Yurong You, Ziyan Wang, and Cewu Lu. Virtual to Real Reinforcement Learning for Autonomous Driving. *arXiv preprint arXiv:1704.03952*, 2017.

[129] Zhiming Hu, James Tu, and Baochun Li. SPEAR: Optimized Dependency-Aware Task Scheduling with Deep Reinforcement Learning. In *IEEE 39th international conference on distributed computing systems (ICDCS)*, pages 2037–2046, 2019.

[130] Hongzi Mao, Mohammad Alizadeh, Ishai Menache, and Srikanth Kandula. Resource Management with Deep Reinforcement Learning. In *ACM Workshop on Hot Topics in Networks*, pages 50–56, 2016.

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

[132] Samet Egemen Arda et al. DS3: A System-Level Domain-Specific System-on-Chip Simulation Framework. *IEEE Trans. on Computers*, 69(8):1248–1262, 2020.

[133] CEDR - Compiler-integrated Extensible DSSoC Runtime. `https://github.com/ua-rcl/CEDR`. [Online; last accessed 15-May-2022.].

[134] Bart Veltman, BJ Lageweg, and Jan Karel Lenstra. Multiprocessor Scheduling with Communication Delays. *Parallel computing*, 16(2-3):173–182, 1990.

[135] Hoeseok Yang and Soonhoi Ha. Ilp based data parallel multi-task mapping/scheduling technique for mpsoc. In *2008 International SoC Design Conference*, volume 1, pages I–134. IEEE, 2008.

[136] Luca Benini, Davide Bertozzi, and Michela Milano. Resource Management Policy Handling Multiple Use-Cases in MpSoC Platforms using Constraint Programming. In *Logic Programming: 24th International Conference, ICLP 2008 Udine, Italy, December 9-13 2008 Proceedings 24*, pages 470–484. Springer, 2008.

[137] Rizos Sakellariou and Henan Zhao. A Hybrid Heuristic for DAG Scheduling on Heterogeneous Systems. In *Int. Parallel and Distributed Processing Symposium*, page 111. IEEE, 2004.

[138] Kallia Chronaki et al. Criticality-Aware Dynamic Task Scheduling for Heterogeneous Architectures. In *Proceedings of the 29th ACM on International Conference on Supercomputing*, pages 329–338, 2015.

[139] Aporva Amarnath et al. Heterogeneity-Aware Scheduling on SoCs for Autonomous Vehicles. *IEEE Computer Architecture Letters*, 20(2):82–85, 2021.

[140] Yu-Kwong Kwok and Ishfaq Ahmad. Dynamic Critical-Path Scheduling: An Effective Technique for Allocating Task Graphs to Multiprocessors. *IEEE Trans. Parallel Distrib. Syst.*, 7(5):506–521, 1996.

[141] Zhao Tong, Xiaomei Deng, Hongjian Chen, Jing Mei, and Hong Liu. QL-HEFT: A Novel Machine Learning Scheduling Scheme Base on Cloud Computing Environment. *Neural Computing and Applications*, 32:5553–5570, 2020.

[142] Javad Behnamian and SMT Fatemi Ghomi. Multi-Objective Fuzzy Multiprocessor Flowshop Scheduling. *Applied soft computing*, 21:139–148, 2014.

[143] Enda Jiang, Ling Wang, and Jingjing Wang. Decomposition-based Multi-Objective Optimization for Energy-Aware Distributed Hybrid Flow Shop Scheduling with Multiprocessor Tasks. *Tsinghua Science and Technology*, 26(5):646–663, 2021.

[144] Farzaneh Abazari, Morteza Analoui, Hassan Takabi, and Song Fu. MOWS: Multi-Objective Workflow Scheduling in Cloud Computing based on Heuristic Algorithm. *Simulation Modelling Practice and Theory*, 93:119–132, 2019.

[145] A Alper Goksoy et al. DAS: Dynamic Adaptive Scheduling for Energy-Efficient Heterogeneous SoCs. *IEEE Embedded Systems Letters*, 14(1):51–54, 2021.

[146] Haluk Topcuoglu, Salim Hariri, and Min-You Wu. Task Scheduling Algorithms for Heterogeneous Processors. In *Proceedings of the Heterogeneous Computing Workshop (HCW'99)*, pages 3–14, 1999.

[147] Yu-Kwong Kwok and Ishfaq Ahmad. Static Scheduling Algorithms for Allocating Directed Task Graphs to Multiprocessors. *ACM Computing Surveys (CSUR)*, 31(4):406–471, 1999.

[148] Babak Hamidzadeh, Yacine Atif, and David J Lilja. Dynamic Scheduling Techniques for Heterogeneous Computing Systems. *Concurrency: Practice and Experience*, 7(7):633–652, 1995.

[149] Xiaojie Wang, Zhaolong Ning, Song Guo, and Lei Wang. Imitation Learning Enabled Task Scheduling for Online Vehicular Edge Computing. *IEEE Transactions on Mobile Computing*, 21(2):598–611, 2020.

[150] Tegg Taekyong Sung and Bo Ryu. Deep reinforcement learning for system-on-chip: Myths and realities. *IEEE Access*, 10:98048–98064, 2022.

[151] Zachary C Lipton. The mythos of model interpretability: In machine learning, the concept of interpretability is both important and slippery. *Queue*, 16(3):31–57, 2018.

[152] Nicholas Frosst and Geoffrey Hinton. Distilling a neural network into a soft decision tree. *arXiv preprint arXiv:1711.09784*, 2017.

[153] Zihan Ding, Pablo Hernandez-Leal, Gavin Weiguang Ding, Changjian Li, and Ruitong Huang. Cdt: Cascading decision trees for explainable reinforcement learning. *arXiv preprint arXiv:2011.07553*, 2020.

[154] Andrew Silva, Matthew Gombolay, Taylor Killian, Ivan Jimenez, and Sung-Hyun Son. Optimization Methods for Interpretable Differentiable Decision Trees Applied to Reinforcement Learning. In *International Conference on Artificial Intelligence and Statistics*, pages 1855–1865. PMLR, 2020.

[155] Alberto Suárez and James F Lutsko. Globally optimal fuzzy decision trees for classification and regression. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(12):1297–1311, 1999.

[156] Greg Brockman et al. OpenAI Gym. *arXiv preprint arXiv:1606.01540*, 2016.

[157] Ruben Rodriguez Torrado, Philip Bontrager, Julian Togelius, Jialin Liu, and Diego Perez-Liebana. Deep Reinforcement Learning for General Video Game AI. In *IEEE Conference on Computational Intelligence and Games (CIG)*, pages 1–8, 2018.

[158] DS3 Simulator. `https://github.com/segemena/DS3.git`. [Online; last accessed 19-March-2023.].

[159] Shengyi Huang and Santiago Ontañón. A closer look at invalid action masking in policy gradient algorithms. *arXiv preprint arXiv:2006.14171*, 2020.

[160] Marcin Andrychowicz et al. What matters for on-policy deep actor-critic methods? a large-scale study. In *International conference on learning representations*, pages 1–10, 2021.

[161] ZCU102 Evaluation Board. https://www.xilinx.com/support/documentation/boards_and_kits/zcu102/ug1182-zcu102-eval-bd.pdf, Accessed 19 March 2023.

[162] Pauli Virtanen et al. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python, 2020. `https://docs.scipy.org/doc/scipy/reference/generated/scipy.interpolate.RBFInterpolator.html#scipy.interpolate.RBFInterpolator`, accessed 13 November 2022.