DTRL: Decision Tree-based Multi-Objective Reinforcement Learning for Runtime Task Scheduling in Domain-Specific System-on-Chips

TOYGUN BASAKLAR, A. ALPER GOKSOY, and ANISH KRISHNAKUMAR, University of Wisconsin - Madison, USA
SUAT GUMUSSOY, Siemens Corporate Technology, USA
UMIT Y. OGRAS, University of Wisconsin - Madison, USA

Domain-specific systems-on-chip (DSSoCs) combine general-purpose processors and specialized hardware accelerators to improve performance and energy efficiency for a specific domain. The optimal allocation of tasks to processing elements (PEs) with minimal runtime overheads is crucial to achieving this potential. However, this problem remains challenging as prior approaches suffer from non-optimal scheduling decisions or significant runtime overheads. Moreover, existing techniques focus on a single optimization objective, such as maximizing performance. This work proposes DTRL, a decision-tree-based multi-objective reinforcement learning technique for runtime task scheduling in DSSoCs. DTRL trains a single global differentiable decision tree (DDT) policy that covers the entire objective space quantified by a preference vector. Our extensive experimental evaluations using our novel reinforcement learning environment 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.

CCS Concepts: • **Computer systems organization → System on a chip**; • **Hardware → On-chip resource management;**

This work was supported in part by NSF CAREER award CNS-2114499, and DARPA (FA8650-18-2-7860). This material is based on research sponsored by Air Force Research Laboratory (AFRL) and Defense Advanced Research Projects Agency (DARPA) under agreement number FA8650-18-2-7860. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes, notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Air Force Research Laboratory (AFRL) and Defense Advanced Research Projects Agency (DARPA) or the U.S. Government.
This article appears as part of the ESWEEK-TECS special issue and was presented in the International Conference on Compilers, Architectures, and Synthesis for Embedded Systems (CASES), 2023.
Authors' addresses: T. Basaklar, A. A. Goksoy, A. Krishnakumar, and U. Y. Ogras, University of Wisconsin - Madison, 1415 Engineering Dr, Madison, Wisconsin, USA, 53706; emails: {basaklar, agoksoy, anish.n.krishnakumar, uogras}@wisc.edu; S. Gumussoy, Siemens Corporate Technology, 755 College Rd E, Princeton, NJ, USA, 08540; email: suat.gumussoy@siemens.com.

1539-9087/2023/09-ART113 $15.00
https://doi.org/10.1145/3609108

Additional Key Words and Phrases: Domain-specific system-on-chip, task scheduling, reinforcement learning, decision trees, resource management, multi-objective optimization

ACM Reference format:
Toygun Basaklar, A. Alper Goksoy, Anish Krishnakumar, Suat Gumussoy, and Umit Y. Ogras. 2023. DTRL: Decision Tree-based Multi-Objective Reinforcement Learning for Runtime Task Scheduling in Domain-Specific System-on-Chips. *ACM Trans. Embedd. Comput. Syst.* 22, 5s, Article 113 (September 2023), 22 pages.
https://doi.org/10.1145/3609108

1 INTRODUCTION

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) [22, 25]. DSSoCs combine specialized hardware accelerators and general-purpose cores to enhance the performance and energy efficiency of a target domain while providing programming flexibility [3, 35, 38]. 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 [15, 30]. 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 [14, 54]. While static schedulers rely solely on design-time information, dynamic scheduling approaches leverage runtime data to make more informed decisions [52, 55]. Conventional optimization-based approaches can achieve near-optimality but are highly prohibitive due to their complexity and runtime overheads [62]. 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 [52]. Certain machine learning approaches for cluster scheduling [37] and task scheduling [30] 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 [41], healthcare [61], autonomous driving [42], and scheduling [26, 36, 37]. 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 [40]. Multi-objective reinforcement learning (MORL) approaches [24] 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 [34]. 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 [7, 11, 58, 60].

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 paper 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) [45] 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 [10] 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– [28], optimization–, and machine learning–based [30] 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 [28] when the highest preference is given to execution time. At the same time, the same DTRL policy achieves up to 3× 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 [1], 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 [28]. 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.

The rest of this paper is organized as follows. Section 2 reviews the related work. Section 3 provides the necessary background for the proposed DTRL framework. Section 4 presents an environment for a simple and standardized evaluation of DTRL. Then, the DTRL framework is described in Section 5. The extensive experimental evaluations and comparisons with state-of-the-art techniques are presented in Section 6, followed by conclusions and directions for future work in Section 7.

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 1.

Scheduling Techniques: Most scheduling approaches in literature use directed flow graphs (DFGs) to model applications [14, 21, 31, 52]. Optimization-based techniques for DFG scheduling, such as integer linear programming (ILP) and constraint programming (CP) [13, 56, 59], provide optimal design-time solutions. However, these techniques have significant drawbacks. First,

Table 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 [13, 56, 59]	High	Very High	No	No
Static Techniques [14, 44, 52]	Low	Low	No	No
Dynamic Techniques [8, 18, 30, 31]	Moderate	High	No	No
RL Approaches [26, 36, 37, 50]	High	High	No	Yes
Multi-Objective Techniques [6, 12, 29]	Moderate-High	High	Yes	No
DTRL (Proposed approach)	**High**	**Low**	**Yes**	**Yes**

optimization-based approaches struggle to derive the optimal decisions in reasonable runtimes for 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 [32, 44, 51]. 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 [23, 30]. Dynamic scheduling techniques exploit the available runtime information to make more effective decisions [8, 18, 31]. 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 [30, 36, 37, 50, 57]. Imitation learning (IL) approaches such as [30, 57] 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 [26, 36, 37]. Furthermore, they have shown limited performance on high-intensity workloads and only support a single objective, making them unsuitable for adapting to various objectives [49]. 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 [6, 12, 29] 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 [58]. A fine-grained sweep of the preference space can result in a large number of policies, leading to explosive memory requirements [11, 60].

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.

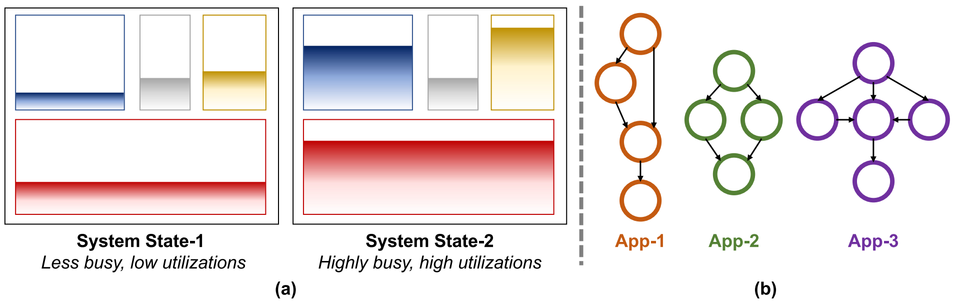

Fig. 1. (a) An illustration of two different system states showing different levels of utilization of system resources (or PEs), and (b) an illustrative example of applications represented as directed flow graphs (DFGs). Nodes in a DFG represent tasks (key computational components) in an application. The edges denote the dependency between tasks and the weight of the edges represents the communication volume between tasks.

3 BACKGROUND AND OVERVIEW

This section overviews the key components of DTRL. Section 3.1 explains the representations of streaming applications as task graphs. Section 3.2 discusses the difference between single- and multiple-objective reinforcement learning formulations. Finally, Section 3.3 describes the differentiable decision tree used as the policy in DTRL.

3.1 Runtime Task Scheduling

DSSoCs provide numerous processing elements for task execution at runtime. As illustrated in Figure 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 Figure 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 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.

3.2 Multi-Objective Reinforcement Learning (MORL)

Task scheduling, at its core, is an NP-hard sequential decision-making problem [14, 54]. 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 $\boldsymbol{r} = [r_1, r_2, \ldots, r_M]^T$, where M is the number of objectives. This vectorized reward can be represented by a vectorized state value function $MV^{\pi}(s)$ [24], as illustrated

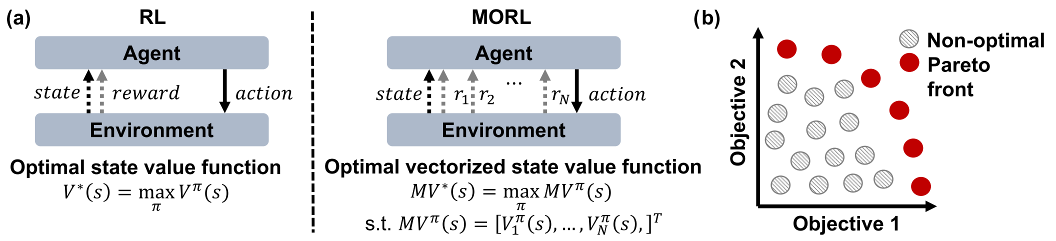
(a)
RL
Agent
state
reward
action
Environment
Optimal state value function
$V^*(s) = \max_\pi V^\pi(s)$
MORL
Agent
state
r_1 r_2 ... r_N
action
Environment
Optimal vectorized state value function
$MV^*(s) = \max_\pi MV^\pi(s)$
s.t. $MV^\pi(s) = [V_1^\pi(s), \ldots, V_N^\pi(s),]^T$
(b)
Objective 2
Objective 1
Non-optimal
Pareto front

Fig. 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.

in Figure 2(a). In the RL domain, *scalarization* is the most commonly used approach to solve multi-objective optimization problems [11, 43, 58, 60]. This approach transforms the reward vector into a single scalar, $f_\omega(r) = \omega^T \boldsymbol{r}$. The MDP is then transformed into a multi-objective Markov decision process (MOMDP), defined by the tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \boldsymbol{r}, \Omega, f_\omega \rangle$, where $\boldsymbol{r}$ and Ω represent the reward *vector* and *preference space*, respectively. Using a preference $\boldsymbol{\omega} \in \Omega$, the function $f_{\boldsymbol{\omega}}(\boldsymbol{r}) = \boldsymbol{\omega}^T \boldsymbol{r}$ 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 Figure 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 5, while the base PPO algorithm details are given in Appendix A.1.

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 [33]. Nevertheless, traditional DTs have a tendency to overfit and fail to generalize well [20].

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 [19, 20, 46]. 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 [46, 48]:

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

where w_η and ϕ_η are the weights and bias of the node η. α_η is the steepness 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 Figure 3. This

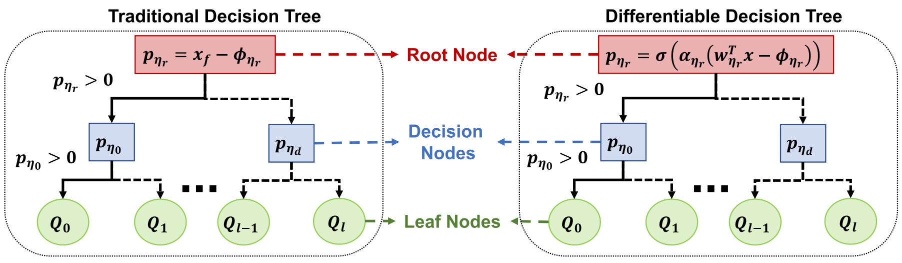
Traditional Decision Tree
Differentiable Decision Tree
$p_{\eta_r} = x_f - \phi_{\eta_r}$
$p_{\eta_r} = \sigma\left(\alpha_{\eta_r}(w_{\eta_r}^T x - \phi_{\eta_r})\right)$
Root Node
$p_{\eta_r} > 0$
p_{η_0}
p_{η_d}
Decision Nodes
$p_{\eta_0} > 0$
Q_0
Q_1
Q_{l-1}
Q_l
Leaf Nodes

Fig. 3. An overview of a traditional decision tree and a differentiable decision tree.

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.

4 RL ENVIRONMENT FOR DSSOCS

OpenAI Gym [16] is a widely popular platform for developing and validating RL algorithms since it provides standardized environments [53]. 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 [2] to integrate it into a Gym environment. We plan to release our integrated infrastructure to the public to stimulate future research.

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

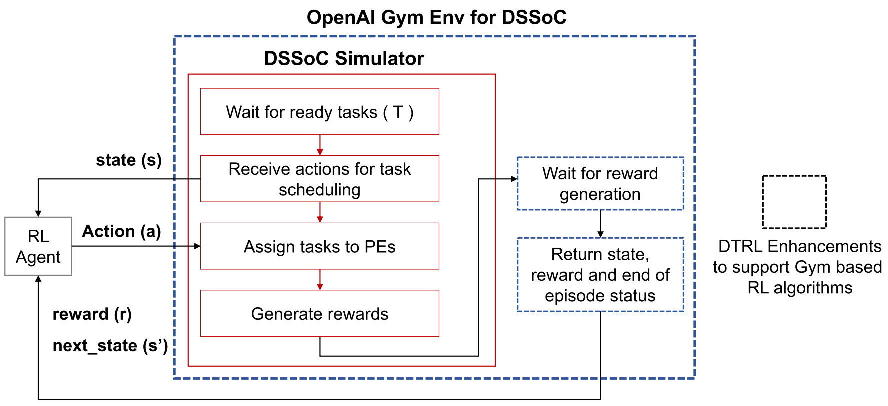
OpenAI Gym Env for DSSoC
DSSoC Simulator
Wait for ready tasks (T)
Receive actions for task scheduling
Assign tasks to PEs
Generate rewards
state (s)
RL Agent
Action (a)
reward (r)
next_state (s')
Wait for reward generation
Return state, reward and end of episode status
DTRL Enhancements to support Gym based RL algorithms

Fig. 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.

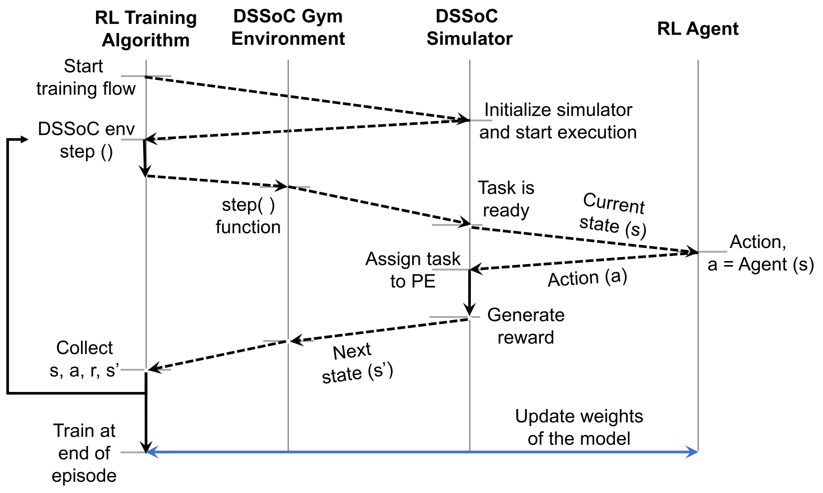

Fig. 5. The end-to-end RL training flow with the DSSoC simulator as a Gym environment. The dashed lines denote the event-driven handshake between the different components in the training process.

To this end, we design handshake events to facilitate communication between the simulator and the RL environment. Figure 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.

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 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 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 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

Table 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	$\mathbb{R}^C$

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.

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.

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 [27] 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 [27] 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 ($l_i,\ i = 1, \dots, |\mathcal{A}|$), 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.

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) [17, 58] 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 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 1, where M is the number of objectives. Therefore, the state value function becomes $\boldsymbol{V_\phi}(s, \boldsymbol{\omega})$, 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:

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

The vectorization of the reward and state value function results in a vectorized advantage function, as presented in Equation (3):

$$\boldsymbol{A}(s_t, a_t, \boldsymbol{\omega}) = \mathbf{r_t} + \gamma \boldsymbol{V_\phi}(s_{t+1}, \boldsymbol{\omega}) - \boldsymbol{V_\phi}(s_t, \boldsymbol{\omega}) \tag{3}$$

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:

$$L_\theta = \frac{1}{T} \sum_{t=0}^{T} min(\rho(\theta)\boldsymbol{\omega}^T \mathbf{A}(s_t, a_t, \boldsymbol{\omega}),\ clip(\rho(\theta), 1-\epsilon, 1+\epsilon)\boldsymbol{\omega}^T \mathbf{A}(s_t, a_t, \boldsymbol{\omega})) \tag{4}$$

$$\rho(\theta) = \frac{\pi_\theta(a_t | s_t, \boldsymbol{\omega})}{\pi_{\theta_{old}}(a_t | s_t, \boldsymbol{\omega})} \tag{5}$$

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 1 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 [9, 45]. *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 $\boldsymbol{A_t}$, target values $\boldsymbol{r_t} + \boldsymbol{V_\phi}(s_{t+1}, \boldsymbol{\omega})$, 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 [9, 45]. We also employ this technique with GAE parameter of 0.95. These

ALGORITHM 1: DTRL Framework

```
Input: Total number of time steps $N$, Number of steps to run per policy rollout $T$, Discount factor $\gamma$,
Number of epochs to update the policy and value network $K$, Minibatch size $b$, Number of child
processes $P$, clipping value $\epsilon$.
Initialize: DDT policy $\pi_\theta$ and value network $\boldsymbol{V_\phi}$ with parameters $\theta$ and $\phi$, Random policy $\pi_\theta$.
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 $\pi_\theta$ 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 $\boldsymbol{A_t}, \boldsymbol{r_t} + \boldsymbol{V_\phi}(s_{t+1}, \boldsymbol{\omega})$, and $\pi_{\theta_{old}}(a_t|s_t, \boldsymbol{\omega})$ using DDT and the value network.
    Transfer populated $(s, a, \boldsymbol{r}, s', \boldsymbol{\omega}, a^m, \boldsymbol{A}, \boldsymbol{r} + \boldsymbol{V_\phi}(s, \boldsymbol{\omega}), \pi_{\theta_{old}}(a|s, \boldsymbol{\omega}))$ 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 $\pi_\theta$.
            Calculate $L_\theta$ and $L_\phi$
            Update $\theta$ and $\phi$ by applying SGD to $L_\theta$ and $L_\phi$.
```

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 Equations (2) and (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 3.

6 EXPERIMENTAL EVALUATION

This section evaluates the proposed DTRL framework for runtime task scheduling in DSSoCs. Section 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 6.2 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.

6.1 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,

Table 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}

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[10].

Simulation and Emulation Frameworks: We first evaluate DTRL using our novel OpenAI Gym environment integrated with an open-source DSSoC simulator, DS3 [2], as described in Section 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 [35] (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 [5]. 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) [28] 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) [30]. This framework uses ETF as the Oracle and trains a regression tree to approximate the Oracle decisions. State-of-the-art MORL approaches [58, 60] 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 [60] requires the action space to be discrete and suffers from sample inefficiency, while PG-MORL [58] 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 runtime overhead compared to DTRL and ETF (as provided in Section 6.2.4) 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 [30]. Therefore, to provide an additional basis for comparison, we introduce a modification of an existing MORL method [39]. 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.

6.2 DTRL Evaluation

A global DDT scheduling policy is obtained by employing the training procedure described in Section 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.

6.2.1 Performance and Energy Consumption Evaluation. Figure 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 Figure 6. The DTRL policy in Figure 6(a) uses a preference vector of $\{1, 0\}$, whereas the results for Scalarized-MOPPO are

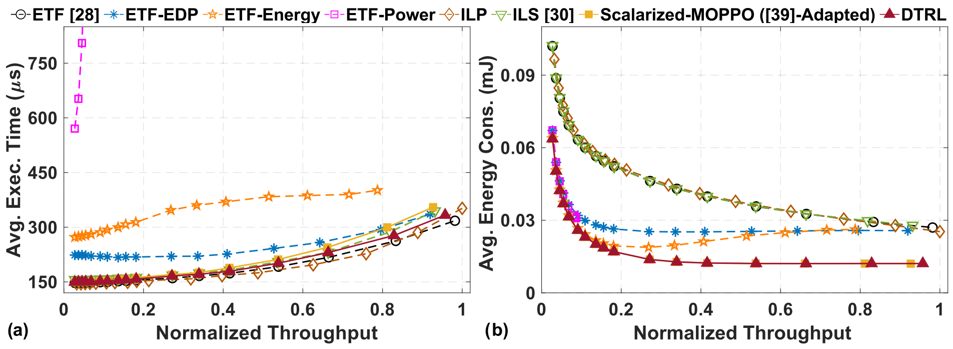

Fig. 6. Comparison of (a) average frame execution time and (b) average energy consumption between ETF [28], ETF-EDP, ETF-Energy, ETF-Power, ILP solution, ILS [30], Scalarized-MOPPO ([39]-Adapted) and DTRL to schedule a workload comprising six streaming applications. The x-axis in Figures 6(a) and 6(b) is normalized to the throughput achieved by the ILP solution.

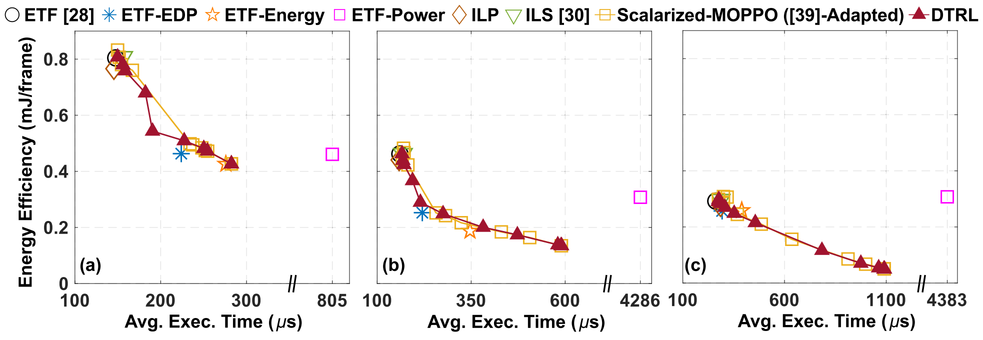

Fig. 7. Average frame execution time (μs) vs. Energy efficiency (mJ / frame) for (a) low (b) medium (c) high target throughputs. Comparison between ETF [28], ETF-EDP, ETF-Energy, ETF-Power, ILP solution, ILS [30], Scalarized-MOPPO ([39]-Adapted), and DTRL are presented. DTRL achieves multiple near-optimal policies using various preferences.

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 6.2.4. 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. *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 Figure 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.

6.2.2 *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. Figures 7(a)–(c) illustrates the curves for low, medium, and high throughput workloads. The evaluation employs preference vectors ($\boldsymbol{\omega}$) separated by a step size of 0.1, $\boldsymbol{\omega} \in \{\{1, 0\}, \{0.9, 0.1\}, \ldots, \{0.1, 0.9\}, \{0, 1\}\}$. Figures 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

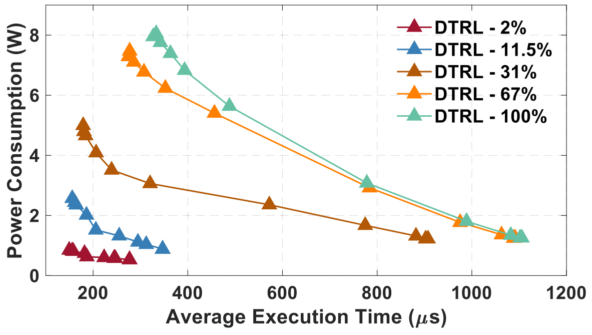

Fig. 8. Pareto front solutions achieved by the proposed DTRL framework that trades off between the average execution time of applications with the power consumption at various target throughputs (in %).

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 6.2.1. 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 Figure 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 trade-off curves at multiple target throughputs strongly demonstrate that DTRL scales to all workload complexities.

6.2.3 Performance Evaluation of DTRL on a Runtime Emulation Platform. We implement DTRL in CEDR [35] and analyze its performance on a Xilinx Zynq ZCU102 FPGA [4]. 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. Figure 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 6.1, ETF incurs high computational complexity due to the quadratic dependency on the number of 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).

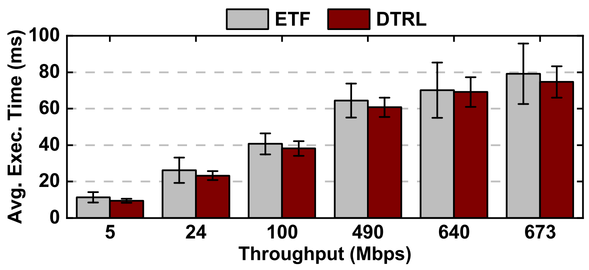

Fig. 9. Comparison of average job execution time between ETF and DTRL on a real hardware platform (Xilinx Zynq ZCU102 FPGA) to schedule a workload comprising three streaming applications. Each bar in the plot represents the average of 50 trials, with each trial consisting of 1390 tasks. Error bars represent the standard deviation of the trials.

6.2.4 Scalability and Limitations. DTRL uses a differentiable decision tree (DDT) at its core to make scheduling decisions. As explained in Section 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. Additionally, 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 CONCLUSION

The optimal allocation of tasks to processing elements (PEs) with minimal runtime overheads is essential to maximize the performance and energy efficiency gains in domain-specific systems-on-chip (DSSoCs). Existing approaches suffer from non-optimal scheduling decisions, high runtime overheads and focus on a single optimization objective. This work presented DTRL, a decision-tree-based multi-objective reinforcement learning technique for runtime task scheduling in DSSoCs. DTRL uses a differentiable decision tree (DDT) policy and a novel reinforcement learning environment for DSSoCs. Experimental evaluations utilizing six domain-specific applications and a comprehensive DSSoC configuration demonstrate that DTRL captures the trade-off between execution time and power consumption, resulting in a Pareto set of solutions using a single DDT policy. Furthermore, DTRL outperforms state-of-the-art heuristic–, optimization–, and machine learning-based schedulers with up to 9× higher performance and up to 3.08× reduction in energy consumption. The trained DDT policy also has negligible runtime overhead, achieving up to 16% higher performance than the state-of-the-art heuristic-based scheduler on the same hardware.

Overall, DTRL provides a promising solution for optimizing the allocation of tasks in DSSoCs, enabling improved performance and energy efficiency in a range of applications. The core algorithm presented in Section 5 can be readily applied to various domains that can be characterized by multiple objectives and discrete action spaces. The proposed approach requires only designing an RL environment specific to the target domain. For example, existing problems like wildfire tracking or games can be effectively addressed using DDT models within the reinforcement learning framework [47], often incorporating additional objectives as constraints. These domains can be expanded to accommodate multiple objectives and effectively tackled using the DTRL approach. DTRL serves as a highly adaptable and deployable solution for multi-objective reinforcement learning, providing considerable flexibility in addressing complex problem domains. Future directions include adding more optimization objectives, further minimizing the runtime overheads, and exploring the applicability of DTRL in other domains. Another key future work is to enable the interpretability of the DDT policy, investigating each node's behavior in the tree.

A APPENDIX

A.1 Proximal Policy Optimization (PPO)

Proximal policy optimization (PPO) [45] is a policy gradient algorithm that aims to improve the training stability of the policy by updating it conservatively according to a certain surrogate objective function. Policy gradient algorithms typically update the policy network by computing the gradient of the policy, multiplied by the discounted cumulative rewards, and using it as a loss function with a gradient ascent algorithm. This update is typically performed using samples from multiple episodes since the discounted cumulative rewards can vary widely due to the different trajectories followed by each episode. To mitigate this variance, an advantage function is introduced as a bias to quantify the benefits the goodness of taking action a in state s and is represented as:

$$A(s_t, a_t) = r_t + \gamma V_\phi(s_{t+1}) - V_\phi(s_t) \tag{6}$$

Here, $\gamma \in [0, 1]$ is the discount factor, and $V_\phi(s)$ is the value network that estimates the expected discounted sum of rewards for a given state s.

At each optimization step during training, the PPO algorithm forces the distance between the new policy ($\pi_\theta(a|s)$) and the old policy ($\pi_{\theta old}(a|s)$) to be small. It achieves its goal using the following loss function and the advantage function:

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

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

where, T is the total time steps of collected data. The equation presented involves two policies: $\pi_{\theta old}(a|s)$, which is used to collect samples by interacting with the environment, and $\pi_\theta(a|s)$, which is being updated using the loss function. PPO introduces a constraint on the difference between $\pi_{\theta old}(a|s)$ and $\pi_\theta(a|s)$ by applying a clipping operation on the ratio $\rho(\theta)$ between two distributions, with the clipping threshold ϵ being a hyperparameter of the algorithm. Additionally, an entropy term may be added to the loss function to promote sufficient exploration.

During training, the value network $V_\phi(s)$ is also updated by minimizing the mean-squared error between estimated and target values using gradient descent as the optimization algorithm:

$$L_\phi = \frac{1}{T}\sum_{t=0}^{T}(V_\phi(s_t) - (r_t + \gamma V_\phi(s_{t+1})))^2 \tag{9}$$

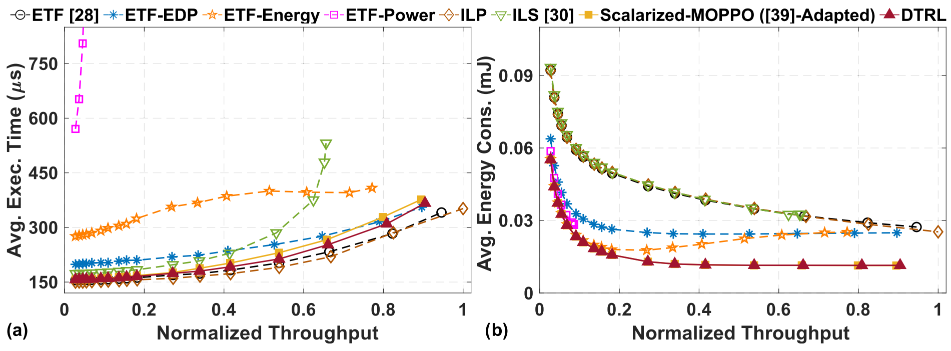

Fig. 10. Comparison of a) average frame execution time and b) average energy consumption between ETF [28], ETF-EDP, ETF-Energy, ETF-Power, ILP solution, ILS [30], Scalarized-MOPPO ([39]-Adapted) and DTRL to schedule a workload comprising six streaming applications. The x-axis in Figures 6(a) and 6(b) is normalized to the throughput achieved by the ILP solution.

The multi-objective variant of PPO algorithm for runtime task scheduling is described in Section 5.2.

A.2 Additional Experimental Results

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.

Figure 10(a) compares the average frame execution time of DTRL with baseline and state-of-the-art schedulers. The DTRL policy in Figure 10(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-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 6.2.4. 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 Figure 10(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.

Figures 11(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 ($\boldsymbol{\omega}$) separated by a step size of 0.1, $\boldsymbol{\omega} \in \{\{1, 0\}, \{0.9, 0.1\}, \ldots, \{0.1, 0.9\}, \{0, 1\}\}$. Figures 11(a)–(c) also shows the energy efficiency (in milli-Joules per frame) of

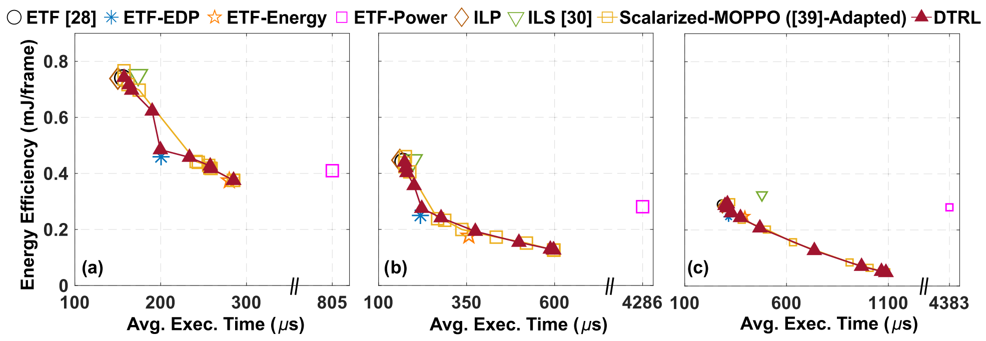

Fig. 11. Average frame execution time (μs) vs. Energy efficiency (mJ / frame) for (a) low (b) medium (c) high target throughputs. Comparison between ETF [28], ETF-EDP, ETF-Energy, ETF-Power, ILP solution, ILS [30], Scalarized-MOPPO ([39]-Adapted), and DTRL are presented. DTRL achieves multiple near-optimal policies using various preferences.

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.

REFERENCES

[1] [n. d.]. CEDR - Compiler-integrated Extensible DSSoC Runtime. https://github.com/ua-rcl/CEDR. [Online; last accessed 15-May-2022.].
[2] [n. d.]. DS3 Simulator. https://github.com/segemena/DS3.git. [Online; last accessed 19-March-2023.].
[3] [n. d.]. 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.].
[4] [n. d.]. ZCU102 Evaluation Board. https://www.xilinx.com/support/ documentation/boards_and_kits/zcu102/ug1182-zcu102-eval-bd.pdf, Accessed 19 March 2023.
[5] 2009. V12.8: User's manual for CPLEX. *International Business Machines Corporation* 46, 53 (2009), 157.
[6] Farzaneh Abazari, Morteza Analoui, Hassan Takabi, and Song Fu. 2019. MOWS: Multi-objective workflow scheduling in cloud computing based on heuristic algorithm. *Simulation Modelling Practice and Theory* 93 (2019), 119–132.
[7] Abbas Abdolmaleki et al. 2020. A distributional view on multi-objective policy optimization. In *International Conference on Machine Learning*. PMLR, 11–22.
[8] Aporva Amarnath et al. 2021. Heterogeneity-aware scheduling on SoCs for autonomous vehicles. *IEEE Computer Architecture Letters* 20, 2 (2021), 82–85.
[9] Marcin Andrychowicz et al. 2021. What matters for on-policy deep actor-critic methods? A large-scale study. In *International Conference on Learning Representations*. 1–10.
[10] Samet Egemen Arda et al. 2020. DS3: A system-level domain-specific system-on-chip simulation framework. *IEEE Trans. on Computers* 69, 8 (2020), 1248–1262.
[11] Toygun Basaklar, Suat Gumussoy, and Umit Y Ogras. 2022. PD-MORL: Preference-driven multi-objective reinforcement learning algorithm. *arXiv preprint arXiv:2208.07914* (2022).
[12] Javad Behnamian and SMT Fatemi Ghomi. 2014. Multi-objective fuzzy multiprocessor flowshop scheduling. *Applied Soft Computing* 21 (2014), 139–148.
[13] Luca Benini, Davide Bertozzi, and Michela Milano. 2008. 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*. Springer, 470–484.
[14] Luiz F. Bittencourt, Rizos Sakellariou, and Edmundo R. M. Madeira. 2010. DAG scheduling using a lookahead variant of the heterogeneous earliest finish time algorithm. In *IEEE Euromicro Conference on Parallel, Distributed and Network-based Processing*. 27–34.

[15] Pradip Bose et al. 2021. Secure and resilient SoCs for autonomous vehicles. In *Proceedings of the International Workshop on Domain Specific System Architecture (DOSSA'21)*. 1–6.
[16] Greg Brockman et al. 2016. OpenAI Gym. *arXiv preprint arXiv:1606.01540* (2016).
[17] Xi Chen, Ali Ghadirzadeh, Mårten Björkman, and Patric Jensfelt. 2019. Meta-learning for multi-objective reinforcement learning. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'19)*. IEEE, 977–983.
[18] Kallia Chronaki et al. 2015. Criticality-aware dynamic task scheduling for heterogeneous architectures. In *Proceedings of the 29th ACM on International Conference on Supercomputing*. 329–338.
[19] Zihan Ding, Pablo Hernandez-Leal, Gavin Weiguang Ding, Changjian Li, and Ruitong Huang. 2020. Cdt: Cascading decision trees for explainable reinforcement learning. *arXiv preprint arXiv:2011.07553* (2020).
[20] Nicholas Frosst and Geoffrey Hinton. 2017. Distilling a neural network into a soft decision tree. *arXiv preprint arXiv:1711.09784* (2017).
[21] A. Alper Goksoy et al. 2021. DAS: Dynamic adaptive scheduling for energy-efficient heterogeneous SoCs. *IEEE Embedded Systems Letters* 14, 1 (2021), 51–54.
[22] D. Green et al. 2018. Heterogeneous integration at DARPA: Pathfinding and progress in assembly approaches. *ECTC, May* (2018).
[23] Babak Hamidzadeh, Yacine Atif, and David J Lilja. 1995. Dynamic scheduling techniques for heterogeneous computing systems. *Concurrency: Practice and Experience* 7, 7 (1995), 633–652.
[24] Conor F. Hayes et al. 2022. A practical guide to multi-objective reinforcement learning and planning. *Autonomous Agents and Multi-Agent Systems* 36, 1 (2022), 1–59.
[25] John L. Hennessy and David A. Patterson. 2019. A new golden age for computer architecture. *Commun. ACM* 62, 2 (2019), 48–60.
[26] Zhiming Hu, James Tu, and Baochun Li. 2019. SPEAR: Optimized dependency-aware task scheduling with deep reinforcement learning. In *IEEE 39th International Conference on Distributed Computing Systems (ICDCS'19)*. 2037–2046.
[27] Shengyi Huang and Santiago Ontañón. 2020. A closer look at invalid action masking in policy gradient algorithms. *arXiv preprint arXiv:2006.14171* (2020).
[28] Jing-Jang Hwang, Yuan-Chieh Chow, Frank D. Anger, and Chung-Yee Lee. 1989. Scheduling precedence graphs in systems with interprocessor communication times. *SIAM J. Comput.* 18, 2 (1989), 244–257.
[29] Enda Jiang, Ling Wang, and Jingjing Wang. 2021. Decomposition-based multi-objective optimization for energy-aware distributed hybrid flow shop scheduling with multiprocessor tasks. *Tsinghua Science and Technology* 26, 5 (2021), 646–663.
[30] Anish Krishnakumar et al. 2020. Runtime task scheduling using imitation learning for heterogeneous many-core systems. *IEEE Transactions on CAD of Integrated Circuits and Systems* 39, 11 (2020), 4064–4077.
[31] Yu-Kwong Kwok and Ishfaq Ahmad. 1996. Dynamic critical-path scheduling: An effective technique for allocating task graphs to multiprocessors. *IEEE Trans. Parallel Distrib. Syst.* 7, 5 (1996), 506–521.
[32] Yu-Kwong Kwok and Ishfaq Ahmad. 1999. Static scheduling algorithms for allocating directed task graphs to multiprocessors. *ACM Computing Surveys (CSUR)* 31, 4 (1999), 406–471.
[33] Zachary C. Lipton. 2018. The mythos of model interpretability: In machine learning, the concept of interpretability is both important and slippery. *Queue* 16, 3 (2018), 31–57.
[34] Chunming Liu, Xin Xu, and Dewen Hu. 2014. Multiobjective reinforcement learning: A comprehensive overview. *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 45, 3 (2014), 385–398.
[35] Joshua Mack, Sahil Hassan, Nirmal Kumbhare, Miguel Castro Gonzalez, and Ali Akoglu. 2023. CEDR: A compiler-integrated, extensible DSSoC runtime. *ACM Transactions on Embedded Computing Systems* 22, 2 (2023), 1–34.
[36] Hongzi Mao, Mohammad Alizadeh, Ishai Menache, and Srikanth Kandula. 2016. Resource management with deep reinforcement learning. In *ACM Workshop on Hot Topics in Networks*. 50–56.
[37] Hongzi Mao, Malte Schwarzkopf, Shaileshh Bojja Venkatakrishnan, Zili Meng, and Mohammad Alizadeh. 2019. Learning scheduling algorithms for data processing clusters. In *ACM Special Interest Group on Data Communication*. 270–288.
[38] Kasra Moazzemi, Biswadip Maity, Saehanseul Yi, Amir M. Rahmani, and Nikil Dutt. 2019. HESSLE-FREE: Heterogeneous systems leveraging fuzzy control for runtime resource management. *ACM Transactions on Embedded Computing Systems (TECS)* 18, 5s (2019), 1–19.
[39] Hossam Mossalam, Yannis M. Assael, Diederik M. Roijers, and Shimon Whiteson. 2016. Multi-objective deep reinforcement learning. *arXiv preprint arXiv:1610.02707* (2016).
[40] Aviv Navon, Aviv Shamsian, Gal Chechik, and Ethan Fetaya. 2020. Learning the pareto front with hypernetworks. *arXiv preprint arXiv:2010.04104* (2020).
[41] OpenAI. 2023. GPT-4 Technical Report. arXiv:2303.08774 [cs.CL]
[42] Xinlei Pan, Yurong You, Ziyan Wang, and Cewu Lu. 2017. Virtual to real reinforcement learning for autonomous driving. *arXiv preprint arXiv:1704.03952* (2017).

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

[44] Rizos Sakellariou and Henan Zhao. 2004. A hybrid heuristic for DAG scheduling on heterogeneous systems. In *Int. Parallel and Distributed Processing Symposium*. IEEE, 111.

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

[46] Andrew Silva, Matthew Gombolay, Taylor Killian, Ivan Jimenez, and Sung-Hyun Son. 2020. Optimization methods for interpretable differentiable decision trees applied to reinforcement learning. In *International Conference on Artificial Intelligence and Statistics*. PMLR, 1855–1865.

[47] Andrew Silva, Taylor Killian, Ivan Dario Jimenez Rodriguez, Sung-Hyun Son, and Matthew Gombolay. 2019. Optimization methods for interpretable differentiable decision trees in reinforcement learning. *arXiv preprint arXiv:1903.09338* (2019).

[48] Alberto Suárez and James F. Lutsko. 1999. Globally optimal fuzzy decision trees for classification and regression. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 21, 12 (1999), 1297–1311.

[49] Tegg Taekyong Sung and Bo Ryu. 2022. Deep reinforcement learning for system-on-chip: Myths and realities. *IEEE Access* 10 (2022), 98048–98064.

[50] Zhao Tong, Xiaomei Deng, Hongjian Chen, Jing Mei, and Hong Liu. 2020. QL-HEFT: A novel machine learning scheduling scheme base on cloud computing environment. *Neural Computing and Applications* 32 (2020), 5553–5570.

[51] Haluk Topcuoglu, Salim Hariri, and Min-You Wu. 1999. Task scheduling algorithms for heterogeneous processors. In *Proceedings of the Heterogeneous Computing Workshop (HCW'99)*. 3–14.

[52] Haluk Topcuoglu, Salim Hariri, and Min-You Wu. 2002. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Transactions on Parallel and Distributed Systems* 13, 3 (2002), 260–274.

[53] Ruben Rodriguez Torrado, Philip Bontrager, Julian Togelius, Jialin Liu, and Diego Perez-Liebana. 2018. Deep reinforcement learning for general video game AI. In *IEEE Conference on Computational Intelligence and Games (CIG'18)*. 1–8.

[54] J. D. Ullman. 1975. NP-complete scheduling problems. *J. Comput. System Sci.* 10, 3 (1975), 384–393. https://doi.org/10.1016/S0022-0000(75)80008-0

[55] Augusto Vega et al. 2021. STOMP: Agile evaluation of scheduling policies in heterogeneous multi-processors. In *DOSSA-3 Workshop@ HPCA*.

[56] Bart Veltman, B. J. Lageweg, and Jan Karel Lenstra. 1990. Multiprocessor scheduling with communication delays. *Parallel Computing* 16, 2-3 (1990), 173–182.

[57] Xiaojie Wang, Zhaolong Ning, Song Guo, and Lei Wang. 2020. Imitation learning enabled task scheduling for online vehicular edge computing. *IEEE Transactions on Mobile Computing* 21, 2 (2020), 598–611.

[58] Jie Xu et al. 2020. Prediction-guided multi-objective reinforcement learning for continuous robot control. In *International Conference on Machine Learning*. PMLR, 10607–10616.

[59] Hoeseok Yang and Soonhoi Ha. 2008. ILP based data parallel multi-task mapping/scheduling technique for MPSoC. In *2008 International SoC Design Conference*, Vol. 1. IEEE, I–134.

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

[61] Chao Yu, Jiming Liu, Shamim Nemati, and Guosheng Yin. 2021. Reinforcement learning in healthcare: A survey. *ACM Computing Surveys (CSUR)* 55, 1 (2021), 1–36.

[62] Junyan Zhou. 2020. Real-time task scheduling and network device security for complex embedded systems based on deep learning networks. *Microprocessors and Microsystems* 79 (2020), 103282.

Received 23 March 2023; revised 2 June 2023; accepted 30 June 2023