Article

MASA: Multi-Application Scheduling Algorithm for Heterogeneous Resource Platform

Quan Peng and Shan Wang *

College of Electronic Science and Technology, National University of Defense Technology, Changsha 410005, China; pengquan21@nudt.edu.cn
* Correspondence: chinafir@nudt.edu.cn

Abstract: Heterogeneous architecture-based systems-on-chip enable the development of flexible and powerful multifunctional RF systems. In complex and dynamic environments where applications arrive continuously and stochastically, real-time scheduling of multiple applications to appropriate processor resources is crucial for fully utilizing the heterogeneous SoC's resource potential. However, heterogeneous resource-scheduling algorithms still face many problems in practical situations, including generalized abstraction of applications and heterogeneous resources, resource allocation, efficient scheduling of multiple applications in complex mission scenarios, and how to ensure the effectiveness combining with real-world applications of scheduling algorithms. Therefore, in this paper, we design the Multi-Application Scheduling Algorithm, named MASA, which is a two-phase scheduler architecture based on Deep Reinforcement Learning. The algorithm is made up of neural network scheduler-based task prioritization for dynamic encoding of applications and heuristic scheduler-based task mapping for solving the processor resource allocation problem. In order to achieve stable and fast training of the network scheduler based on the actor–critic strategy, we propose optimization methods for the training of MASA: reward dynamic alignment (RDA), earlier termination of the initial episodes, and asynchronous multi-agent training. The performance of the MASA is tested with classic directed acyclic graph and six real-world application datasets, respectively. Experimental results show that MASA outperforms other neural scheduling algorithms and heuristics, and ablation experiments illustrate how these training optimizations improve the network's capacity.

Keywords: multiapplication scheduling; heterogeneous resources; combinatorial optimization; deep reinforcement learning; training optimization methods

Citation: Peng, Q.; Wang, S. MASA: Multi-Application Scheduling Algorithm for Heterogeneous Resource Platform. *Electronics* **2023**, *12*, 4056. https://doi.org/10.3390/electronics12194056

Academic Editor: Lukáš Kohútka

Received: 1 September 2023
Revised: 24 September 2023
Accepted: 26 September 2023
Published: 27 September 2023

1. Introduction

With the rapid development of software-defined technologies and hardware architectures, next-generation terminal systems are possible that can simultaneously implement communication, radar, timing, and other radio-frequency (RF) functions [1]. Heterogeneous System-on-Chip (SoC) integrates General Purpose, Special Purpose, and other types of processor resources (FPGAs, Accelerators, etc.) on a single chip [2], realizing a balance between performance and energy and enabling the development of flexible and powerful multifunction RF systems. As the complexity of RF functions increases, heterogeneous SoCs that power multifunctional RF systems must meet stringent performance requirements before deployment. For example, RF systems in autonomous vehicles must simultaneously process multiple applications under highly dynamic environmental conditions to accomplish their missions safely [3,4]. Therefore, how to efficiently schedule multiple strong real-time applications onto a limited heterogeneous resource system is the key to fully utilize the potential of heterogeneous SoC, as well as to enhance the flexibility and scalability of the terminal's RF system.

Efficient scheduling of applications on heterogeneous resource requires complex combinatorial optimization algorithms. The prime objective of scheduling algorithms is to optimize the deployment of structurally different RF applications, such as communications and radar, by assigning the application's different functional modules to the appropriate Processing Elements (PEs) in the heterogeneous resources. Meanwhile, intra-application dependencies are generally described by the form of a directed acyclic graph (DAG) that illustrates the relationship between tasks in a point-to-point format, and these tasks can organize fine-grained parallel computation on different PEs. Tasks and PEs should be matched as closely as possible. If a processor with low computing capability is assigned to a task with high computational demand, it could easily cause the task to become a computational bottleneck for the system. Conversely, if a fast processor is assigned to a task with low computational demand, the processor's efficiency cannot be guaranteed. Furthermore, in comparison to scheduling a single application, the task of runtime scheduling for multiple applications becomes significantly more complicated. To address this issue, this paper to study a multi-application scheduling algorithm designed for heterogeneous resources.

Static scheduling algorithms generally generate optimal schedules for individual applications on a resource system. However, compared to the scheduling of a single application, the processing of multiple applications for heterogeneous runtime scheduling is more complex. Currently, the neural network approach has proved its advancement in resource management in cloud computing, edge computing and other environments [5–9], so this paper draws on its ideas to apply neural network scheduling algorithms to the resource management problem of terminal heterogeneous resources. Therefore, for the characteristics of continuous stochastic arrival of applications, interdependence among application and terminal resource heterogeneity in multifunctional RF systems, this paper designs a Multi-Application Scheduling Algorithm (MASA) based on Deep Reinforcement Learning (DRL) to minimize the average application execution time (denoted as avg. JET) as the optimization global objective.

The main contributions of this paper are summarized as follows:

1. Designing a DRL-based multi-application scheduling algorithm-MASA, which consists of two parts: a neural network scheduler (NN-Scheduler) and a Heuristic scheduler, to solve heterogeneous resource management in a dynamic environment.
2. Adopting a self-attention mechanism [10] in NN-Scheduler to realize feature vector extraction for multiple applications, and a hybrid allocation algorithm DEFT in Heuristic scheduler to reduce the idle time of PE and improve the actual performance of scheduling.
3. Proposing optimizing methods for network training with respect to the practical problems of terminal heterogeneous resources. The Reward Dynamic Alignment (RDA) method is proposed to obtain the correct empirical data, initial episodes early termination method and asynchronous multi-agents joint training method to improve the speed and utility of network training.

The rest of the paper is organized as follows. Section 2 presents research related to the task scheduling algorithm. Section 3 introduces the multi-application scheduling scenario and defines the average application execution time minimization problem. Section 4 proposes the multi-application scheduling algorithm-MASA and presents RDA, early termination of the initial episodes and asynchronous multi-agents joint training optimization methods. Simulation results of the proposed algorithm are discussed in Section 5 to validate its performance. Finally, the paper is summarized in Section 6.

2. Related Work

The complexity of DAG scheduling has been demonstrated to be NP-complete [11], and its complexity is greatly increased when PE resources are heterogeneous. Scheduling strategies can broadly be classified into two types: dynamic and static DAG scheduling. In static DAG scheduling, the DAG information is known in advance (at the beginning of the scheduling process). Conversely, the DAG topology in dynamic scheduling is unknown in advance, and all configuration information is obtained at runtime. Scheduling decisions are then made in real time.

Most current static scheduling algorithms on heterogeneous resources aim to optimize the time required to execute a single DAG. For example, the Mixed Integer Programming approach uses objective functions and constraints to formulate the task scheduling problem, exchanging computational and time complexity for an optimal solution [12]. Genetic algorithms and particle swarm algorithms based on heuristic stochastic search algorithms can find the optimal value of the parameter set through a large number of iterative experiments, and such algorithms can be applied to multi-objective optimization problems [13–16]. Heuristic-based list task scheduling algorithms demonstrate their high execution capabilities in a heterogeneous processor resource. HEFT (Heterogeneous Earliest Finish Time) is the dominant algorithm for heterogeneous static list scheduling [17]. HEFT assigns priority to each task based on the critical path length, and later assigns PEs to tasks with the highest priority on the list based on the earliest finish time (EFT),mapping the task to the active PE with the minimum EFT value; in addition, HEFT uses a scheme that tries to insert tasks into the PE's idle time slots, and CPATH is a critical path aware dynamic scheduler for heterogeneous systems that prioritizes tasks in the DAG based on the lowest-cost longest-path approach, submitting high priority tasks to high frequency PEs and low frequency PEs when work is enabled. This scheduling approach does not consider communication costs for tasks [18]. These algorithms have been further expanded and extended in recent years and applied to the scheduling of real hardware platforms, such as FPGA [19].

Dynamic scheduling algorithms for multiple DAGs are also investigated, and many machine learning based scheduling algorithms for dynamic scheduling scenarios are described below. Xie G et al. proposed an adaptive scheduler (ADS) based on RL with the aim of improving reliability and minimizing completion time. However, ADS dynamically schedules DAGs with higher-level DAGs from the priority level of DAGs alone, and the large computational granularity is not suitable for radio application streaming analysis [3,20]. Biao H et al. investigated how to dynamically schedule dynamic applications on heterogeneous embedded systems and proposed energy-efficient scheduling algorithms for the optimization of the system energy consumption problem. However, this paper uses an objective function and a constraint formulation to design the scheduling problem, which has a relatively high time complexity [21]. Tegg T et al. proposed SoCRATE, a task scheduling scheme for the domain of System-on-a-Chip based on the DRL algorithm. SoCRATE employs an integrated network structure to generate scheduling decisions. This results in a higher level of complexity in the neural network's decision space, which prevents the neural network from converging to a steady state more quickly [22]. Amarnath A et al. designed a heterogeneous-aware multilevel scheduler, HetSched, which utilizes information about the runtime state of the underlying heterogeneous SoCs as well as the real-time nature of the application to meet the growing throughput requirements of self-driving cars. However, it takes into account the task security level and the granularity of task division is large, which is not adapted to the RF application environment [4]. Mao H et al. proposed the Decima algorithm to solve the task scheduling problem in the field of cloud computing with a combination of graph neural network and actor–critic training strategy, but Decima is used to solve the isomorphic resource-scheduling problem and is not applicable to the scheduling problem of heterogeneous resource systems [23].

3. A Multi-Application Scheduling Formal Definition

Resource management for terminal heterogeneous SoCs is a domain-specific resource management problem, and the following are some of the definitions that will be used later.

Definition 1. *An RF application consists of multiple specific function modules, such as Fast Fourier Transforms (FFTs), encoders and decoders, and other computationally intensive software components. In this paper, an application (WiFi-TX/RX, Range Detection, etc.) is defined as a* *job, and a function module is defined as a task. The terms task and node are used interchangeably throughout this paper.*

Definition 2. *Task scheduling includes task prioritization and task mapping, where task mapping describes the process of assigning PEs to tasks.*

3.1. Scheduling Scenario Description

RF signal-processing applications are mostly computationally intensive tasks, and assigning the task to a suitable PE is an essential step, e.g., the FFT can be processed 10 times faster on a hardware accelerator than on a GPP. If directly using the neural network to process both job DAGs and PE information and generate executable tasks and available PE pairs, the complexity of the neural network's decision space is exponentially increased, preventing the neural network from converging to a steady state. Therefore, in this paper, we design a two-phase scheduling algorithm MASA to reduce the complexity of the decision space, i.e., task prioritization and task-mapping phases, as shown in Figure 1. Firstly, in the task prioritization phase, the NN-Scheduler mainly processes the information of job DAG without considering the information of PE. Second, in the task-mapping phase, the Heuristic Scheduler makes decisions based only on the identified high-priority tasks and PE information.

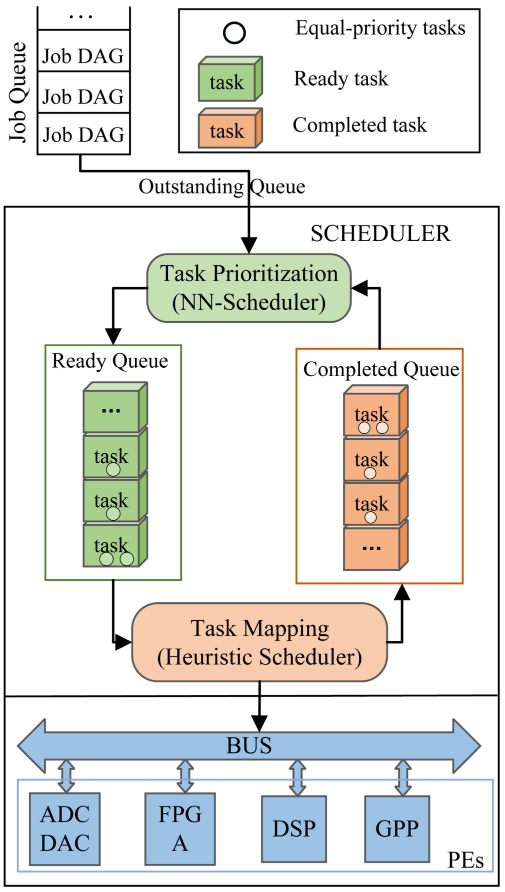


Figure 1. Multi-application scheduling scenario.

The NN-Scheduler sorts the tasks in the multi-DAG, and the high priority tasks are moved to the ready queue to wait for execution, while the low priority tasks waiting for pretasks to be completed are left in the outstanding queue. The hardware database includes static profiles (PE type, estimated computational execution time for each task, maximum number of tasks to be processed by the PE) and dynamic profiles (PE utilization, status of the PE as busy or idle) of the PE. In each mapping phase, ready tasks are deployed to available PEs according to the Heuristic Scheduler and assigned to executable tasks after waiting for the PEs to become idle. After PEs have processed the executable tasks, the SCHEDULER removes the dependencies of the tasks and updates the status information of the tasks and the PEs. After the above is completed, a new scheduling decision phase is triggered and the NN-Scheduler moves the outstanding tasks to the ready queue if all of their predecessors have been deleted. After each scheduling decision is completed, SCHEDULER updates the environment state to continue in subsequent decision phases until all waiting tasks in the outstanding queue are removed.

3.2. Optimization Objective

The simulation environment for heterogeneous resources includes m independent jobs, referred to as $J = [j_1, j_2, \ldots, j_m]$, and K heterogeneous processing elements (PE), referred to as $P = [p_1, p_2, \ldots, p_K]$. It is necessary to investigate the impact of randomly arriving jobs (streaming mode) on the scheduling algorithm in different mission scenarios of varying scales, thus the *scale* shown in Algorithm 1 is an adjustable parameter value.

Algorithm 1. Flowchart of Heterogeneous Resource Simulation Environment

```
1.  Input: Job's profile $J = [j_1, \ldots, j_m]$, PE's profile $P = [p_1, \ldots, p_K]$, Maximum simulation
    length CLK, Job Queue capacity maximum C, job inject interval' expectation Scale
2.  Output: Average execution time for multi-application (avg.JET)
3.  for each episode do
4.    clk=0
5.    $clk_{inj} \sim \exp\left(\frac{1}{Scale}\right)$
6.     repeat
7.       if Job Queue capacity maximum not reached
8.         Inject job into the Job Queue every $clk_{inj}$
9.       end if
10.      for all tasks for each job in the job Queue do
11.        Task coding based on feature networks
12.        Task ordering based on policy network
13.        for Ready Task do
14.            Mapping task to PE based on heuristic algorithm
15.        end for
16.      end for
17.      $clk \leftarrow clk + 1$
18.    until $clk \rightarrow CLK$
19.    Computer avg.JET using Equation (5)
20. end for
```

Each job is modeled as a DAG, denoting the job DAG as $G\{N, E\}$, in which each task $n_i \in N$ represents the software components that composed the application, and the directed edge $e_{i,j} \in E$ is the link from task n_i to n_j. For any $n_i \in N$, the parent node has a higher priority than the child nodes. $comp(n_i, p_m)$ represents the computational execution time of task n_i on p_m, as provided by the profile of the heterogeneous resource PE. $comm(n_i, n_j)$ represents the communication delay from n_i to n_j, as shown in Equation (1). $w_{i,j}$ is the transmission data volume between n_i and n_j, task n_i is mapped to processor p_m, denoted $p_{m(n_i)}$. $B\left(p_{m\,(n_i)}, p_{n\,(n_j)}\right)$ denotes the bandwidth between p_m and p_n. For task n_i and its parent node $pred(n_i)$, assignment to different PEs incurs communication overhead, and frequent switching of PEs leads to an increase in task execution time.

$$comm(n_i, n_j) = \max_{n_j \in pred(n_i)} \frac{w_{i,j}}{B\left(p_{m(n_i)}, p_{n(n_j)}\right)} \tag{1}$$

The earliest start time (EST) and earliest finish time (EFT) of task n_i on processor p_m are given by Equations (2) and (3), where $\mathrm{pred}(n_i)$ is the parent node of task n_i, avail $[p_m]$ denotes the earliest time that p_m can be used to perform the task, and AFT (n_j) is the time that n_j actually finishes.

$$\mathrm{EST}(n_i, p_m) = \max\left\{\mathrm{avail}[p_m], \max_{n_j \in \mathrm{pred}(n_i)}\left(\mathrm{AFT}(n_j) + \mathrm{comm}(n_j, n_i)\right)\right\} \tag{2}$$

$$\mathrm{EFT}(n_i, p_m) = \mathrm{comp}(n_i, p_m) + \mathrm{EST}(n_i, p_m) \tag{3}$$

The true execution time of a completed application $j \in J_{comp}$ is exec (j), given by Equation (4), where n_{exit} is the exit node of application and AFT (n_{exit}) is the actual completion time of the exit node.

$$\mathrm{exec}(j) = \max\{\mathrm{AFT}(n_{exit})\} \tag{4}$$

In this paper, the final objective of the scheduling algorithm is to minimize the average execution time of multiple applications (avg.JET) as shown in Equation (5). Where J_{comp} is the set of completed job DAGs and $J_{comp} \in J$, $\left|J_{comp}\right|$ is the number of completed job DAGs.

$$\min \mathrm{avg.JET} = \frac{\sum_{j \in J_{comp}} \mathrm{exec}(j)}{\left|J_{comp}\right|} \tag{5}$$

4. Multi-Application Scheduling Algorithm (MASA) Based on DRL

The MASA proposed in this paper mainly addresses two key issues of multi-application scheduling in dynamic environments: (1) scheduling decisions for multiple job DAGs in streaming mode, and (2) neural network training methods in dynamic mission environments.

Task scheduling is essentially a sequential decision optimization problem, as shown in Figure 2. This section represents the multi-application scheduling scenario as a Markov Decision Making (MDP) process [24], when the ready task completes (releasing PEs) or a new application arrives (adding a job DAG), SCHEDULER takes the information of the job in the current job Queue as the input state, outputs the scheduling action, and after the action completes and returns a reward to SCHEDULER to determine whether the action is good or bad. The reward is designed according to the framework's final optimization objective.

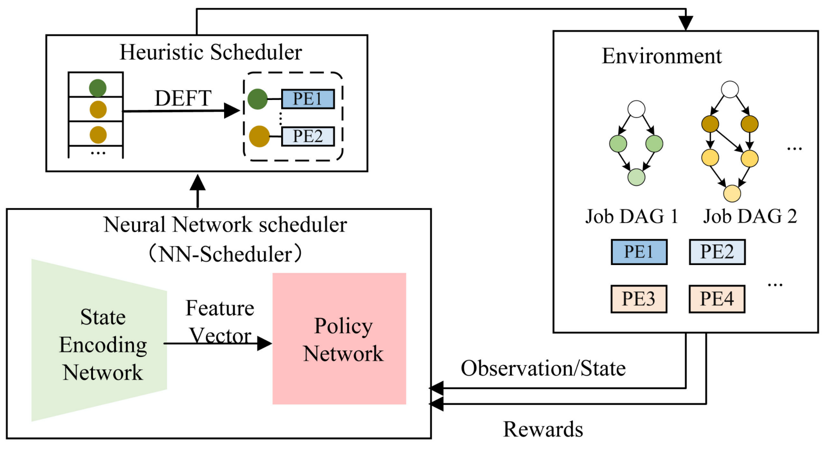


Figure 2. MASA Diagram.

4.1. Framework of MASA

4.1.1. NN-Scheduler

NN-Scheduler solves the multitask prioritization problem, which contains state encoding network and policy network. As shown in Figure 3, the state encoding network is used to encode the job DAG into feature vectors, and the policy network is used to prioritize the tasks.

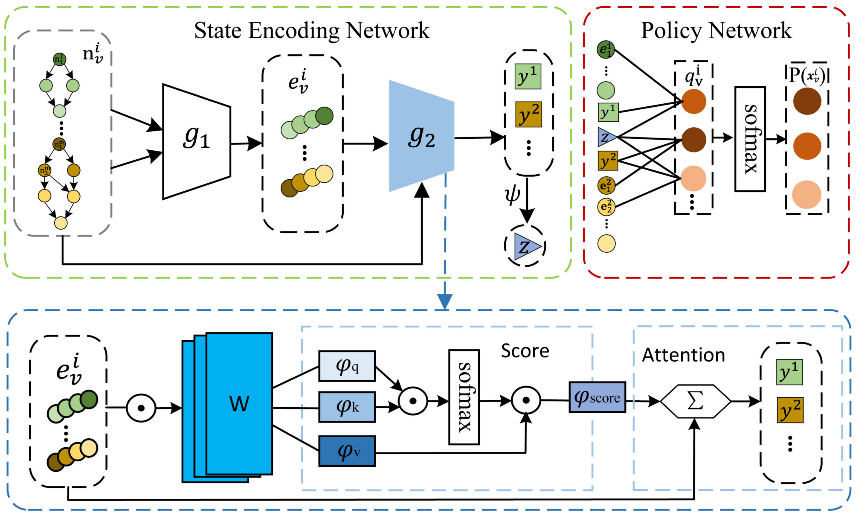


Figure 3. NN-Scheduler Implementation.

1. State encoding network

The state encoding network design references graph neural networks and self-attention mechanisms [10] to achieve scalable network input dimensions. The input states contain the jobs DAG's attributes (e.g., number of remaining tasks, examined task computational execution time, the PE types to which the tasks can be mapped, and DAG dependency structure), and the outputs are feature vectors at the task-level, the job-level, and the global-level, respectively.

The task-level feature vector is performed by the g_1-network, which transforms the attribute information of each job DAG through a nonlinear transformation into a task-level feature vector e_v^i. e_v^i captures the dependencies of the front and back nodes, as shown in Equation (6):

$$e_v^i = g\,[\sum_{u \in sub(n_v^i)} f(u)] + x_v^i \tag{6}$$

where x_v^i represents the attribute information of each node n_v^i and sub (n_v^i) is the set of child nodes of n_v^i. g_1 $(*)$ contains the nonlinear transformation, where $f \sim \log(\cdot/n)$, $g \sim \exp(n \times \cdot)$, $n \to \infty$.

The g_2-network's input vectors, e_v^i, derived from $|J| \times |j|$ space. We adopt the attention mechanism to represent the complex relationship between e_v^i, transforming the raw data into scalable states to capture the feature vectors y^i of the entire job DAG. The job-level feature vector y^i is obtained by summarizing all the feature vectors e_v^i in the DAG, as indicated in the blue box in Figure 3. Equation (7) calculates the weighted vector φ_{score}^i, while Equation (8) represents the attention function that computes the weighted sum. This sum indicates the correlation between the weighted vector φ_{score}^i and the vector sequence $e^i = \left\{e_1^i, e_2^i\ \ldots, e_{|j|}^i\right\}$, where $|j|$ is the number of tasks in job j_i.

$$\varphi_{score}^i = \text{softmax}\left(\frac{\varphi_q \cdot \varphi_k}{\sqrt{d_k}}\right) \cdot \varphi_v = \text{softmax}\left(\frac{\left(w_q \cdot e_v^i\right)^T \left(w_k \cdot e_v^i\right)}{\sqrt{d_k}}\right)\left(w_v \cdot e_v^i\right) \tag{7}$$

$$y^i = \mathrm{att}\left(\varphi^i_{\mathrm{score}}, e^i\right) = \sum_{v=1}^{|j|} \varphi^i_{\mathrm{score}} \cdot e^i_v \tag{8}$$

In order to be able to better represent the connection between all job DAGs, a global feature vector z, $z = \psi\left(y^1, y^2 \ldots y^n\right)$ represents a nonlinear function.

2. Policy network for task prioritization

The prioritization of tasks is achieved using the policy network after obtaining the feature vectors that correspond to the tasks in the job DAG. For each node n in the job DAG, its priority value q^i_v is calculated according to Equation (9), as indicated by the red box in Figure 3.

$$q^i_v = q\left(e^i_v, y^i, z\right) \tag{9}$$

It can be seen from the formula that the q^i_v of each node in each DAG is related to three elements, i.e., the task-level vector e^i_v containing the attribute information and dependencies of the node itself; the information y^i contained in the DAG where each node is located; and all the global vectors z representing the connections between job DAGs.

Tasks n^i_v with high priority values q^i_v in the state space are transferred to the set of ready tasks $\mathcal{A}_t$. Each episode updates $\mathcal{A}_t$. Subsequently, the sofmax function is used to calculate the mapping probability $P\left(n^i_v \in \mathcal{A}_t\right)$ for each ready task in $\mathcal{A}_t$, as shown in Equation (10).

$$P(n^i_v \in \mathcal{A}_t) = \frac{\exp\left(q^i_v\right)}{\sum_{n^i_v \in \mathcal{A}_t} \exp\left(q^i_v\right)} \tag{10}$$

4.1.2. Heuristic Scheduler

Because it involves the selection of PE type, the heterogeneous computing environment makes the assignment of PEs more difficult than the problem of assigning PEs in a homogeneous environment. Therefore, after obtaining the mapping probabilities corresponding to high-priority tasks, the task-mapping problem in the second scheduling phase is realized based on a heuristic algorithm. There are some implicit constraints in the resource allocation process, such as the sum of the computational requirements of multiple tasks deployed on a PE must not exceed the upper limit of this PE's computational resources, and the sum of the task data volume overlays must not exceed the physical bandwidth between the PEs.

This paper aims to improve execution efficiency in a continuous dynamic environment through the inclusion of a heuristic algorithm that duplicates critical parent nodes, reducing application execution time. The mapping scheme of the selected node in this study is based on the DEFT algorithm [25]. Comparison of the computational results of the EFT algorithm and the Copy Parent Earliest Finish Time (CPEFT) algorithm helps select the smallest result as the most optimal value. Equations (11) and (12) detail how the CPEFT algorithm tries to replicate each parent node of the selected node in order to compare and arrive at the earliest possible completion time. Equation (13) explains how the DEFT algorithm selects the minimum result of the EFT and CPEFT algorithms, regardless of whether it involves duplicated parent nodes or not. This ensures that computations are performed with the minimum result.

$$\mathrm{CPEST}(\mathrm{perd}(n_i), n_i, p_m) = \min_{n_j \in \mathrm{pred}(n_i)}\left(\mathrm{AFT}\left(n_j\right) + \mathrm{comm}\left(n_j, n_i\right)\right) \tag{11}$$

$$\mathrm{CPEFT}(n_i, p_m) = \max_{n_a, n_b \in \mathrm{perd}(n_i), a \neq b}\{\mathrm{CPEST}(n_a, n_i, p_m), \mathrm{CPEST}(n_b, n_i, p_m)\} + \mathrm{comp}(n_i, p_m) \tag{12}$$

$$\mathrm{DEFT}(n_i) = \min\{\mathrm{CPEFT}(n_i, p_m), \mathrm{EFT}(n_i, p_m)\} \tag{13}$$

4.1.3. Description of the Empirical Data

The NN-Scheduler obtains state information s_t and executes action $\{a_{n,t}\}_{n=1}^{n\prime}$ at each interaction time step t. This paper differs from standard MDP processes in that the moment when the environment state changes is uncertain. The policy network generates multiple task decisions in time step t. When all the actions in time slot t (time step [t, t + 1)) are completed, rewards are generated and the environment state is updated. The environment state changes if a new job DAG is reached during action execution.

1. State space

The NN-Scheduler receives an input state represented by $S = [s_1, s_2, \ldots, s_o]$. s_n represents the state of the nth job, and o is the number of concurrent jobs in the job queue at the current time, where o is a variable and its value cannot exceed the maximum job queue capacity C. Concurrent jobs include previously unfinished jobs and newly arrived jobs, and the NN-Scheduler can learn about the potential interference between concurrent jobs through training. The state s_n of the nth job can be represented by the following equation:

$$s_n = \left(\left(\overline{p_v}, \overline{EET_v}, \overline{status_v} \right)_{v=1}^{|j|}, |OT|, |RT|, JWT \right) \tag{14}$$

The task n_v's assignable PE number is p_v; EET_v is the estimated execution time corresponding to the task; $status_v$ is the current state of the task, classified by the labels ready, completed, or outstanding; $|j|$ is the number of task nodes in job j. $|OT|$ refers to the number of outstanding tasks that are waiting for the parent node to finish its execution. Similarly, $|RT|$ refers to the number of ready tasks in the set of ready tasks $\mathcal{A}_t$, and JWT is the time that job j_n has been running from the time it arrived on the system to the time it is currently running.

2. Action space

At each time step, the SCHEDULER observes the state s_t of the environment and performs the number of n′ actions, $\{a_{n,t} \sim \pi_\theta(a|s_t)\}_{n=1}^{n\prime}$. The action space is shown in the red box in Figure 3, and the NN-Scheduler outputs the set of selected high-priority tasks n_v^i.

3. Reward

The correct reward is computed according to the RDA method (Section 4.2.1), taking into account the inconsistency between the time step of the scheduler's interaction with the environment and the time step of the environment's interaction with the scheduler. r_t is computed by constructing the reward under each clock signal as r'_{clk}, and then calculating r_t from it. r'_{clk} is given by Equation (15), where $|n_{comp}|$ is the number of newly completed tasks under each clock.

$$r'_{clk} = |n_{comp}| \tag{15}$$

4.2. Network Training

This paper addresses the problem of training neural networks based on the policy gradient algorithm, Actor Critic (AC) [26,27]. AC evaluates the merits of an action by using a dominance function, and if the action value function Q (S, A) is better than the state value function V (S), the corresponding action is improved. At each time step t, the actor network observes the environment state s_t and chooses to execute action $\{a_{n,t}\}_{n=1}^{n\prime} \sim \pi_{\theta,n}(s_t)$. n′ denotes the number of tasks with the same priority.

The Actor network's loss function is

$$Loss_{actor} = \sum_{t=0}^{T} \log \pi_\theta(a_{n,t}|s_t) \left(Q_{\pi_\theta}(s_t, a_{n,t}) - V_{\pi_\theta}(s_t) \right) \tag{16}$$

The Critic network assesses the state value function $V_{\pi_\theta}(s_t)$ and computes its loss function as shown in Equation (17).

$$\mathrm{Loss_{critic}} = \frac{1}{2}\left(Q_{\pi_\theta}(s_t, a_{n,t}) - V_{\pi_\theta}(s_t)\right)^2 \tag{17}$$

$$\nabla_\theta J(\theta) \triangleq \nabla_\theta[\mathrm{Loss_{actor}} + \mathrm{Loss_{critic}} + \eta \cdot E_S[H(\pi_\theta(\cdot|s_t))]] \tag{18}$$

The training policy learns the optimal strategy using the regular entropy to make the output probability distribution of actions more uniform and to prevent convergence to a single output action. This paper considers introducing entropy regularity $\eta \cdot E_S[H(\pi_\theta(\cdot|s_t))]$ into the loss function, represented with entropy $H(\cdot)$ and weight η. In summary, the gradient strategy is as follows.

To achieve stable and fast training convergence, this paper also employs the following methods to optimize the training process: (1) Reward Dynamic Alignment (RDA) is used to address the problem of incorrect reward propagation. This problem arises due to inconsistent interaction time steps between SCHEDULER and the environment. (2) Gradually increasing the length of episodes can improve the initial training phase's inference ability for neural networks and help address the challenge of making multiple job DAG scheduling decisions. (3) Asynchronous multi-agent joint training is another method used to speed up the neural network's training process.

Algorithm 2 shows the pseudocode of the network training process in MASA. Throughout the training process, asynchronous multi-agent joint training is performed. In order to avoid wasting training time, the initial events are terminated early, and Line 4 samples the event length T from an exponential distribution with a small initial expectation T_{mean}. (See Training Optimization Method 2 for more information). In order to avoid errors caused by randomness during job arrivals and inconsistent task execution times, Line 11 calculates the correct time-step reward based on the clock reward (see Training Optimization Method 1). Additionally, Line 22 implements the increment of the average length of episodes T_{mean}. Lastly, the strategy parameter θ of MASA is updated in Line 23.

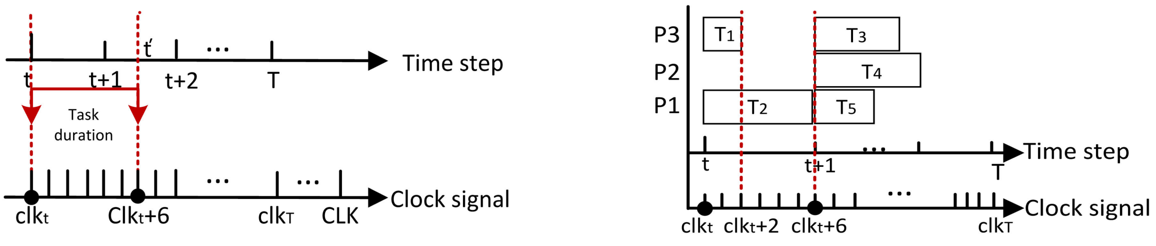
4.2.1. Reward Dynamic Alignment Method

The standard MDP process is suitable for round-based decision problems, where the change in the state of the environment occurs after the agent has executed an action [28]. In heterogeneous resource systems, external factors such as the random arrival of jobs and internal factors such as the different completion times of multiple actions at the same timestep can cause the state of the environment to change while the SCHEDULER is executing an action. This paper proposes the RDA method as a solution to incorrect reward propagation during finite MDP.

1. incorrect rewards

Scenario 1. *SCHEDULER can achieve real-time task scheduling by observing the real-time environment state and taking immediate actions while making decisions in the current time step. For instance, as depicted on the Figure 4a, when the action $a_{n,t}$ ends at time step, denoted by t', the immediate reward for the action is also granted at t'. While the action $a_{i,t}$ is being executed, the random arrival of a job can modify the priority relation between tasks, and SCHEDULER observes S_{t+1} when $t' \in (t+1, t+2]$. This circumstance leads to the incidence of the incorrect reward for the action at time $t+1$.*

Scenario 2. *Scheduling decisions of tasks with the same priority are formed at the same timestep. However, due to the differences in the performance of heterogeneous PE as well as the task requirements, each decision process completes at a different time, resulting in an inconsistency between the immediate rewards and the observed rewards, a situation that also leads to the occurrence of incorrect rewards. For example, as shown on the Figure 4b, although task T_1 has been accomplished* *earlier than expected, the scheduling policies for the subsequent tasks T_3, T_4 and T_5 are applied after the completion of task T_2. As a result, there is a delay in the reward for task T_1.*



(**a**) (**b**)

Figure 4. (**a**) Incorrect reward lead by external factors; (**b**) Incorrect reward lead by internal factors.

Algorithm 2. Training Algorithm

```
1.  //Assume global shared parameter vector $\theta_1$ and $\theta_2$, one-thread parameter $\theta'_1$ and $\theta'_2$,
    globally shared iteration counter M, globally shared iteration max.counter $M_{max}$, maximum
    episode length mean $T_{mean}$, decay factor $\gamma$,
2.  Initial thread time step counter $t \leftarrow 1$
3.  repeat
4.      sample episode length $T$
5.      Reset actor/critic network's policy gradient: $\theta_1 \leftarrow 0$ and $\theta_2 \leftarrow 0$
6.      Synchronize global parameter to one-thread parameters $\theta'_1 = \theta_1$ and $\theta'_2 = \theta_2$
7.      $t_{\mathrm{start}} = t$, get state $s_t$
8.      repeat
9.          Perform $a_t$ according to policy $\pi_{\theta'_1}(a_t|s_t)$
10.         Assign PE for task
11.         Calculate the reward by clock's reward according to Equation (19)
12.         Receive reward $r_t$ and new state $s_{t+1}$
13.             $t \leftarrow t + 1$
14.             $M \leftarrow M + 1$
15.     until Job Queue is empty (terminal $s_t$) or $t - t_{\mathrm{start}} == T$
16.     $Q(s,t) = \begin{cases} 0 & for\ terminal \\ V_{\pi_{\theta'_2}}(s_t) & for\ non-terminal\ s_t \end{cases}$
17.     for $j \in \{t-1, \ldots, t_{\mathrm{start}}\}$ do
18.         $Q(s,j) \leftarrow r_j + \gamma Q(s,j+1)$
19.         Accumulate actor's local gradient updates:
                $d\theta_1 \leftarrow d\theta_1 + \nabla_{\theta'_1} log\pi_{\theta'_1}\left(a_j, s_j\right)\left(Q(s,j) - V_{\pi_{\theta'_2}}\left(s_j\right)\right) + \eta\nabla_{\theta'_1} H\left(\pi_{\theta'_1}\left(\cdot\middle|s_j\right)\right)$
20.         Accumulate critic's local gradient updates: $d\theta_2 \leftarrow d\theta_2 + \nabla_{\theta'_2}\frac{1}{2}\left(Q(s,j) - V_{\pi_{\theta'_2}}\left(s_j\right)\right)^2$
21.     end for
22.     $T_{mean} \leftarrow T_{mean} + \varepsilon$
23.     Update global network parameters $\theta_1 = \theta_1 + \alpha d\theta_1$ and $\theta_2 = \theta_2 + \beta d\theta_2$
24. Until $M > M_{max}$
25. Output the global network parameters $\theta_1$ and $\theta_2$
```

2. Implementation of RDA

The RL process of an MDP can be summarized as a $\left\{s_t, \{a_{n,t}\}_{n=1}^{n'}, \{r_{n,t}\}_{n=1}^{n'}\right\}_{t=1}^{T}$ sequence. The reward r_t corresponding to the previous decision affects the SCHEDULER's action decision a_{t+1} at time step t + 1, so how to obtain the correct r_t. The essential reason for generating incorrect rewards is the inconsistency of the time step at which the SCHEDULER interacts with the environment. Therefore, this paper proposes to compute the reward r'_{clk} on each clock signal so that the computation of r'_{clk} is independent of the interaction time step. The time-step reward r_t consists of the clock reward r'_{clk}, i.e., the value of r_t consists of a superposition of $\{r'_{clk}\}$. The RDA method is specifically described below.

- Step 1: Decouple the empirical data associated with the time step to obtain the sequence $\{s_t, a_t\}_{t=1}^{T}$ as well as $\{r_t\}_{t=1}^{T}$. T is the last interaction time step in the episodes. To obtain the sequences $\{r'_{clk}\}_{clk=1}^{CLK}$ and $\left\{\sum_{clk=1}^{clk_{now}} r'_{clk}\right\}_{clk_{now}=1}^{CLK}$, compute the r'_{clk} on each clock signal and perform the $\sum_{clk=1}^{clk_{now}} r'_{clk}$ computation independently of the interaction timestep. Here, CLK refers to the clock length of the simulation.
- Step 2: At the beginning of the action, memorize the clock signal clk_{start}, and at the end, memorize the clock signal clk_{end}. Use Equation (19) to calculate r_t.

$$r_t = \sum_{clk=1}^{clk_{end}} r'_{clk,t} - \sum_{clk=1}^{clk_{start}} r'_{clk,t} = \sum_{clk_{start}}^{clk_{end}} r'_{clk,t} \tag{19}$$

In summary, the RL process based on the RDA method can be summarized in Equation (20).

$$\left\{s_t, \{a_{n,t}\}_{n=1}^{n'}, \{r_{n,t}\}_{n=1}^{n'}\right\}_{t=1}^{T} \to \left\{s_t, \{a_{n,t}\}_{n=1}^{n'}, \left\{\sum_{clk_{start}}^{clk_{end}} r'_{clk,n,t}\right\}_{n=1}^{n'}\right\}_{t=1}^{T} \tag{20}$$

In this way, the correct empirical data can be obtained using the RDA method during the interaction between SCHEDULER and the environment, no matter how dynamically the set of actions is changing.

4.2.2. Early Termination of the Initial Episode Method

During the initial phase of neural network training, network parameters are randomly set, resulting in a weak initial scheduling decision for SCHEDULER. As jobs randomly and successively arrive during the training process, the initial job DAG may not receive a reasonable scheduling decision. This may result in a negative impact on the final job execution time, and this process is irreversible. Furthermore, in most of the initial episodes, the outstanding queue may accumulate a large number of waiting tasks. If too much empirical data are sampled during these episodes, it will lead to increased duration of network training. To prevent such a situation, this paper limits the initial episode length and increases it gradually during the training process. The memoryless termination process is employed for achieving incrementally longer episodes. Timestep T is randomly sampled from an exponential distribution, causing the episode length T to obey an exponential distribution as well $T \sim exponential1/T_{mean}$.

4.2.3. Asynchronous Multi-Agent Joint Training Method

Training RL models using continuous sampling is difficult to converge due to the correlation between empirical data. Using empirical replay, the correlation of the data can be broken. However, there is another asynchronous multi-agent joint training method that can break the correlation of empirical data by creating multiple Agents [29,30], starting multiple training environments for sampling and using the collected empirical data for training, accumulating the gradients obtained from training in multiple processes and updating the shared parameters together after a certain number of steps. Compared to the experience playback method that requires an experience pool to store historical samples and randomly selects training samples, independent and identically distributed samples produced with asynchronous training not only reduce storage space but also greatly accelerate the sampling speed and consequently improve the training speed. Additionally, different strategies used in distinct training environments create a more uniform distribution of empirical data, facilitating the stable training of neural networks.

5. Experiments

This section aims to verify the performance of our proposed MASA algorithm. Firstly, we compare the performance of the proposed algorithm with that of current resource-scheduling algorithms. Next, ablation experiments are conducted to verify the effectiveness of the proposed training optimization method. To demonstrate the scalability of the proposed algorithm for complex mission environments, the experiments simulate the task-scheduling problem in streaming mode using six real-world RF applications in the field of wireless communication and radar processing.

5.1. Experimental Environment Setting

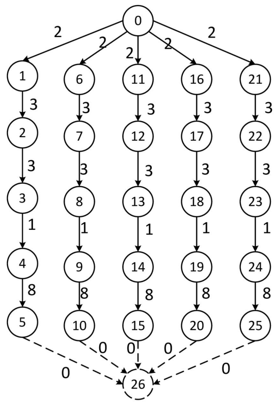
During simulation execution, a job randomly arrives every $\mathrm{clk_{inj}}$ clock signal in the job queue to simulate the dynamic task environment. $\mathrm{clk_{inj}}$'s expectations are scale, $scale = \mathrm{E}\left(\mathrm{clk_{inj}}\right)$. The inter-arrival time of the $\mathrm{clk_{inj}}$ follows the exponential distribution, $\mathrm{clk_{inj}} \sim \mathrm{exponential}\left(\frac{1}{\mathrm{scale}}\right)$.

Assuming the multifunction RF system has the ability to handle different RF applications, real-world application configurations are more complex than classic application configurations. Table 1 shows the classic DAG comprising of 10 randomly generated tasks. WiFi-TX, WiFi-RX, Single-carrier Tx (SCT), Single-carrier Rx (SCR), Range Detection, and Temporal Mitigation are the six common RF applications. Their corresponding hardware resource profiles are obtained from engineering data. Figure 5a displays the respective DAGs for WiFi-TX [31]. Here, Inverse Fast Fourier Transform (FFT) tasks are computationally intensive, taking 10 times longer to process on a CPU than on a HW accelerator. Figure 5b shows the computational execution time of each task on the corresponding PE that we obtained from statistics in the actual project; in Figure 5a, the value inside the circle represents the task ID, and the value beside the connecting line represents the data volume between the WiFi-Tx tasks. The communication bandwidth between PEs in the hardware configuration file is set as follows: the bandwidth value between HW Acc2 and the other PEs (cpu A15, cpu A15 and HW Acc1) is set to 10, and the bandwidth value be-tween all other PEs is set to 1000. The data volume between tasks and the communication bandwidth between PEs are known, and the communication delay can be derived based on Equation (2). Based on the above profiles in the simulation platform, MASA determines which PE to assign to the current highest priority task in the task-mapping phase based on the constraints of equations such as EST and EFT. Network structure and training parameters are listed in Table 2.

These applications show different workload characteristics due to differences in DAG structure and computational requirements. For instance, WiFi-TX demonstrates a considerable level of parallelism compared to SCT and SCR, which contain sequential links among nodes, thus consuming less energy during low bandwidth conditions.

Table 1. Benchmark of applications.

Application Type	Application	Number of Tasks	Max Width, Depth
Real-world' Communication Application	WiFi-TX	27	5, 7
	WiFi-RX	34	5, 10
	Single-carrier Tx	10	1, 10
	Single-carrier Rx	10	1, 10
Real-world' Radar Application	range detection	7	2, 6
	Temporal Mitigation	10	2, 6
Classic DAG		10	random



(**a**)

PE type / (Task ID) Functional Components	execution time (us)				
	cpu A15	**cpu A7**	**HW Acc1**	**HW Acc2**	**HW Acc3**
Scramble Encode (0)	10	22	8	-	-
Interleaves (1,6,11,16,21)	4	10	-	-	-
QPSK Modulation (2,7,12,17,22)	8	15	-	-	-
Pilot insertion (3,8,13,18,23)	3	5	-	-	-
Inverse-FFT (4,9,14,19,24)	118	150	55	16	50
CRC (5,10,15,20,25)	3	5	-	-	-

(**b**)

Figure 5. (**a**) WIFI-Tx′ DAG; (**b**) WIFI-Tx task execution time on different PEs profiles.

Table 2. Algorithm parameter setting.

Parameter	Value	Parameter	Value
Job queue capacity	3	Optimizer	Adam
Node input dimensions	6	Scale (job inject interval′ expectation)	20/40/60/80/100
Output dimensions	8	CLK (simulation length)	10,000
Maximum depth	3	T_{mean} (episode length′ expectation)	10,000
Hidden dimensions	[16, 8]	M_{max} (number of Agent)	1/2/4/6/8/10
β (learning rate)	0.0001	η (entropy coefficient)	0.1
α (learning rate)	0.0003	y (discount factor)	0.98

For the purpose of comparison, two types of representative scheduling algorithms have been considered: The heuristic algorithms MET [32] and HEFT [17], and the neural scheduling algorithms SCARL [33], decima [23], and SoCRATES [22]. This paper incorporates the mainstream homogeneous resource-scheduling algorithm decima into the comparative experiments to emphasize the necessity of designing heterogeneous scheduling algorithms. Decima is dedicated to resource scheduling in homogeneous environments, and SCARL only supports DAG for link structure, so its configuration must be modified in simulation slightly, and each experiment is repeated five times.

5.2. Algorithm Performance Analysis

We present the performance results of the neural network algorithm and heuristic algorithm for different application configurations in Figures 6–8. The scheduling algorithm is employed to schedule and execute both classical and real-world applications' DAGs. We plot the trend of the average application execution time with the job DAG arrival rate. The horizontal axis depicts different scale values, the smaller the scale, the faster the application arrival rate; the vertical axis depicts the average application execution time.

5.2.1. MASA in Classic DAG Environment

In Figure 6, we show the performance results of different scheduling algorithms under the classical DAG environment configuration. We also plot the algorithms' performance results under different numbers of applications, respectively, as in Figure 6a–f. As the arrival rate of applications increases, the average application execution time of each algorithm gradually increases. However, the performance of the MASA algorithm proposed in this paper consistently outperforms the other algorithms. The graph shown in Figure 6a represents the execution results of a classical DAG with successive arrivals to the scheduler, under the condition of following an exponential distribution. To comprehend the performance of the scheduling algorithm, an analysis is conducted on the average application execution time at different scales. At a low arrival rate (scale = 100), the heuristic-based algorithm's scheduling performance is clearly weaker than the neural network-based algorithm's scheduling performance. When the arrival rate increases (scale = 20), the MASA algorithm proposed in this paper also outperforms the other neural scheduling algorithms, with an improvement of about 9% over the performance of SCRAL and about 18% over the performance of decima and SoCRATES. Figure 6f illustrates an application pool of 10 classical DAGs, each with the same arrival probability. As shown by the figure, the algorithm presented in this paper offers superior performance compared to other algorithms in all application arrival rates, while also improving with an increase in the number of applications.

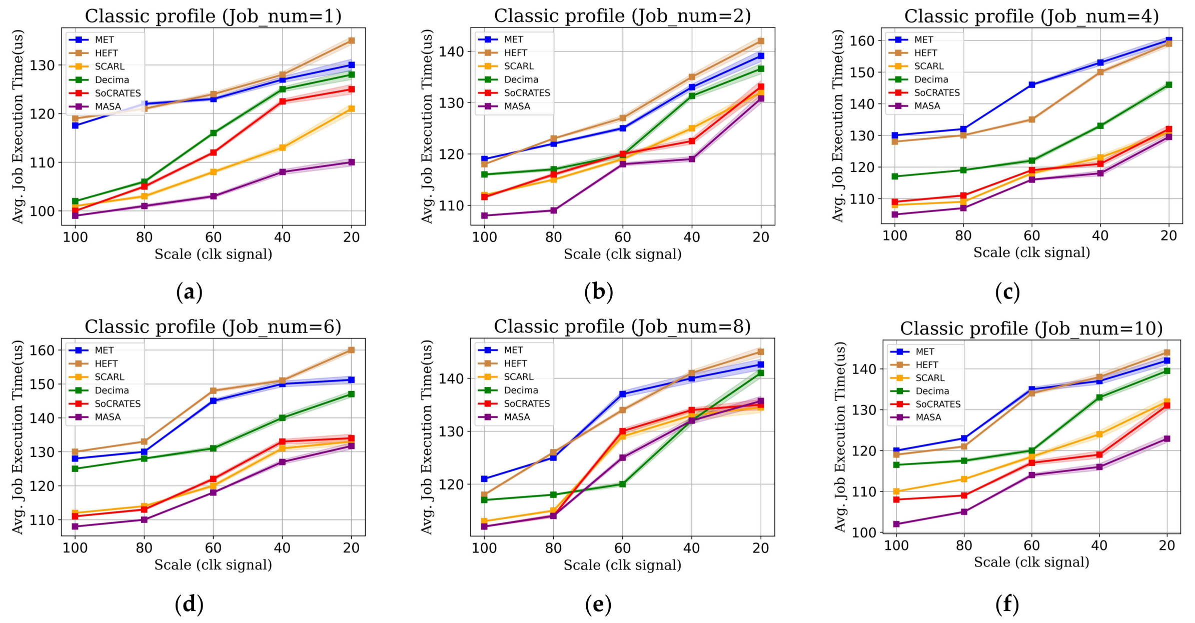


Figure 6. (**a**) Results of scheduling algorithms in different numbers of classical DAGs: (**a**) the DAG pool consists of one classical DAG; (**b**) the DAG pool consists of 2 random classical DAGs; (**c**) 4; (**d**) 6; (**e**) 8; (**f**) 10.

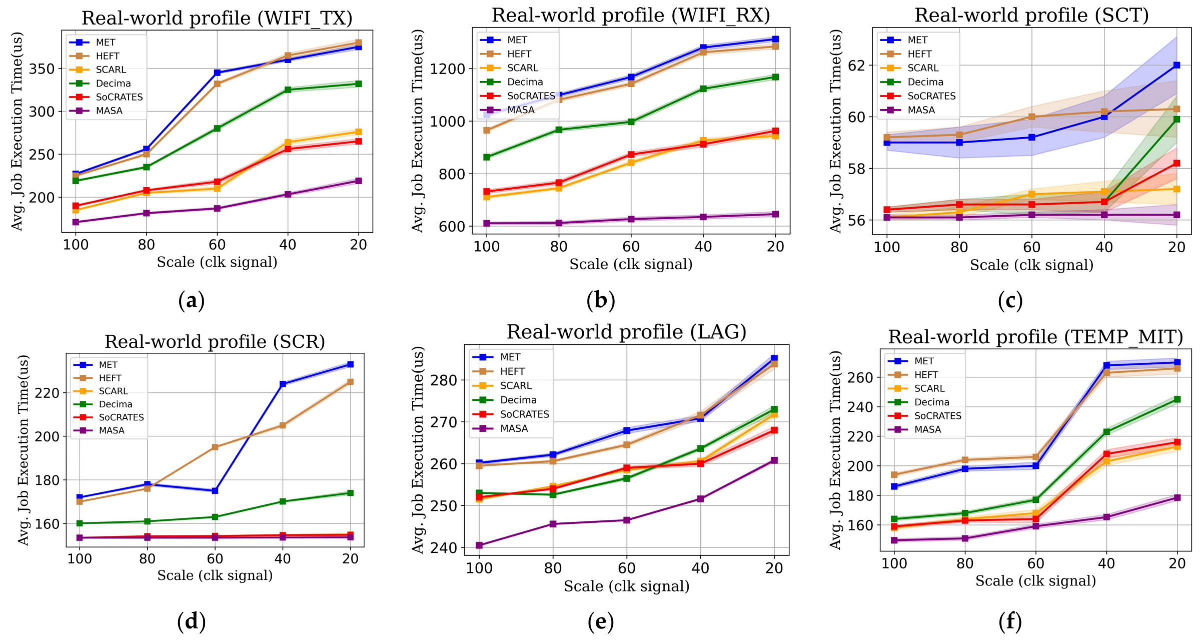


Figure 7. Scheduling results of scheduling algorithms for different real RF applications (**a**) WiFi-TX; (**b**) WiFi-RX; (**c**) Single-carrier Tx; (**d**) Single-carrier Rx; (**e**) Ranging detection; (**f**) Temporal mitigation.

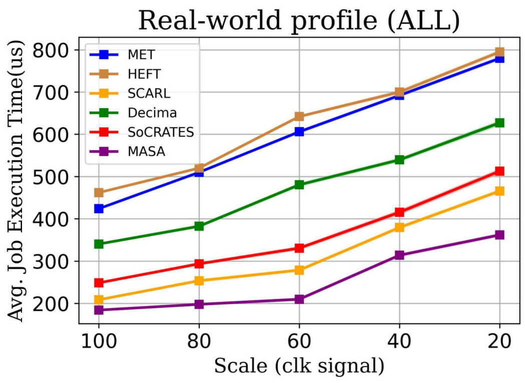


Figure 8. Algorithm performance results in mixed real-world environments (consisting of 6 real-world applications with the same probability).

The data presented in Figure 6 show that the heuristic algorithm can yield results comparable to other algorithms when the number of tasks is small and the state environment is relatively uncomplicated. However, as the task injection rate increases, the complexity of the state space faced by the scheduler based on the heuristic algorithm increases dramatically, and the heuristic algorithm is unable to solve the situation very well. Decima is based on a graph neural network for encoding the state space. Compared to the heuristic algorithm, it can provide a more detailed explanation of the complex state space, leading to significantly improved performance, especially when dealing with large task injection rate, but it should be noted that Decima is mainly proposed to solve the isomorphic resource-scheduling problem. Based on this foundation, SCARL and SoCRATES introduce the fundamental concept of RL to improve the results when the model is trained on the dynamically transformed heterogeneous resource platform through the interaction between the environment and the Agent, and the experimental results also prove the effectiveness of RL in the problem setting. To further encode the complex and dynamic state space of the heterogeneous resource platform, our proposed MASA introduces the mechanism of attention, which further improves its encoding ability. The final outcomes demonstrate that the attention mechanism can significantly enhance the encoding capacity of the network model as the task volume increases and the encoding environment becomes more complex.

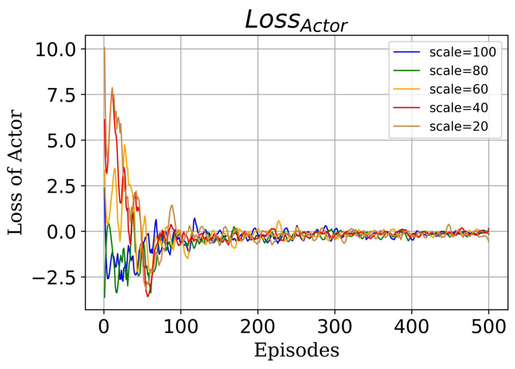
5.2.2. MASA in Real-World Environment

To further illustrate the superiority of our proposed algorithms, we test the performance of each algorithm in a job profile in a real mission scenario, and the experimental results are shown in Figures 7a–f and 8. The configuration for WiFi-RX job is the most complex, consisting of 34 tasks with complicated dependencies. However, its application execution time is approximately three times longer than that of the latter.

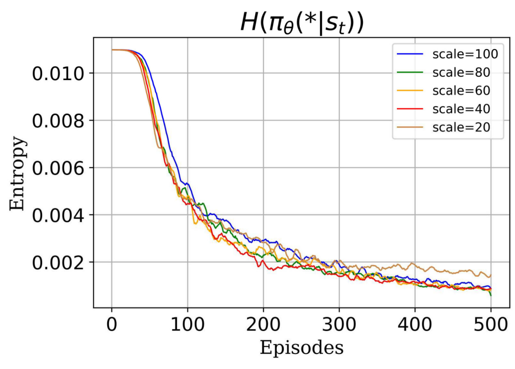
The WiFi-TX job includes 27 tasks, which is more than a typical task configuration environment. However, its execution time does not increase much compared to a classical environment because the dependencies between its tasks are mostly linear, and it can be observed that MASA outperforms the other compared algorithms in every application arrival rate (scale). The configuration of WiFi-RX is the most complicated, which contains the largest number of tasks and complex dependencies between tasks, so the execution time of this task is much larger than all the other application configurations in this paper. It can be seen that the algorithm proposed in this paper, MASA, has a significant improvement over the other heuristics and neural scheduling algorithms, which is mainly due to the fact that MASA adopts the attention mechanism, which can effectively analyze the topology between complex tasks to improve the performance of the neural scheduling network. The SCT task configuration includes only 8 tasks and straightforward dependencies, so its execution time is relatively low, and when the task injection rate is low, there is not much difference in the performance of the neural scheduling algorithms (better than the heuristic algorithms), but when the task injection rate reaches the maximum, the performance of the neural scheduling algorithms is better than the heuristic algorithms. Nevertheless, when the task injection rate is maximum, MASA's algorithm surpasses the performance of other comparable algorithms. The task environment configuration of SCR is similar to that of SCR, except it demands a higher data volume between tasks. According to the experimental results, both SoCRATES and MASA's scheduling schemes yield the lowest task execution time, and MASA slightly outperforms SoCRATES. The configurations of the range detection and Temporal Mitigation environments show that MASA performs better than the other algorithms at all task arrival rates. In conclusion, the study demonstrates that the simulation environment is scalable by validating the proposed algorithm under different real profiles.

This paper includes a simulation of six applications (WiFi-TX/RX, SCT, SCR, range detection, and Temporal Mitigation) concurrently running in a real environment to analyze algorithm performance under varying task arrival rates. Figure 8 shows that the heuristic algorithm does not dominate the overall task scheduling due to the extended execution time of the WiFi-TX. The complexity of the task environment results in higher demands on the performance of the scheduling algorithms as the task injection rate increases, and the MASA algorithm gradually demonstrates its dynamic scheduling capability in complex task environments. The real-world job profiles are more complex, so the scheduler's ability to encode the state space puts forward higher requirements, and its final average task execution time depends largely on each scheduler's ability to interpret the state space. MASA's excellent dynamic scheduling ability is due to the fact that the introduced attention mechanism, which eliminates the need to adjust the *scale* of the network model based on the specific task environment and provides a better interpretation of the dynamic environment.

Figure 9a illustrates the convergence process of actor network during training. It can be observed that the actor–critic strategy results in convergence around 100 episodes. The fast convergence of $\mathrm{LOSS}_{\mathrm{Actor}}$ in MASA proves its stronger learning ability in dynamic environments.



(**a**)



(**b**)

Figure 9. (**a**) Analysis of Actor convergence results; (**b**) Entropy curves for the task environment.

Figure 9b presents the change of entropy $H(\pi_\theta(\cdot|s_t))$ in Equation (18), describing the task environment faced by the scheduler. As training progresses, the MASA network effectively resolves the job DAGs, leading to a gradual reduction of uncertainty in the environment. Lower values of $H(\pi_\theta(\cdot|s_t))$ are indicative of the fact that MASA effectively encodes the complex environment so that its complex state space can be mapped into a relatively stable encoded vector space.

5.3. Ablation Experiment

This paper proposes several training optimization methods to stabilize neural network training in dynamic environments and improve its convergence speed. This section analyzes the utility of these methods. The effect of training optimization methods on scheduling performance is observed in both large-scale and small-scale complex environments. The simulation parameter settings in this section are consistent with those in Figure 6a.

5.3.1. Reward Dynamic Alignment Analysis

In Figure 10, we plot the convergence of the training results of the MASA algorithm with and without the RDA optimization method during training at different application arrival rates. The introduction of the RDA method makes it clear that the network converges to a stable state earlier and the final training results in better performance. The RDA optimization method maintains the average application execution time in a more stable state, even with changes in the arrival rate of applications. RDA solves the reward inaccuracy problem by buffering the rewards under clock signals, thus improving the performance of network training. RDA is a lightweight optimization method that can be directly applied to similar application scenarios, as it only memorizes the decision start and end times of the task in comparison to the traditional RL method.

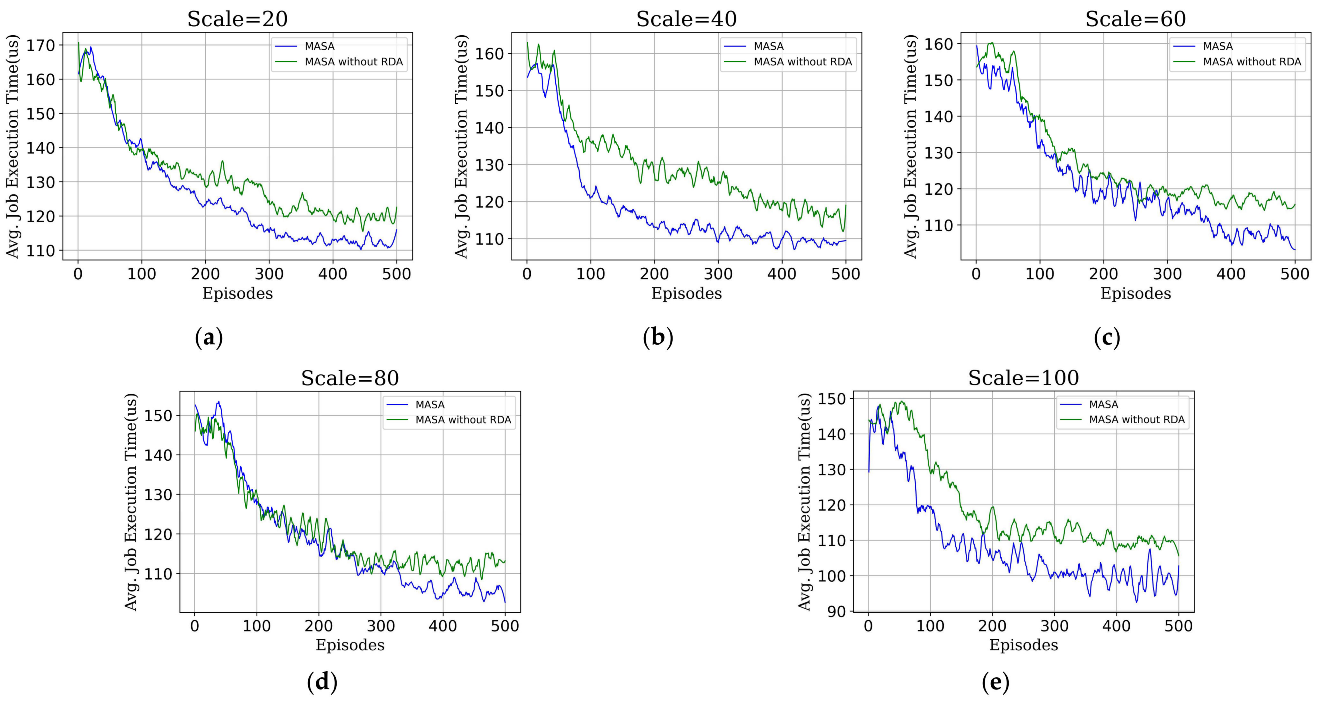


Figure 10. RDA validity analysis at different application scales: (**a**) scale = 20; (**b**) scale = 40; (**c**) scale = 60; (**d**) scale = 80; (**e**) scale = 100.

5.3.2. Asynchronous Multi-Agent Joint Training Analysis

The direct training of neural networks using a single agent is unstable and has poor performance, as the correlation of the empirical data is high. To improve the convergence speed of network training, we consider the method of asynchronous multi-agent joint training in this paper. Figure 11 shows the effect of different numbers of agents on network training. When the number of agents is one, no multi-asynchronous action joint training is used. The experimental results indicate that asynchronous multi-agent joint training can effectively improve the convergence speed of network training. The greater the number of agents, the shorter the time for the network to converge to a stable state. When the number of agents is 10, the training curve of the network is the smoothest and can converge to a stable value in a shorter period of time, resulting in optimal performance of the network.

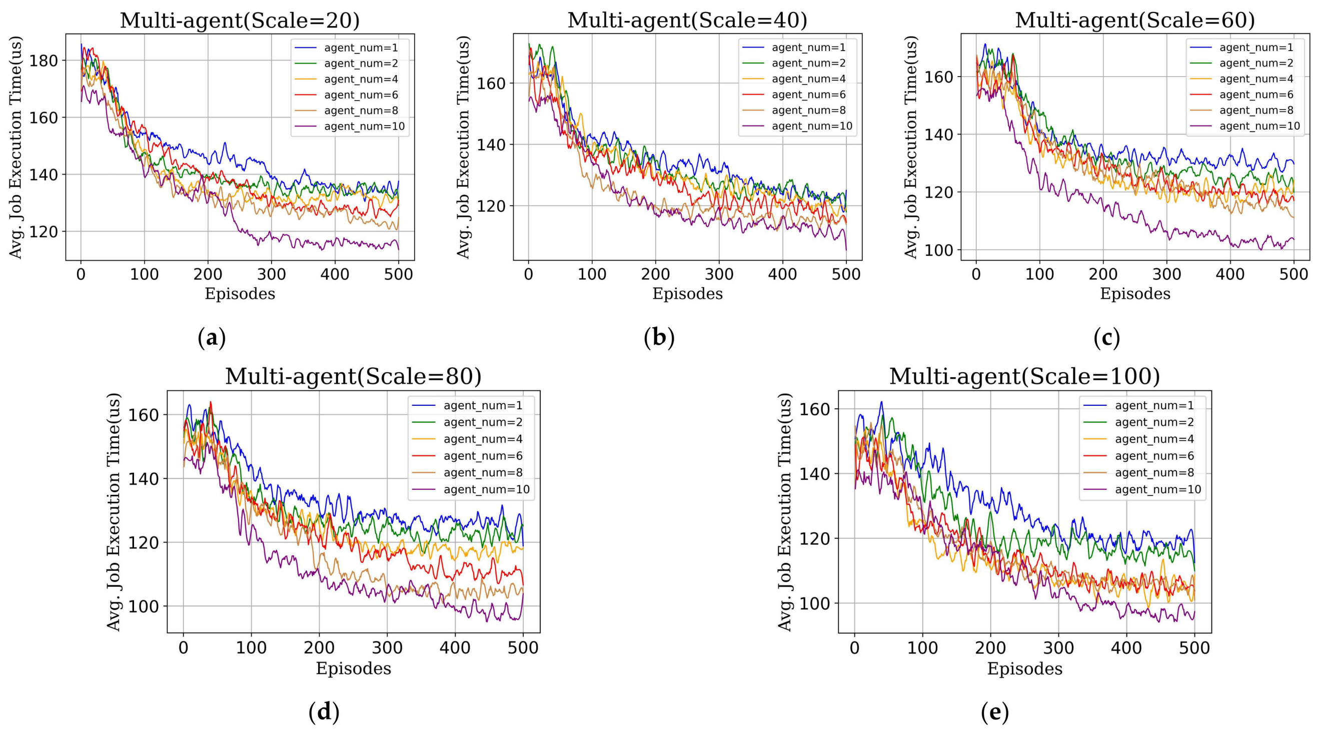


Figure 11. Analyzing the effectiveness of asynchronous multi-agent under different application scales: (**a**) scale = 20; (**b**) scale = 40; (**c**) scale = 60; (**d**) scale = 80; (**e**) scale = 100.

6. Conclusions

This paper proposes a neural scheduling algorithm, called MASA, to address the heterogeneous resource-scheduling problem in dynamic environments, and the attention mechanism is adopted to represent complex relations between job DAGs. To train networks fast and stable in a dynamic environment, this paper proposes the RDA, early termination of initial episodes, and asynchronous multi-agent joint training methods. Experimental results demonstrate that the MASA algorithm proposed in this paper outperforms both neural scheduler algorithms and heuristic algorithms in task configurations under hypothetical and real applications. Additionally, ablation experiments confirm the effectiveness of the RDA algorithm and the asynchronous multi-agent training strategy proposed in this paper in enhancing the ability of the network to be trained in dynamic environments. In the current simulation environment, in order to enhance the high fidelity of the heterogeneous scheduling algorithms, we have included the inter-PE communication overhead into the application execution time overhead. However, considering that the communication bandwidth resources between processing elements are relatively abundant compared to the volume of intertask data transmission, we have not scrutinized the magnitude of the communication transmission overhead on the total execution time in our current work, and we will further explore this issue.

Future work will further explore the application of MASA algorithms in large-scale task scheduling and test their reliability in more radio signal processing. It is also crucial to verify whether the scheduling algorithms proposed in this paper have high fidelity and high reliability.

Author Contributions: Methods and data collation, formal analysis and validation and writing (original manuscript preparation), Q.P.; writing (review and editing), funding acquisition and conceptualization, S.W. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Data is unavailable due to privacy.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Venkataramani, A.; Chiriyath, A.R.; Dutta, A.; Herschfelt, A.; Bliss, D.W. The DASH SoC: Enabling the Next Generation of Multi-Function RF Systems. In Proceedings of the 2022 56th Asilomar Conference on Signals, Systems, and Computers, Pacific Grove, CA, USA, 31 October–2 November 2022; pp. 905–912.
2. Krishnakumar, A.; Ogras, U.; Marculescu, R.; Kishinevsky, M.; Mudge, T. Domain-Specific Architectures: Research Problems and Promising Approaches. *ACM Trans. Embed. Comput. Syst.* **2023**, *22*, 1–26. [CrossRef]
3. Xie, G.; Zeng, G.; Li, Z.; Li, R.; Li, K. Adaptive dynamic scheduling on multifunctional mixed-criticality automotive cyber-physical systems. *IEEE Trans. Veh. Technol.* **2017**, *66*, 6676–6692. [CrossRef]
4. Amarnath, A.; Pal, S.; Kassa, H.; Vega, A.; Buyuktosunoglu, A.; Franke, H.; Wellman, J.; Dreslinski, R.; Bose, P. HetSched: Quality-of-Mission Aware Scheduling for Autonomous Vehicle SoCs. *arXiv* **2022**, arXiv:2203.13396.
5. Mao, H.; Alizadeh, M.; Menache, I.; Kandula, S. Resource management with deep reinforcement learning. In Proceedings of the 15th ACM Workshop on Hot topics in Networks, Atlanta, GA, USA, 9–10 November 2016; pp. 50–56.
6. Deng, S.; Zhao, H.; Xiang, Z.; Zhang, C.; Jiang, R.; Li, Y.; Yin, J.; Dustdar, S.; Zomaya, A.Y. Dependent function embedding for distributed serverless edge computing. *IEEE Trans. Parallel Distrib. Syst.* **2021**, *33*, 2346–2357. [CrossRef]
7. Wang, C.; Li, R.; Wang, X.; Taleb, T.; Guo, S.; Sun, Y.; Leung, V.C. Heterogeneous Edge Caching Based on Actor-Critic Learning With Attention Mechanism Aiding. *IEEE Trans. Netw. Sci. Eng.* **2023**. [CrossRef]
8. Chai, F.; Zhang, Q.; Yao, H.; Xin, X.; Gao, R.; Guizani, M. Joint Multi-task Offloading and Resource Allocation for Mobile Edge Computing Systems in Satellite IoT. *IEEE Trans. Veh. Technol.* **2023**. [CrossRef]
9. Liu, Z.; Huang, L.; Gao, Z.; Luo, M.; Hosseinalipour, S.; Dai, H. GA-DRL: Graph Neural Network-Augmented Deep Reinforcement Learning for DAG Task Scheduling over Dynamic Vehicular Clouds. *arXiv* **2023**, arXiv:2307.00777.
10. Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A.N.; Kaiser, A.; Polosukhin, I. Attention is all you need. *Adv. Neural Inf. Process. Syst.* **2017**, *30*, 1–11.
11. Hartmanis, J. Computers and intractability: A guide to the theory of np-completeness (michael r. garey and david s. johnson). *Siam Rev.* **1982**, *24*, 90. [CrossRef]
12. Bénichou, M.; Gauthier, J.; Girodet, P.; Hentges, G.; Ribière, G.; Vincent, O. Experiments in mixed-integer linear programming. *Math. Program.* **1971**, *1*, 76–94. [CrossRef]
13. Lambora, A.; Gupta, K.; Chopra, K. Genetic algorithm-A literature review. In Proceedings of the 2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon), Faridabad, India, 14–16 February 2019; pp. 380–384.
14. Hassan, R.; Cohanim, B.; De Weck, O.; Venter, G. A comparison of particle swarm optimization and the genetic algorithm. In Proceedings of the 46th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference, Austin, TX, USA, 18–21 April 2005; p. 1897.
15. Xie, Y.; Gui, F.; Wang, W.; Chien, C. A two-stage multi-population genetic algorithm with heuristics for workflow scheduling in heterogeneous distributed computing environments. *IEEE Trans. Cloud Comput.* **2021**, *11*, 1446–1460. [CrossRef]
16. Rathnayake, U. Migrating storms and optimal control of urban sewer networks. *Hydrology* **2015**, *2*, 230–241. [CrossRef]
17. Topcuoglu, H.; Hariri, S.; Wu, M. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Trans. Parallel Distrib. Syst.* **2002**, *13*, 260–274. [CrossRef]
18. Chronaki, K.; Rico, A.; Casas, M.; Moretó, M.; Badia, R.M.; Ayguadé, E.; Labarta, J.; Valero, M. Task scheduling techniques for asymmetric multi-core systems. *IEEE Trans. Parallel Distrib. Syst.* **2016**, *28*, 2074–2087. [CrossRef]
19. Kohútka, L.; Mach, J. A New FPGA-Based Task Scheduler for Real-Time Systems. *Electronics* **2023**, *12*, 1870. [CrossRef]
20. Xie, G.; Peng, H.; Xiao, X.; Liu, Y.; Li, R. Design flow and methodology for dynamic and static energy-constrained scheduling framework in heterogeneous multicore embedded devices. *ACM Trans. Des. Autom. Electron. Syst. (TODAES)* **2021**, *26*, 1–18. [CrossRef]
21. Hu, B.; Yang, X.; Zhao, M. Online energy-efficient scheduling of DAG tasks on heterogeneous embedded platforms. *J. Syst. Architect.* **2023**, *140*, 102894. [CrossRef]
22. Sung, T.T.; Ryu, B. Deep Reinforcement Learning for System-on-Chip: Myths and Realities. *IEEE Access* **2022**, *10*, 98048–98064. [CrossRef]
23. Mao, H.; Schwarzkopf, M.; Venkatakrishnan, S.B.; Meng, Z.; Alizadeh, M. Learning scheduling algorithms for data processing clusters. In Proceedings of the ACM Special Interest Group on Data Communication, Beijing, China, 19–23 August 2019; pp. 270–288.
24. Puterman, M.L. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*; John Wiley & Sons: Hoboken, NJ, USA, 2014; ISBN 1118625870.
25. Zhou, Y.; Li, X.; Luo, J.; Yuan, M.; Zeng, J.; Yao, J. Learning to optimize dag scheduling in heterogeneous environment. In Proceedings of the 2022 23rd IEEE International Conference on Mobile Data Management (MDM), Paphos, Cyprus, 6–9 June 2022; pp. 137–146.
26. Konda, V.; Tsitsiklis, J. Actor-critic algorithms. *Adv. Neural Inf. Process. Syst.* **1999**, *12*, 1008–1014.
27. Sutton, R.S.; Mcallester, D.; Singh, S.; Mansour, Y. Policy gradient methods for reinforcement learning with function approximation. *Adv. Neural Inf. Process. Syst.* **1999**, *12*, 1057–1063.

28. Arjona-Medina, J.A.; Gillhofer, M.; Widrich, M.; Unterthiner, T.; Brandstetter, J.; Hochreiter, S. Rudder: Return decomposition for delayed rewards. *Adv. Neural Inf. Process. Syst.* **2019**, *32*, 1–12.
29. Xiao, F.; Wang, L. Asynchronous consensus in continuous-time multi-agent systems with switching topology and time-varying delays. *IEEE Trans. Autom. Control* **2008**, *53*, 1804–1816. [CrossRef]
30. Mnih, V.; Badia, A.P.; Mirza, M.; Graves, A.; Lillicrap, T.; Harley, T.; Silver, D.; Kavukcuoglu, K. Asynchronous methods for deep reinforcement learning. In Proceedings of the International Conference on Machine Learning, New York, NY, USA, 20–22 June 2016; pp. 1928–1937.
31. Arda, S.E.; Krishnakumar, A.; Goksoy, A.A.; Kumbhare, N.; Mack, J.; Sartor, A.L.; Akoglu, A.; Marculescu, R.; Ogras, U.Y. DS3: A system-level domain-specific system-on-chip simulation framework. *IEEE Trans. Comput.* **2020**, *69*, 1248–1262. [CrossRef]
32. Braun, T.D.; Siegel, H.J.; Beck, N.; Bölöni, L.L.; Maheswaran, M.; Reuther, A.I.; Robertson, J.P.; Theys, M.D.; Yao, B.; Hensgen, D. A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. *J. Parallel Distrib. Comput.* **2001**, *61*, 810–837. [CrossRef]
33. Cheong, M.; Lee, H.; Yeom, I.; Woo, H. SCARL: Attentive reinforcement learning-based scheduling in a multi-resource heterogeneous cluster. *IEEE Access* **2019**, *7*, 153432–153444. [CrossRef]