IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received September 30, 2020, accepted October 2, 2020, date of publication October 6, 2020, date of current version October 21, 2020.

*Digital Object Identifier 10.1109/ACCESS.2020.3029040*

# Panda: Reinforcement Learning-Based Priority Assignment for Multi-Processor Real-Time Scheduling

**HYUNSUNG LEE[1], JINKYU LEE[2], (Member, IEEE), IKJUN YEOM[2], AND HONGUK WOO[2], (Member, IEEE)**

[1]Department of Electrical and Computer Engineering, Sungkyunkwan University, Suwon 16419, South Korea
[2]Department of Computer Science and Engineering, Sungkyunkwan University, Suwon 16419, South Korea

Corresponding author: Honguk Woo (hwoo@skku.edu)

This work was supported in part by the Ministry of Science and ICT (MSIT), South Korea, through the ICT Creative Consilience Program under Grant IITP-2020-0-01821, supervised by the Institute for Information and Communications Technology Planning and Evaluation (IITP), in part by the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT, and Future Planning, through the Basic Science Research Program, under Grant 2020R1A2C2009809 and Grant 2019R1A2B5B02001794, in part by the Kakao I Research Supporting Program, and in part by Samsung Electronics.

**ABSTRACT** Recently, deep reinforcement learning (RL) technologies have been considered as a feasible solution for tackling combinatorial problems in various research and engineering areas. Motivated by this recent success of RL-based approaches, in this paper, we focus on how to utilize RL technologies in the context of real-time system research. Specifically, we first formulate the problem of fixed-priority assignments for multi-processor real-time scheduling, which has long been considered challenging in the real-time system community, as a combinatorial problem. We then propose the RL-based priority assignment model Panda that employs (i) a taskset embedding mechanism driven by attention-based encoder-decoder deep neural networks, hence enabling to efficiently extract useful features from the dynamic relation of periodic tasks. We also present two optimization schemes tailored to adopt RL for real-time task scheduling problems: (ii) the response time analysis (RTA)-based policy gradient RL and guided learning schemes, which facilitate the training processes of the Panda model. To the best of our knowledge, our approach is the first to employ RL for real-time task scheduling. Through various experiments, we show that Panda is competitive with well-known heuristic algorithms for real-time task scheduling upon a multi-processor platform, and it often outperforms them in large-scale non-trivial settings, e.g., achieving an average 7.7% enhancement in schedulability ratio for a testing system configuration of 64-sized tasksets and an 8-processor platform.

**INDEX TERMS** Priority assignment, global fixed priority scheduling, encoder-decoder neural network, reinforcement learning, real-time system.

## I. INTRODUCTION

Recently, exploiting advanced machine learning techniques to learn heuristics for combinatorial problems has yielded positive results. For example, in [1], an end-to-end deep neural network (DNN) model using a unique output structure confined within a given input, namely the *Pointer* network, was presented to be applicable in the field of solving combinatorial problems, with several example cases such as the traveling salesman problem (TSP). The Pointer network structure facilitates the choice of permutations of input sequences in a supervised learning manner, thereby requiring data samples labeled by another reliable solution. In [2], reinforcement learning (RL) enabled this model structure to learn heuristics by interacting with an environment instead of requiring samples of supervised learning. In particular, the work in [3] exploited the *Transformer* model [4] as well as a greedy rollout baseline, and achieved competitive performance. It can minimize cost and execution time, compared to state-of-the-art heuristics for TSP with many points. This RL-based heuristic learner for combinatorial problems was adopted for task scheduling and resource

The associate editor coordinating the review of this manuscript and approving it for publication was Jihwan P. Choi.

management in computing systems, e.g., cloud computing and server cluster operations [5]–[7].

In this paper, focusing on real-time task scheduling, we formulate priority assignments for global fixed-priority scheduling (GFPS) on a multi-processor platform as a combinatorial problem. We then adapt the RL-based learner with several optimization schemes custom-tailored for the problem, hence demonstrating the applicability of the RL-based learner for problems similar to real-time scheduling. The priority assignment problem for GFPS can be described as as follows. For a set of $n$-periodic tasks each of which has its own time constraint (simply, an $n$-sized taskset), we aim to find a schedulable priority assignment where each task in the taskset is assigned a priority (an integer from 1 to $n$); that is, GFPS with the priority assignment is able to schedule the $m$ highest-priority tasks in each time slot upon a platform comprised of $m$ homogeneous processors without incurring any deadline violation of the periodic tasks over time.

Since the notion of a schedulability test in real-time systems is a typical tool used to determine whether a target taskset is schedulable by GFPS with a given priority assignment, we exploit existing schedulability tests for GFPS [8]–[11]. Our target problem is to find a priority assignment for an $n$-sized taskset, which is deemed schedulable by an existing schedulability test for GFPS. We view this problem as a combinatorial optimization process in the $n!$ search space, i.e., all the permutations of $n$ unique integers ranging from 1 to $n$. In the research domain of real-time scheduling, the priority assignment problem for GFPS has been considered challenging, especially for large-scale tasksets upon a multi-processor platforms. This is because, while existing heuristic priority assignments have been unable to cover schedulable priority assignments for a number of tasksets [12], it becomes computationally intractable to exhaustively apply $n!$ priority assignments for a given taskset to a schedulability test for GFPS if $n$ is large.

In adapting the RL-based learner for the priority assignment problem, we formulate a taskset partition as an RL state in which two dynamic sets of priority-assigned tasks and priority-unassigned tasks are maintained over time. We then adopt RL actions by which a task with the next highest priority is recurrently selected and the taskset partition is updated. In doing so, we employ the encoder-decoder neural network structure enriched by several attention mechanisms. The structure enables to systematically extract the contextual information about the relation of tasks and their priority assignments from the taskset partition. Furthermore, we present two optimization schemes, the response time analysis (RTA)-based policy gradient RL and guided learning schemes which facilitate our model training processes. Our model performs comparably to well-known heuristic algorithms, and outperforms them in large-scale non-trivial settings; for example, it shows 7.7∼13% improvement with respect to the ratio of schedulable priority assignments for target tasksets (schedulability ratio) over the best-performance heuristic algorithm. Furthermore, our model achieves the comparable performance to the union of all the heuristic algorithms.

The main contributions of this paper are as follows:

- We present an RL-based priority assignment model for GFPS on a multi-processor real-time system, namely **Panda** (priority assignment network with deep attention).
- We also present RL training optimization in **Panda** for multi-processor real-time scheduling such as response time analysis (RTA)-based policy gradient learning and deadline monotonic (DM)-based guided learning.
- We then conduct various experiments, and demonstrate the robust schedulability performance and scalability of **Panda** which can handle large tasksets of up to 64 tasks, in comparison with several well-known heuristic algorithms and their union approach.

The rest of the paper is organised as follows. Section II briefly describes our RL-based approach to the problem of priority assignments for multi-processor real-time task scheduling. Section III presents the structure of the priority assignment model **Panda**, and Section IV describes the model training optimization. Section V and Section VI provide the experiment results and the review on related research works. Finally, Section VII concludes the study.

## II. OVERALL SYSTEM

In this section, we first describe the problem of multi-processor real-time scheduling and then present our RL-based approach to the problem.

### A. PRIORITY ASSIGNMENT PROBLEM FOR GFPS

In this paper, we focus on GFPS of $n$ tasks with time constraints in a real-time system of $m$ homogeneous multi-processors. Here, formulation of global fixed-priority assignments or scheduling states that consecutive invocations (jobs) of a periodic task have the same priority, and each job can be assigned on any processor without predetermined partitions of processors. This is consistent with common category definitions in real-time scheduling [12].

Given an $n$-sized taskset, each task $\tau_i$ is specified by its period $T_i$, worst-case execution time $C_i$, and relative deadline $D_i$. In the global fixed-priority assignment, each task is statically assigned a unique priority $i \in \{1, \ldots, n\}$. With this periodic task model, we can consider several variants of multi-processor real-time scheduling, such as implicit or constrained deadline tasks, and preemptive or non-preemptive tasks. A deadline is said to be implicit if $D_i = T_i$, while it is said to be constrained if $D_i \leq T_i$. A job can be either preemptive or non-preemptive. Unless stated otherwise, we focus on periodic, implicit, and preemptive tasks in our GFPS problem.

For a pair consisting of an $n$-sized taskset $\{\tau\}$ and a priority assignment $\pi$, a schedulability test $\mathbf{Test}(\pi, \{\tau\})$ indicates if $\{\tau\}$ can be scheduled on a target platform with the $\pi$-priority scheduling policy, by performing schedulability analysis on

whether or not there exists such a task (in $\{\tau\}$) whose job misses its deadline. $\pi$ corresponds to a permutation of task indexes of $\{\tau\}$.

In general, this test processes every task $\tau_i$ to check if interference of tasks with higher priority than $\tau_i$ cannot be larger than its deadline [9]. Interference of $\tau_i$ quantifies the time period when jobs of $\tau_i$ cannot be executed due to other tasks' jobs in execution. A test for task $\tau_i \in \{\tau\}$ can be defined upon an $m$-processor platform as

$$D_i - C_i \geq \left\lfloor \frac{1}{m} \sum_{\tau_j \in \mathcal{H}(\tau_i)} \mathbf{Intf}(\tau_j, \tau_i) \right\rfloor, \tag{1}$$

where $\mathcal{H}(\tau_i)$ denotes a set of higher priority tasks than $\tau_i$, and $\mathbf{Intf}(\tau_j, \tau_i)$ denotes the upper bound of the amount of interference of task $\tau_i$ caused by task $\tau_j$, calculated via schedulability analysis. Then, if the inequality Eq. (1) holds for all tasks in $\{\tau\}$, $\{\tau\}$ is deemed to be schedulable. This can be further formulated as a schedulability test $\mathbf{Test}(\pi, \{\tau\})$ where a system configuration such as $m$-processors is presumed for the respective $\pi$-priority scheduling policy. Accordingly, $\mathbf{Test}(\pi, \{\tau\}) = 1$ ensures that the taskset $\{\tau\}$ with the priority assignment $\pi$ is deemed schedulable, while $\mathbf{Test}(\pi, \{\tau\}) = 0$ does not. For example, suppose we have a taskset $\{\tau\} = \{\tau_1, \tau_2, \tau_3\}$ and its priority assignment $\pi = [2, 3, 1]$; then, $\mathbf{Test}([2, 3, 1], \{\tau_1, \tau_2, \tau_3\}) = 1$ indicates that the scheduling policy with the fixed task priority assignment such as 2 for $\tau_1$, 3 for $\tau_2$, and 1 for $\tau_3$ (simply, $\pi$-priority scheduling) is deemed schedulable.

Existing studies have addressed the priority assignment problem for GFPS, by (i) developing improved schedulability tests and (ii) applying heuristic priority assignments to the tests. For (i), several studies have developed schedulability tests that reduce pessimism of calculating the right-hand-side of Eq. (1) [8]–[11]. For (ii), there have been several heuristic priority assignments: deadline monotonic (DM: the smaller is $D_i$, the higher is the priority), DkC (the smaller is $D_i - k \cdot C_i$, the higher is the priority, where $k$ depends on the number of processors) [13], and DM-DS (the heavier is the utilization or the smaller is the slack time, the higher is the priority) [14], [15].

Existing schedulability tests for multi-processor real-time systems are inherently pessimistic in upper-bounding the right-hand-side of Eq. (1), and are not able to identify necessarily all schedulable priority assignments of tasksets. However, it has been considered that the capability limit of the schedulability tests is insignificant in practice, compared to the loose bound of traditional heuristic algorithms for priority assignments in real-time task scheduling [16]. In that sense, we focus on *schedulability ratio* as the performance metric of our Panda model to represent the fraction of priority assignments yielded by Panda that successfully pass our target schedulability test in [8]. We then compare the schedulability ratio of the target schedulability test associated with existing heuristics priority assignments. There have been a few schedulability tests [11], [17] that allow application of the optimal priority assignment (OPA) technique [18], [19] only with $O(n^2)$ time-complexity. Such OPA-compatible schedulability tests exhibit low schedulability performance compared to the state-of-the-art OPA-incompatible tests [8]–[11] for a given priority assignment. On the other hand, the tests once incorporated into the OPA technique may have higher schedulability performance by exploiting the OPA under the tests. Therefore, we also compare the schedulability ratio of the low-schedulability-performance OPA-compatible tests associated with the OPA technique.

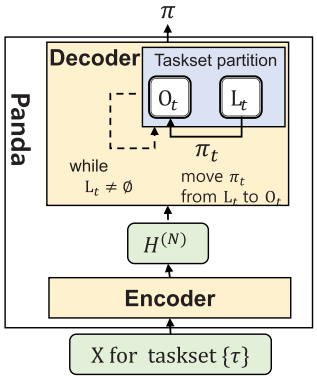


**FIGURE 1.** Overall model structure of Panda: Panda($\{\tau\}$) $\mapsto$ $\pi$, where Panda yields a priority assignment $\pi$ for an input taskset $\{\tau\}$ via the encoder-decoder neural networks.

### B. A SCHEDULER WITH ENCODER-DECODER

Figure 1 depicts the model structure of Panda where an $n$-sized taskset $\{\tau\}$ is given as input and its priority assignment $\pi$ of the same size is generated as output. That is,

$$\textsf{Panda}(\{\tau\}) \mapsto \pi, \quad \text{e.g.,}$$
$$\textsf{Panda}(\{\tau_1, \tau_2, \tau_3\}) \mapsto [2, 3, 1],$$

where $\tau_1$, $\tau_2$, and $\tau_3$ are assigned priorities 1, 3, and 2 respectively. We set $\pi$ to be a permutation of task indexes of $\{\tau\}$ in that $\pi = [\pi_1, \pi_2, \ldots, \pi_n]$ can specify a priority-ordered taskset $[\tau_{\pi_1}, \tau_{\pi_2}, \ldots, \tau_{\pi_n}]$ where $\pi_i = 1, \ldots, n$ and $\pi_i \neq \pi_j$, provided all priorities are distinct. Thus, without loss of generality, we simply call this priority assignment $\pi$, permutation. As shown in Figure 1, the model consists of encoder and decoder networks. The encoder network transforms the raw features of $\{\tau\}$ to vector representation $H^{(N)}$, and the decoder network iteratively makes inferences using the vector representation to render a priority assignment $\pi$.

As explained, the model generates a permutation of a given taskset. In doing so, the decoder network internally maintains a partition of the taskset $(\mathrm{O}, \mathrm{L})$. Our model first initializes O to be empty and $\mathrm{L} = \{1, 2, \ldots, n\}$. Then the decoder completes O by iteratively selecting a task in L and moving it to the tail of O. It continues to update the ordered set O of priority-assigned tasks and the other set $L$ of priority-unassigned tasks by

$$\mathrm{O}_{t+1} = \mathrm{O}_t + [\pi_t], \quad \mathrm{L}_{t+1} = \mathrm{L}_t - \{\pi_t\}, \tag{2}$$

over discrete time-steps $t$. Note that $O_t$ and $L_t$ represent O and L at step $t$ respectively, and this decoding process recurs for $n$-steps until L becomes empty. Accordingly, for an $n$-sized taskset, $O_t \cup L_t = \{1, 2, \ldots, n\}$ holds for all $t = 1, \ldots, n$, and the initial carnality of L is $n$. For achieving a schedulable policy, the model makes use of such a time-varying partition in the decoder network. According to RL formation, we also notate $(O_t, L_t)$ as state $s_t$.

### C. MODEL TRAINING WITH REINFORCE

To train the encoder-decoder model, we exploit the REINFORCE algorithm [20] that maximizes expected cumulative rewards of the actions by gradient ascent. Regarding RL formulation, each action selects a task in $L_t$ and moves it to $O_t$, while observing the state $s_t = (O_t, L_t)$. Then, reward values can be determined according to the schedulability evaluation of whether an input taskset $\{\tau\}$ is deemed schedulable with the output priority $\pi$, i.e., the reward is defined as $\mathbf{Test}(\pi, \{\tau\})$.

This reward design based on a schedulability test seems natural but incurs a severe problem of sparse rewards [21]. Consider an $n$-sized taskset. There exist $n!$ priority assignments for the taskset. Only a few could be feasible permutations (schedulable priority assignments) especially when we have non-trivial samples of $n$-sized tasksets and $m$-processor system configurations that together render a low chance of schedulable priority assignments, i.e., when having high taskset utilization $\sum_{\tau_i \in \{\tau\}} C_i/T_i$. This poses a sparsity of reward signals, which in turn produces calculated gradients of zero in most cases.

**TABLE 1. Ratio of schedulable priority assignments with respect to the size of tasksets ($n$).**

| $n$-tasks | All Perm.(%) | Random Perm.(%) | Sched. by DM(%) |
|---|---|---|---|
| 4 | 54.26 | - | 94.1 |
| 6 | 21.45 | - | 92.2 |
| 8 | 7.45 | - | 91.3 |
| 10 | - | 2.62 | 87.6 |
| 12 | - | 0.91 | 87.3 |
| 14 | - | 0.27 | 87.1 |
| 16 | - | 0.07 | 86.9 |

In Table 1, we show the ratio of schedulable priority assignments with respect to the size of tasksets $n = 4, 6, \ldots, 14, 16$. The table column "All Perm." denotes the cases where all possible permutations for each taskset are exhaustively tested. The column "Random Perm." denotes the cases where up to 10K permutations randomly selected for each taskset are tested. The column "Sched. by DM" denotes the ratio of priority assignments deemed schedulable by the deadline monotonic (DM). Note that for small $n \leq 8$, we test all the permutations, but for large $n \geq 10$, we test only up to 10K permutations, considering the required testing time. As observed, the former two cases commonly show rapid decrements, as the size of tasksets increases, while the schedulable priority assignment ratio by DM shows slow decrements. The test results in Table 1 clarify the cause of sparse rewards such that for large $n$, only a small fraction of samples in RL training can be useful.

To tackle this issue, we make use of the response time analysis (RTA) [9] for individual tasks, so that we obtain more frequent reward signals in model training. It is possible to calculate the response time of partially priority-assigned tasks and the upper bound of interference caused by those tasks, as a task can be interfered with only higher priority tasks. This calculation can be incorporated into our RL formulation as a reward, allowing the model to learn quickly with useful reward signals more frequently generated. We call this training scheme, *progressive RTA-reward*.

In addition, we exploit the notion of importance sampling [22] to data-efficiently guide a model at the initial training phase by leveraging an existing heuristic method. We empirically observe that it is effective to guide our model to learn with samples generated by the DM (deadline monotonic) heuristic policy especially when the model rarely learns to gather sufficient positive rewards. We call this training scheme, *DM-based guided learning*.

## III. ATTENTION-BASED SCHEDULER MODEL

In this section, we describe our model structure, an attention-based deep neural model that consists of encoder and decoder networks. For representing model parameters, we use superscript $\theta$. For example, we denote the affine transform as

$$\mathbf{Aff}_v(x) = W_v^{(\theta)} x + b_v^{(\theta)} \tag{3}$$

where $v$ is a name by which its implementation should be distinguished from others. In the following, we often omit $v$, assuming that the model parameters (e.g., $W^{(\theta)}$, $b^{(\theta)}$) are properly defined for each transform.

### A. TASK ENCODER

According to the GFPS problem, we structure the model to use a taskset specification as input and yield its priority assignment. We presume that an underlying platform configuration such as the number of processors is confined in a training environment. Our training environment system is implemented to support configurable $m$-processor platform settings.

In task encoding, a set of raw level representation of input tasks are first transformed to the vector representation that can be further processed for extracting their relational features. For each task $\tau_i$, its specification parameters such as period $T_i$, worst execution time $C_i$, deadline $D_i$, and additional features in Table 2 are re-scaled and contained in the respective raw representation.

**TABLE 2. Raw features for a task.**

| | |
|---|---|
| Task spec. | $C_i, T_i, D_i$ |
| Task spec. in log-scale | $\log(1+C_i), \log(1+T_i), \log(1+D_i)$ |
| Utilization, etc | $C_i/T_i$, $C_i/D_i$, $D_i/T_i$ |
| Slack time, etc | $(T_i - C_i)$, $(T_i - D_i)$, $(D_i - C_i)$ |

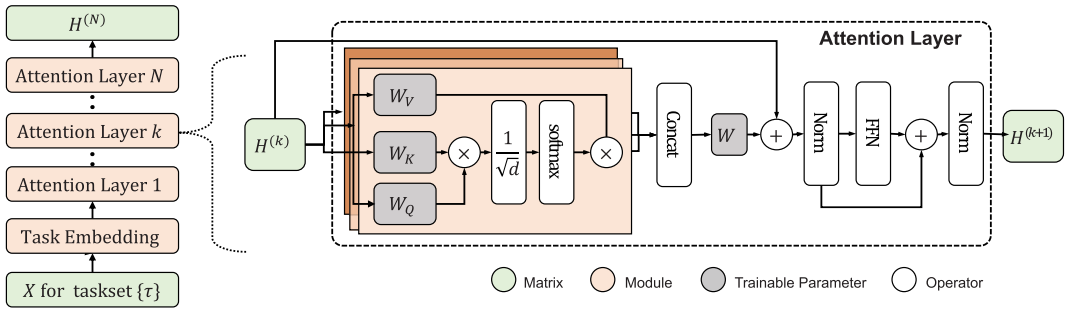


**FIGURE 2.** Task encoder network where $N$ attention-based embedding layers are stacked and relational features of tasks are extracted.

Figure 2 illustrates the detailed structure of the task encoder network that yields a matrix representation in which each column individually corresponds to a task embedding vector. Each task $\tau_i$ in a given $n$-sized taskset has its raw vector representation $x_i$ combining the input features in Table 2. In our implementation, $x_i$ is a 12-dimensional vector. The first embedding layer (Task Embedding) individually transforms $x_i$ into a vector $h_i^{(0)}$ via

$$h_i^{(0)} = \mathbf{Aff}(\tanh(\mathbf{Aff}(x_i)) \tag{4}$$

where $\tanh(\cdot)$ is applied element-wisely. In $\mathbf{Aff}(\cdot)$ (defined in Eq. (3)), $W^{(\theta)}$ is a $d \times 12$ matrix where $d$ is the size of hidden layers and embedding data; we set $d = 128$ in our implementation. For all $\tau_i$, we obtain the matrix

$$H^{(0)} = [h_1^{(0)}, h_2^{(0)}, \ldots, h_n^{(0)}]. \tag{5}$$

To implement the task encoder network, we adapt Transformer [4] where multiple attention layers are stacked. Each attention layer is formulated as a composite function for $k = 0, 1, \ldots, N-1$.

$$H^{(k+1)} = \mathbf{Norm}(E + \mathbf{FFN}(E))$$
$$\text{where } E = \mathbf{Norm}(H^{(k)} + \mathbf{MHA}(H^{(k)})) \tag{6}$$

Here $\mathbf{Norm}(\cdot)$ represents a normalization layer, e.g., batch normalization [23], and layer normalization [24]; and $\mathbf{FFN}(\cdot)$ denotes a multi-layer perceptron with ReLU activation. Note that $H^{(0)}$ (in Eq. (5)) is the input of the first attention layer. With $N$-attention layers, the output $H^{(k+1)}$ of the $k$-th layer is fed to the $(k+1)$-th layer. The $N$-step processing upon the attention layers generates the matrix representation $H^{(N)}$ which contains the relational features in a given taskset. To embed tasks into vector representation with respect to their relationship, we specifically exploit self-attention $\mathbf{SA}(\cdot)$ which is a variant of scaled dot-product attention $\mathbf{Att}(\cdot)$ [4]. These mechanisms are defined as

$$\mathbf{SA}(X) = \mathbf{Att}(W_Q^{(\theta)}X, W_K^{(\theta)}X, W_V^{(\theta)}X),$$
$$\mathbf{Att}(Q, K, V) = V\ \mathbf{Softmax}(\frac{1}{\sqrt{d}}K^T Q). \tag{7}$$

Self-attention is effective for embedding data of multiple tasks when embedding each individual task represents the relation with the other tasks from a scheduling perspective. $\mathbf{Softmax}(\cdot)$ of a matrix performs the softmax [25] operation over each column. In self-attention, queries $Q$, keys $K$, and values $V$ are represented by the respective projection on the same input $X$, e.g., $W_Q^{(\theta)}X$, $W_K^{(\theta)}X$, and $W_V^{(\theta)}X$. For achieving stable embedding quality, we use multiple self-attention modules in parallel, similar to the method in [4].

$$\mathbf{MHA}(X) = W^{(\theta)}\ \mathbf{Concat}(\mathbf{SA}_1(X), \mathbf{SA}_2(X), \ldots) \tag{8}$$

We represent the task encoding procedure for priority assignments in the first part of Algorithm 1, where a taskset is converted into a matrix representation by the *TaskEmbedding* function in Eq. (4), and the computation of $N$-attention layers in Eq. (6) with several self-attentions is abstracted through the *AttentionLayer* function.

### B. PRIORITY DECODER

The priority decoder is used to make sequential inferences on priorities. It selects $\pi_1$ first indicating the index of the task of which priority is highest, and then selects the index of the task with the second-highest priority $\pi_2$. This procedure continues until every task has been assigned its priority, comprising $n$-iterative operations between the $\mathrm{O}_t$ and $\mathrm{L}_t$ sets in Eq. (2). Specifically, the priority decoder takes a taskset partition as a state

$$s_t = (\mathrm{O}_t,\ \mathrm{L}_t) \tag{9}$$

and uses its respective embedding data as input at (time)-step $t$. Then the priority decoder infers a probability distribution in which $\pi_t$ is mapped with each element in $\mathrm{L}_t$, i.e., a set of unassigned task indexes. Figure 3 depicts the recurrent structure of the priority decoder network. The second part of Algorithm 1 represents the decoder network implementation with taskset partition embedding, context vector calculation, and probability calculation steps for a next priority-assigned task. The complexity of the $n$-iterative process with this 3-steps decoding procedure is $O(n^2d^2)$ for an $n$-sized taskset where $d$ is the size of embedding data. In the following, the procedure will be explained in detail, where the 3-steps correspond to Eq. (10), (11), and (12) respectively.

In applying deep neural networks to a combinatorial optimization problem, it is essential to have proper representation of contextual information relevant to the

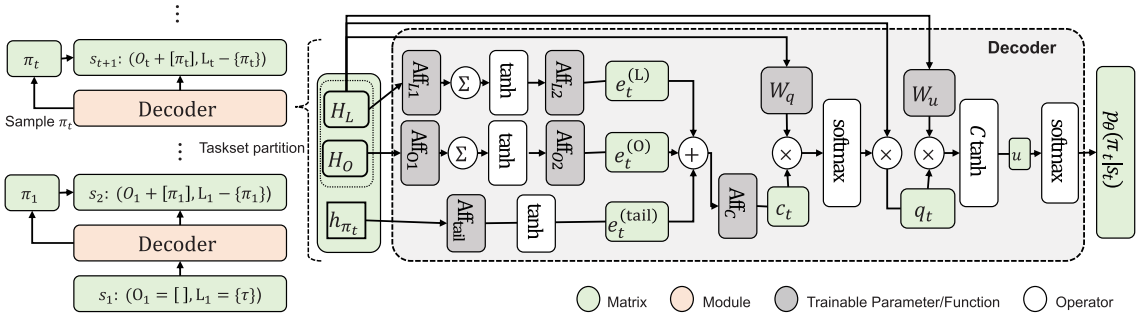


**FIGURE 3.** Priority decoder network where recurrent task selection upon the time-varying taskset partition is made over discrete steps *t* for *n*-sized taskset: each Aff with its model parameters is distinctively represented by its subscript that is omitted in Eq. (10)-(11).

problem structure [3]. In the GFPS problem structure, we view the recurrent relation in the taskset partition $(\mathrm{O}_t, \mathrm{L}_t)$ as the most important context. Maintaining the partition over time-steps allows the priority decoder to continuously make use of the schedulability-related features during model training. It is obvious that $\mathrm{L}_t$ manages the contextual information of valid RL actions, since at step $t$, it continues to strictly confine the RL actions (i.e., selecting one from $\mathrm{L}_t$) within $\mathrm{L}_t$. Similarly, $\mathrm{O}_t$ manages the contextual information of RL states. Consider $\tau_{\pi_{t+1}}$ selected at step $t+1$. Obviously, it is assigned a priority $t+1$. Only such task as $\tau_i \in \mathrm{O}_t$ can interfere and preempt $\tau_{\pi_{t+1}}$ since each has a priority higher than $t+1$. We leverage this contextual information.

The embedding output of the task encoder network that was previously explained (i.e., $H^{(N)} = [h_1^{(N)}, \ldots, h_n^{(N)}]$ where each $h^{(N)}$ denotes task embedding data) is fed to the priority decoder network.

$$e_t^{(O)} = \tanh\left(\mathbf{Aff}\left(\sum\nolimits_{j\in \mathrm{O}_t} \tanh\left(\mathbf{Aff}(h_j)\right)\right)\right),$$
$$e_t^{(L)} = \tanh\left(\mathbf{Aff}\left(\sum\nolimits_{j\in \mathrm{L}_t} \tanh\left(\mathbf{Aff}\left(h_j\right)\right)\right)\right) \quad (10)$$

The vectors $e_t^{(O)}$ and $e_t^{(L)}$ represent the embedding data for each set in the time-varying partition $(\mathrm{O}_t, \mathrm{L}_t)$. From an RL formulation perspective, the vectors can be seen as the embeddings for complex RL states.

The priority decoder follows the structure of attention-based networks by which tasks are successively selected and the taskset partition is updated. It is commonly observed that fixed-priority assignment heuristics assign similar priorities to tasks of similar task parameter values in most cases. Thus, we also conjecture that successively selected tasks by the priority decoder share similar properties, and exploit this relation when making successive selections. Specifically, we utilize the previously selected task $h_{\pi_{t-1}}^{(N)}$, and combine it with the vectors for the time-varying partition in Eq. (10) to acquire a context vector $c_t$,

$$c_t = \mathbf{Aff}(e_t^{(O)} + e_t^{(L)} + e_t^{(\mathrm{tail})}) \quad \text{where } e_t^{(\mathrm{tail})} = \tanh(\mathbf{Aff}(h_{\pi_{t-1}}^{(N)})). \quad (11)$$

**Algorithm 1** Model Interference of **Panda**

*// (i) Task encoding for n-sized taskset data X*
$H^{(0)} = \texttt{TaskEmbedding}(X)$ using Eq. (4)
**for** $k \leftarrow 1$ **to** N **do**
  $H^{(k)} = \texttt{AttentionLayer}_i(H^{(k-1)})$ using Eq. (6)
**end for**
*// (ii) Priority decoding using the taskset partition (O, L)*
$O \leftarrow [], L \leftarrow \{1, 2, \ldots, n\}$
**for** $t \leftarrow 1$ **to** $n$ **do**
  $H_{\mathrm{L}} = \{h_j | j \in L\}$ for $h_j \in H^{(N)}$
  Calculate the embedding $e_t^{(C)}, e_t^{(O)}$ using Eq. (10)
  Calculate the context $c_t$ using Eq. (11)
  Calculate the probability $p_\theta(\pi_t|s_t)$ using Eq. (12)
  Sample $\pi_t$ from $p_\theta(\pi_t|s_t)$
  $O.\texttt{append}(\pi_t)$, $L.\texttt{remove}(\pi_t)$
**end for**
*// Return priority assignment inferred*
**return** $O$

The context vector is calculated based on the taskset partition at the current step $t$ and a previously selected task at $t-1$, which are considered most relevant to successive priority assignments for GFPS.

Based on the context vector, the priority decoder then computes the probability distribution $p_\theta(\cdot|s_t)$ that the task $\tau_{\pi_t}$ (which is not yet assigned a priority) will be assigned priority $t$ for state $s_t$ (in Eq. (9)). This distribution is formulated as

$$p_\theta(\pi_t|s_t) = \mathbf{Softmax}(C \cdot \tanh(h_i\, W_u^{(\theta)}\, q_t)) \quad \text{for } i \in L_t$$
$$\text{where } q_t = H_{\mathrm{L}}\, \mathbf{Softmax}(H_{\mathrm{L}}\, W_q^{(\theta)}\, c_t). \quad (12)$$

Here, $H_{\mathrm{L}}$ represents the matrix of priority-unassigned tasks, which is extracted from $H^{(N)}$. Thus, $q_t$ corresponds to the aligned context for the priority-unassigned tasks. By $q_t$, the distribution becomes confined within the valid indexes for those priority-unassigned tasks. Note that $C$ yields soft-clipping where $C = 5.0$ is found empirically. As a result, using the probability calculated in Eq. (12), the priority decoder is able to select one from $(n-t+1)$ unassigned tasks at step $t$.

## IV. MODEL TRAINING SCHEMES

In model training, we update the model parameters $\theta$ by employing a gradient ascent method, and specifically, we use the REINFORCE algorithm [20] to estimate the gradient. For describing the procedure of model training, we introduce a notion of trajectory

$$\omega = [s_1^{(\omega)}, \pi_1^{(\omega)}, s_2^{(\omega)}, \pi_2^{(\omega)}, \ldots, s_n^{(\omega)}, \pi_n^{(\omega)}], \tag{13}$$

which is a finite sequence of states $s_t$ and actions $\pi_t$ governed by a policy that is used to generate the trajectory. Unlike a general RL formulation where state transitions are stochastic, under our target policy, state transitions are deterministic in that $s_{t+1}$ is completely determined by $s_t$ and $\pi_t$ (i.e., selecting an individual task, as in Eq. (2) and (9)). Therefore, for our target policy $p_\theta$ and a given taskset $\{\tau\}$, we can represent the probability of trajectory $\omega_{p_\theta}$, $p_\theta[\pi = \omega_{p_\theta}|\{\tau\}]$ that is governed by policy $p_\theta$ as

$$p_\theta[\pi = \omega_{p_\theta}|\{\tau\}] = \prod_{i=1}^{n} p_\theta[\pi_t = \pi_t^{(\omega_{p_\theta})}|s_t^{(\omega_{p_\theta})}]. \tag{14}$$

Then, the estimated gradient of the expected reward of trajectories by $p_\theta$ can be calculated as

$$\nabla_\theta J = \mathbb{E}_{\omega_{p_\theta}\sim p_\theta}\left[\sum_{t=1}^{n}\left(R_t\ \nabla_\theta \log p_\theta[\pi_t = \pi_t^{(\omega_{p_\theta})}|s^{(\omega_{p_\theta})}]\right)\right] \tag{15}$$

where $R_t = \sum_{j=t}^{n}\gamma^{j-t}r_j$ and $\gamma \in (0, 1]$ denotes a discount factor. Since the return $R_t$ should be consistent with the evaluation metric of scheduling, we employ the reward scheme based on a schedulability test that determines whether a given taskset $\{\tau\}$ is deemed schedulable with priority assignment $\pi$. In general, a schedulability test can be performed only when all tasks are assigned a priority; thus, it allows formulation of reward $r_j$ as

$$r_j = \begin{cases} \textbf{Test}(\pi, \{\tau\}) & \text{if } j = n \\ 0 & \text{otherwise.} \end{cases} \tag{16}$$

Each task selection is considered to contribute equally to the schedulability of a permutation (a priority assignment $\pi$). Thus, we set the discount factor $\gamma = 1$. The $\gamma$ and Eq. (16) together render the same $R_t$ for all $t$. Thus, we denote $R_t$ as $R$, and rewrite Eq. (15) as

$$\begin{aligned}\nabla_\theta J &= \mathbb{E}_{\omega_{p_\theta}\sim p_\theta}\left[R\ \nabla_\theta \log p_\theta[\pi = \omega_{p_\theta}|\{\tau\}]\right] \\ &\approx \frac{1}{|\mathcal{B}|}\sum_{\{\tau\}'\in\mathcal{B}}\left(R\ \nabla_\theta \log p_\theta[\pi = \omega_{p_\theta}|\{\tau\}']\right)\end{aligned} \tag{17}$$

where $\omega_{p_\theta}$ is the trajectory of policy $p_\theta$ upon taskset $\{\tau\}'$, and $\mathcal{B}$ is a batch of training taskset samples. We then obtain the model update $\theta \leftarrow \theta + \lambda\ \nabla_\theta J$, with learning rate $\lambda$.

### A. PROGRESSIVE RTA-REWARD SCHEME

As described above, it is natural to leverage a schedulability test for training a model that intends to solve the priority assignment problem for GFPS. However, we encountered the issue of sparse rewards [21] such that achievable reward values were minimal for most priority assignments, severely inhibiting model training. It is because there exist only a few schedulable priority assignments, particularly for large tasksets, as explained in Table 1. It turns out that $\nabla_\theta J = 0$ holds in most cases, and model parameters can be barely updated. This often renders our model unable to learn a stable policy.

To tackle this issue of sparse rewards, we calculate the response times of individual tasks whenever a task is assigned its priority, and incorporate them into our RL formulation as auxiliary rewards. We refer to this RL scheme as *progressive RTA-reward*, formulated based on Eq. (1) as

$$r_i^{\text{aux}} = \frac{1}{n}\mathbb{I}\left(D_i - C_i \geq \left\lfloor \frac{1}{m}\sum_{\tau_j\in\mathcal{H}(\tau_i)}\textbf{Intf}(\tau_j, \tau_i)\right\rfloor\right) \tag{18}$$

where the indicator $\mathbb{I}(\cdot)$ yields 1 if a given input is evaluated as true, but 0, otherwise. Recall that $\mathcal{H}(\tau_i)$ denotes a set of higher priority tasks than $\tau_i$. Similar to Eq. (15), we set the discount factor $\gamma = 1$, and obtain $R_t^{\text{aux}} = \sum_{j=t}^{n} r_j^{\text{aux}}$ with which we can replace the return in the gradient estimate as

$$\begin{aligned}&\nabla_\theta J_{\text{aux}} \\ &= \mathbb{E}_{\omega_{p_\theta}\sim p_\theta}\left[\sum_{t=1}^{n}\left(R_t^{\text{aux}}\ \nabla_\theta \log p_\theta[\pi_t = \pi_t^{(\omega_{p_\theta})}|s^{(\omega_{p_\theta})}]\right)\right].\end{aligned} \tag{19}$$

By incorporating the auxiliary rewards into the training objective, therefore, we reformulate the gradient of the objective function as

$$\nabla_\theta J_{tot} = \nabla_\theta J + \nabla_\theta J_{\text{aux}}. \tag{20}$$

Notice that this objective function is augmented by the progressive RTA-reward that renders frequent reward signals.

We also adopt the baseline reduction scheme [26], and incorporate the respective baselines into Eq. (17) and (19) to decrease the variance of estimated gradients. Specifically, we have

$$\begin{aligned}\nabla_\theta J_{tot} &= \nabla_\theta J + \nabla_\theta J_{\text{aux}} \\ &\approx \frac{1}{|\mathcal{B}|}\sum_{\{\tau\}'\in\mathcal{B}}\left((R - B)\nabla_\theta \log p_\theta[\pi = \omega_{p_\theta}|\{\tau\}']\right) \\ &\quad + \frac{1}{|\mathcal{B}|}\sum_{\{\tau\}'\in\mathcal{B}}\sum_{t=1}^{n}\left((R_t^{\text{aux}} - B_t^{\text{aux}})\right. \\ &\quad \left.\nabla_\theta \log p_\theta[\pi_t = \pi_t^{(\omega_{p_\theta})}|s^{(\omega_{p_\theta})}]\right)\end{aligned} \tag{21}$$

where $B$ and $B_t^{\text{aux}}$ are baselines. These baselines are calculated by another policy, say a $\beta$-parameterized baseline policy, which is similar to the target policy but with a different set of parameters $\beta$. It is gradually improved by iterative

**Algorithm 2** Model Training of **Panda**

Initialize the target model parameters $\theta$
***// (i) Guided Learning using the DM heuristic algorithm***
**for** $i \leftarrow 1, \ldots, N_{\text{guide}}$ **do**
  $\Delta\theta \leftarrow 0$, Sample batch $\mathcal{B}$ from training data set $\mathcal{D}$
  **for** taskset $\{\tau\}$ in $\mathcal{B}$ **do**
    Sample a trajectory $\omega_q$ using the DM policy $q$
    Calculate $R$ for $\omega_q$ using Eq. (16)
    Calculate $p_\theta[\pi_t = \pi_t^{(q)} | s_t^{(q)}]$ for all $t$
    $\Delta\theta \leftarrow \Delta\theta + R\,\nabla_\theta \log p_\theta[\pi = \omega_q | \{\tau\}]$ using Eq. (26)
  **end for**
  $\theta \leftarrow \theta + \frac{\lambda}{|\mathcal{B}|}\Delta\theta$
**end for**
***// (ii) Normal Learning using the target policy*** $p_\theta$
Initialize the baseline model parameters $\beta$ by $\beta \leftarrow \theta$
**for** $i \leftarrow 1, \ldots, N_{\text{train}}$ **do**
  $\Delta\theta \leftarrow 0$, Sample batch $\mathcal{B}$ from training data set $\mathcal{D}$
  **for** taskset $\{\tau\}$ in $\mathcal{B}$ **do**
    Sample trajectory $\omega_{p_\theta}$ using $p_\theta$ with stochastic sampling
    Calculate $R$, $R_t^{\text{aux}}$ for $\omega_{p_\theta}$ using Eq. (16), (18)
    Calculate $p_\theta[\pi_t = \pi_t^{(\omega_{p_\theta})} | s_t^{(p_\theta)}]$ for all $t$
    Sample trajectory $\omega_{p_\beta}$ using $p_\beta$ with greedy sampling
    Calculate $B$, $B_t^{\text{aux}}$ using Eq. (22)
    $\Delta\theta \leftarrow \Delta\theta + (R - B)\,\nabla_\theta \log p_\theta\left[\pi = \omega_{p_\theta} | \{\tau\}\right] +$
    $\sum_t \left((R_t^{\text{aux}} - B_t^{\text{aux}}) \nabla_\theta \log p_\theta[\pi_t = \pi_t^{(\omega_{p_\theta})} | s^{(\omega_{p_\theta})}]\right)$ using Eq. (21)
  **end for**
  $\theta \leftarrow \theta + \frac{\lambda}{|\mathcal{B}|}\Delta\theta$
  **if** paired-T Test on $R \neq B$ **then**
    $\beta \leftarrow \theta$
  **end if**
**end for**

updates of $\beta \leftarrow \theta$ according to the paired-T test on $(R \neq B)$, similar to [3]. Then, we obtain the baselines from the $\beta$-parameterized model, i.e.,

$$B = \textbf{Test}(\pi_{p_\beta} | \{\tau\}), \quad B_t^{\text{aux}} = \sum_{j=t}^{n} r_j^{\text{aux}} \tag{22}$$

where $\pi_{p_\beta}$ is a priority assignment by the $p_\beta$ policy of the $\beta$-parameterized model.

### B. GUIDED LEARNING BASED ON DM

When adopting our RL-based learner for large $n$-sized tasksets (i.e., $n \geq 48$), especially during the early stage of training, we frequently experienced rare updates of model parameters, which contribute to nonconverged models.

To tackle this challenge, we leverage the importance sampling and guided learning scheme [22], [26]. Specifically, we employ off-policy learning at the initial training phase, making use of samples generated by a heuristic method or a base policy. We empirically observe that deadline monotonic (DM) fits well as a base policy for guided learning. This is because the DM heuristic method can be structured as a scheduling policy that sequentially performs priority assignments, starting from the highest, similar to our **Panda** model. Suppose we have such a base policy $q$ using DM, and a trajectory generated by $q$ such as $\omega_q = [s_1^{(\omega_q)}, \pi_1^{(\omega_q)}, s_2^{(\omega_q)}, \pi_2^{(\omega_q)}, \ldots, s_n^{(\omega_q)}, \pi_n^{(\omega_q)}]$. To exploit trajectory data by $q$ in model training, we incorporate the importance ratio [26] into estimated gradients as

$$\nabla_\theta J = \mathbb{E}_{\omega_q \sim q}\left[R \sum_{t=1}^{n} \left(\rho_t \nabla_\theta \log p_\theta[\pi_t = \pi_t^{(\omega_q)} | s^{(q)}]\right)\right]. \tag{23}$$

Here, $\rho_t$ is the importance ratio that represents the ratio of probabilities of sample trajectories by base policy $q$ and our target policy $p_\theta$. Then, it is possible to simplify the equation further using the below Eq. (24). Because we use a deterministic policy using DM as base policy $q$, we necessarily have $q[\pi_{t'} = \pi_{t'}^{(\omega_q)} | s_{t'}^{(q)}] = 1$.

$$\begin{aligned}\rho_t &= \prod_{t'=1}^{t} \frac{p_\theta[\pi_{t'} = \pi_{t'}^{(\omega_q)} | s_{t'}]}{q[\pi_{t'} = \pi_{t'}^{(\omega_q)} | s_{t'}]} = \prod_{t'=1}^{t} p_\theta[\pi_{t'} \\ &= \pi_{t'}^{(\omega_q)} | s_{t'}^{(q)}]\end{aligned} \tag{24}$$

During the early stage of model training, we estimate the target policy as a uniform distribution on valid actions, i.e., $p_\theta[\pi_t = \tau | s_t] \approx \frac{1}{n-t+1}$. As a result, we can revise the gradient estimate as

$$\nabla_\theta J = \mathbb{E}_{\omega_q \sim q}\left[R \sum_{t=1}^{n} \left(\frac{(n-t)!}{n!} \nabla_\theta \log p_\theta[\pi_{t'} = \pi_{t'}^{(\omega_q)} | s^{(q)}]\right)\right] \tag{25}$$

In Eq. (25), we observe that the estimated gradient $\nabla_\theta J$ is a weighted sum of the gradients $(R_t \nabla_\theta \log p_\theta[\pi_{t'} = \pi_{t'}^{(\omega_q)} | s^{(q)}])$ over steps $t$ with varying weights $(n-t)!/n!$. The weight decreases as $t$ increases to $n$. That is, for low-priority tasks that are selected at large $t$, the gradient is likely to vanish. To prevent the vanishing gradient problem, we cap the weight by $\max(\frac{(n-t)!}{n!}, 1)$. Capping is effective for reducing variances and stabilizing the gradient estimate [27]. We thus rewrite the above Eq. (25) with capping as

$$\begin{aligned}\nabla_\theta J &= \mathbb{E}_{\omega_q \sim q}\left[R \sum_{t=1}^{n} \left(\nabla_\theta \log p_\theta[\pi_t = \pi_t^{(\omega_q)} | s^{(q)}]\right)\right] \\ &= \mathbb{E}_{\omega_q \sim q}\left[R\,\nabla_\theta \log p_\theta\left[\pi = \omega_q | \{\tau\}\right]\right] \\ &\approx \frac{1}{|\mathcal{B}|} \sum_{\{\tau\}' \in \mathcal{B}} \left(R\,\nabla_\theta \log p_\theta\left[\pi = \omega_q | \{\tau\}'\right]\right).\end{aligned} \tag{26}$$

Notice that the gradient estimate above is equivalent to Eq. (17) except that samples are obtained from the base policy $q$ using DM, not by $p_\theta$ which we aim to train.

In Algorithm 2, we compile our RL training schemes introduced for the GFPS problem, and present them as one integrated solution. To facilitate training of our model at the early stage, we conduct a two-step training process: (i) in the first part, we perform DM-based guided learning,

which utilizes the gradient estimate driven by base policy $q$ for training target policy $p_\theta$ (through gradient estimates in Eq. (25) and (26)). Recall that $q$ employs DM to generate trajectory data used for learning. (ii) In the second part, we transit to normal learning by target policy $p_\theta$. As explained in Section IV-A, we employ progressive RTA-reward in Eq. (18) and baseline reduction in Eq. (21). For each taskset $\{\tau\}$, $p_\theta$ samples its trajectory to calculate returns ($R$, $R_t^{\text{aux}}$ using Eq. (16), (18)) and gradient estimates (in Eq. (20)). Baseline policy $p_\beta$ samples its trajectory in the same way except that it exploits greedy sampling. Unlike stochastic sampling that is generally used for model training, greedy sampling selects such a task with the largest $p_\beta[\pi_t|s_t]$ value for each $t$. The calculation outputs are all combined in Eq. (26), and are used for updating the model parameters $\theta$. Regarding the transition time from guided learning to normal learning, we empirically set it to be after training with 100 batches, since the model is ready to infer schedulable priority assignments at that time. The algorithm complexity is $O(\frac{|\mathcal{D}|}{|\mathcal{B}|}n^2\,d^2)$ where $|\mathcal{D}|$ and $|\mathcal{B}|$ denote the size of a batch and a training data set respectively, and $d$ is the size of embedding data.

## V. EVALUATION

In this section, for evaluating **Panda**, we present the experiment result with respect to the following questions.

- **RQ1.** Can the RL-based priority assignment model **Panda** perform comparatively to (or outperform) existing algorithms for GFPS in terms of schedulability performance?
- **RQ2.** If so, to what extent do the RL training schemes of **Panda** affect its performance?
- **RQ3.** Does **Panda** operate in a reasonable amount time?

### A. IMPLEMENTATION AND EXPERIMENT SETTING

Here, we describe our implementation and experiment settings, including taskset data generation, evaluation metric, heuristic algorithms, and our model implementation with hyperparameters and optimizers.

**TABLE 3. The generation rules for task parameters.**

| Task Parameter | Generation Rule |
|---|---|
| $T_i$ | $\lfloor \exp(\text{Uniform}(\log T_{\min},\ \log T_{\max})) \rfloor$ |
| $C_i$ | $\lfloor T_i U_i \rfloor$ |
| $D_i$ | $T_i = D_i$ for implicit deadline |

#### 1) TASKSET GENERATION

To generate taskset data samples, we exploit the *Randfixedsum* algorithm [28], which has been used widely in research of scheduling problems [29]–[31]. For each taskset $\{\tau\}$, we configure the number of tasks $n$ and (total) taskset utilization $u$. The *Randfixedsum* algorithm randomly selects utilization of $U_i$ for each task $\tau_i \in \{\tau\}$, i.e., $u = \sum_{i=1}^{n} U_i$. For each task $\tau_i$, the algorithm then generates a set of task parameter samples each of which follows the rules in Table 3, yielding values for $T_i$, $C_i$, and $D_i$ under the predetermined utilization $U_i$. These parameters are all given non-negative integers by the floor function, as shown in the table.

#### 2) EVALUATION METRIC

We use the schedulability ratio as our performance metric, which represents the ratio between the number of schedulable tasksets and the number of tested tasksets. Recall that a schedulable taskset indicates that a priority assignment returned by an algorithm passes the given schedulability test. Specifically, we have

$$\text{Schedulability Ratio} = \sum_{\{\tau\}\in\mathcal{D}} \textbf{Test}(\pi, \{\tau\}) \;/\; |\mathcal{D}| \qquad (27)$$

where $\pi$ denotes the priority assignment by a specific algorithm or model for $\{\tau\}$ (e.g., $\pi$ = **Panda**$(\{\tau\})$ in case of evaluating **Panda**), and $\mathcal{D}$ denotes a set of taskset samples. Note that a higher schedulability ratio explicitly leads to more coverage on real-time tasksets to be successfully scheduled in practice.

#### 3) HEURISTIC ALGORITHMS IN COMPARISON

For comparison, we need to choose a schedulability test for GFPS, that determines whether a taskset is deemed schedulable by GFPS with a priority assignment. We target a schedulability test called RTA-LC [8], [11], which has been known to perform superior than the others in terms of covering schedulable tasksets. For heuristic priority assignment policies, we consider and implement DM, DkC [13], and DM-DS [14], [15], as explained in Section II-A. We then compare RTA-LC associated with the priority assigned by **Panda**, with that assigned by the heuristic priority assignment policies. Also, we compare the former with another type of studies mentioned in Section II-A, which is the optimal priority assignment (OPA) technique [18], [19] associated with the state-of-the-art OPA-compatible test, DA-LC [11].

**TABLE 4. Hyperparameter setting.**

| | Description | Default Setting |
|---|---|---|
| Data Set | Num. of training samples | $200K$ |
| | Num. of validation samples | $5K$ |
| | Num. of testing samples | $5K$ |
| Network Setting | Num. Attention layers $N$ | 2 |
| | Embedding dimension $d$ | 128 |
| | Parameter initialization | $\text{Uniform}(-\frac{1}{\sqrt{d}}, \frac{1}{\sqrt{d}})$ |
| Model Fitting | Batch size | 256 |
| | Learning rate | $10^{-4}$ |
| | Epoch | 30 with early stopping |
| | Gradient Clipping | $\lvert\nabla\theta\rvert \leq 1.5$ |
| | Learning rate decay | 0.9 per epoch |
| | Optimizer | Adam |

#### 4) MODEL IMPLEMENTATION

Our **Panda** model implementation is based on Python v3.6 and PyTorch v1.2 [32]. Table 4 lists the hyperparameter

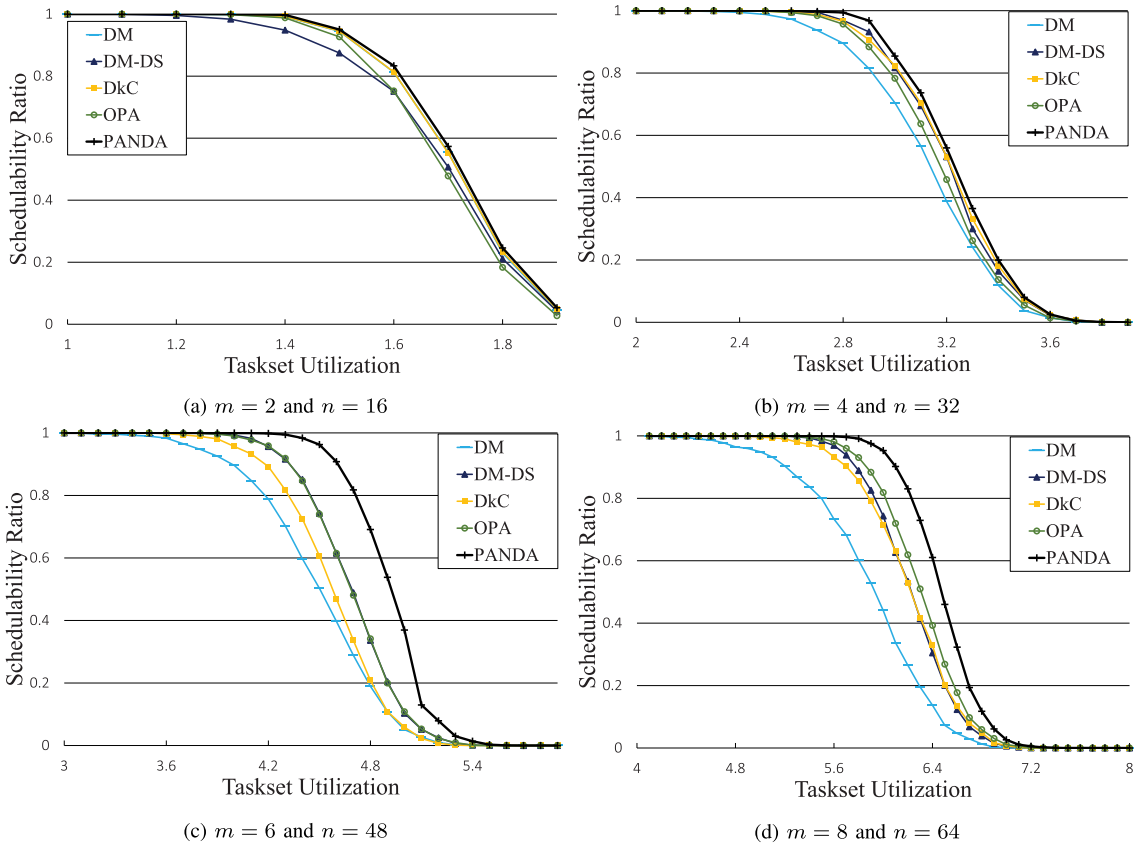


**FIGURE 4.** Schedulability ratio with respect to different taskset utilization settings for *implicit* deadlines: *m* denotes the number of processors and *n* denotes the number of tasks. For example, in (c), when the system configuration of 48-sized tasksets (*n*) and a 6-processor platform (*m*) is given, Panda finds schedulable priority assignments for 96.2% of test tasksets of utilization 4.5 (*u*) while the best-performance algorithm in comparison, OPA finds 74%.

settings for model training, where the Adam optimizer [33] is used. We train and test the models on a system of an Intel(R) Core(TM) i9-9940X processor with an NVIDIA RTX 2080 GPU. In addition, we implement the heuristic algorithms and schedulability tests using Cython [34]. We empirically observe that model training with specific taskset samples of the same (taskset) utilization range shows rapid convergence and stable inference performance. Therefore, we combine multiple models as an integrated solution, where each model is responsible for a specific utilization range, e.g., one model for utilization range [0, 0.9], another for (0.9, 1.9], and so on. This ensemble-like structure works in a real testing environment because all the task parameters including the utilization are explicitly given for the GFPS problem. Thus, a taskset can be properly redirected toward a specific model according to its utilization.

### B. PERFORMANCE COMPARISON WITH HEURISTICS [RQ1]

Figure 4 represents the overall performance of our model and the other algorithms for an *implicit* deadline platform, where the X-axis denotes the utilization of a taskset and the Y-axis denotes the schedulability ratio.

- DM, DM-DS, DkC are the heuristic priority assignment algorithms with the high-performance RTA-LC schedulability test.
- OPA is the optimal priority assignment technique with the OPA-compatible DA-LC test.
- PANDA is our RL-based priority assignment model.

In these tests, the number of testing taskset samples is 5K for each test configuration of $m$-processors and $n$-sized tasksets with the utilization $u$ (e.g., 8-processors and 64-sized tasksets with utilization 6.4), and the schedulability ratio in Eq. (27) for 5K samples is measured for our model and each compared algorithm. The graphs in Figure 4 shows specific test scale configurations, i.e., $m$-processors and $n$-sized tasksets where $m = 2, 4, 6, 8$ and $n = 16, 32, 48, 64$.

As shown, our model demonstrates competitive performance with the other algorithms for all test configurations. Our model often outperforms the other algorithms for cases of large $n$-sized tasksets. For example, our model achieves improvements of 13% and 7.7% on average for configurations of $m = 6$ and $n = 48$ (in Figure 4(c)) and $m = 8$ and $n = 64$ (in Figure 4(d)) respectively, compared to the best-performance algorithm, OPA (specifically, OPA with DA-LC). In this comparison, we focus on the non-trivial utilization ranges where the schedulability ratio of the

heuristic algorithms in the comparison is not 1; e.g., for configurations of $m = 6$ and $n = 48$, the utilization range $[3, 6)$ is given for comparison purposes.

This result demonstrates that our model is capable of scaling well by exploiting the ability of an RL-based approach that is able to efficiently search on a large-scale problem space. The configuration of large-scale tasksets produces complicated GFPS problems, providing opportunity to our model and other machine learning-based scheduling approaches for improving schedulability performance.

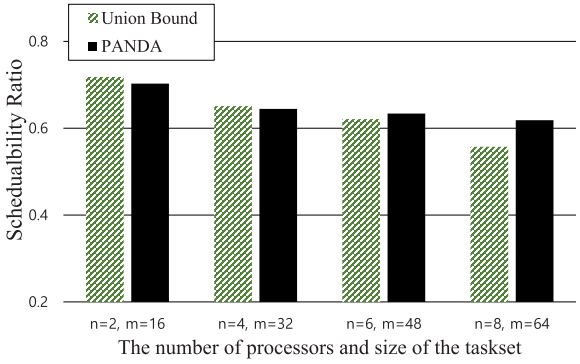


**FIGURE 5.** Panda vs the union of the heuristic algorithms in comparison.

In Figure 5, we compare our model performance with the union performance (Union Bound in the figure) of all the algorithms compared in Figure 4. In measuring the union performance, we classify a taskset to be scheduled if it is schedulable by *at least* one of the algorithms in the comparison, and calculate the average schedulability ratio in the non-trivial utilization range. As observed, **Panda** performs comparatively to the union of all algorithms in most cases, and in particular, shows its superiority to cases of large-scale GFPS settings, e.g., large tasksets of $n = 48,\ 64$, with up to 2∼6% improvement over the approach that collectively uses all the other algorithms.

### C. EVALUATION OF TRAINING OPTIMIZATION SCHEMES [RQ2]

Our model **Panda** is based on encoder-decoder neural networks and its training is driven by RL with two optimization schemes, *progressive RTA-rewards* and *guided learning*, which are custom-tailored for the GFPS problem. In Figure 6, we show the utility of these optimization schemes by comparing the learning curves of the variants of **Panda** over time, where the X-axis denotes the number of training samples, and the Y-axis denotes the schedulability ratio.

- `BASIC` is the plain encoder-decoder network model.
- `RTA` is the model with the progressive RTA-rewards scheme.
- `GUIDE` is the model with the guided learning scheme.
- PANDA is our model with both schemes.

Indeed, it is difficult for us to train the `BASIC` model successfully (due to sparse rewards, as explained previously),

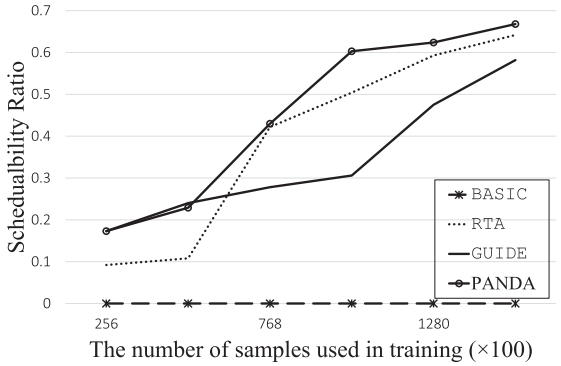


**FIGURE 6.** Learning curves of Panda variants: $m = 6$, $n = 48$, and utilization range [4, 4.9].

and its learning curve remains around zero. It is interesting to compare the learning curves of the `GUIDE` and `RTA` models. As shown, the `GUIDE` model shows relatively better performance than the `RTA` model at the early stage of training, while the `RTA` model shows relatively better performance than the `GUIDE` model as more samples are used for training. This is because the guided learning scheme is explicitly related to the early stage of training, but the effect of the progressive RTA-reward scheme spans the entire training timeline.

As expected, both `GUIDE` and `RTA` perform worse than **Panda**; however, the gap between **Panda** and `RTA` decreases as more samples are used for training. From this observation, one might think that the guided learning becomes less effective with large sample numbers. However, in the absence of the guided learning support, we also observed that our model training often failed to learn at all. Note that the learning curve of `RTA` that we have here was found after many trials.

### D. INFERENCE TIME WITH RESPECT TO SAMPLING SIZE [RQ3]

In terms of model inferences, our model generates $k$-multiple priority assignments for each taskset. It can be done by sampling different tasks from the probability distributions that the model iteratively produces during the priority decoding process. There exists a trade-off between achieved schedulability ratio and required inference time depending on the size of $k$. Figure 7 illustrates such a trade-off where the X-axis represents the sampling size $k$ as well as the inference time (e.g., the bar on 32 and $0.28s$ indicates the case of sampling size $k = 32$ and average inference time $0.28s$) and the Y-axis represents the schedulability ratio. In the figure, PANDA-G denotes a variant of **Panda** that yields only a single priority assignment by the greedy strategy, and PANDA denotes our model with different $k$ settings. Note that the cases of PANDA with $k = 1$ and PANDA-G are different. Although both yield a single priority assignment, the former conducts sampling according to the distributions given by the priority decoder, while the latter

performs greedy selection. Obviously, we achieve improved performance along with larger $k$, by trading-off the inference time. For example, by increasing sampling size $k$ from 2 to 128, the schedulability ratio of our model improves from 53% to 58% and the inference time increases from 0.08$s$ to 1.01$s$, as in Figure 7.

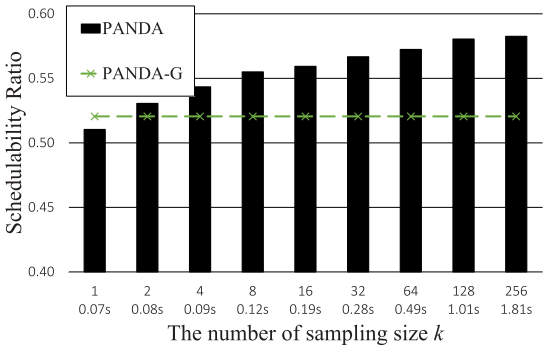


**FIGURE 7.** Trade-off of the size of sampling and inference time: $m = 4$, $n = 32$, and utilization 3.0.

It is possible for our model running on an ordinary PC system to maintain the inference time around 1$s$ with $k = 128$. Thus, we normally set $k = 128$ in our tests. This inference time required for processing a single taskset by our model is evaluated to be sufficient because GFPS is considered an offline problem. It is also feasible to apply parallel processing on the model inferences, thereby increasing $k$ and the performance.

## VI. RELATED WORKS

Many efforts have been made to solve combinatorial optimization problems using neural networks. In [35], the Hopfield neural network was introduced to address TSP with a small number of points. The Hopfield network was later adopted for job-shop scheduling [36], [37]. Until recently, however, this approach was considered not-fully-investigated in that it requires reconstruction of a model for every new instance and it might have difficulty in solving large instance problems.

Recently, the Pointer network was presented as a well-structured method to adopt deep neural networks and several application examples in [1]. The Pointer network model was trained in a supervised manner with input-output samples, producing a permutation as output for a given input sequence. The work in [2] extended the Pointer network to adopt an RL algorithm, i.e., policy gradient Actor-Critic. While we also use the policy gradient algorithm for training, we employ the attention mechanism with several optimization schemes on a Pointer network variant for the GFPS problem.

The recent advance of RL has expedited automation of system operations in many areas. They include energy optimization in data centers [38], [39], cluster resource management [5]–[7], job placement in cloud networks [40], network slicing [41], and compiler optimization [42]. In this paper, we adopt RL for the GFPS problem, which is considered challenging in the area of real-time systems. Liu and Layland [43] showed that rate monotonic (RM) is optimal for single processor fixed-priority scheduling. Unlike the case of a single processor, however, no optimal solution for GFPS has been known for a multi-processor platform [12].

Several heuristic algorithms have been introduced for GFPS upon a multi-processor. As explained in Section II-A, they can be categorized into heuristic algorithms with a high-performance schedulability test and OPA with an OPA-compatible low-performance schedulability test. We summarize those algorithms and schedulability tests in Table 5, and refer to the survey paper [12] for more related information. We evaluate our RL-based model through comparison with those heuristic algorithms in terms of schedulability ratio.

**TABLE 5.** Summary of related studies on GFPS.

| Name | Description |
|---|---|
| DA [17] | Deadline analysis (DA) sched. test |
| RTA [9] | Response time analysis (RTA) sched. test |
| DA-LC [11] | DA test with limited carry-in |
| RTA-LC [8], [11] | RTA test with limited carry-in |
| DM, RM [12] | Deadline monotonic (DM) priority assign. alg.<br>· higher priority for shorter deadline<br>Rate monotonic (RM) priority assign. alg.<br>· higher priority for smaller period |
| DM-DS [14], [15] | DM priority assign. alg. with deadline slack<br>· higher priority for higher util. or smaller slack |
| DkC [13] | Priority assign. alg. using slack time $D_i - kC_i$<br>· $k$ depends on $m$-processors<br>e.g., $k = (m-1)\sqrt{5 * m^2 - 6 * m + 1}$ |
| OPA [18], [19] | Audsley's optimal priority assign. (OPA) alg.<br>· only with OPA-compatible sched. test<br>· choosing a task and check the sched. test, for all priority-unassigned tasks |

To the best of our knowledge, our work is the first to adopt RL for the GFPS problem (or any real-time scheduling problem).

## VII. CONCLUSION

In this paper, we presented **Panda**, the RL-based priority assignment model for multi-processor real-time scheduling (i.e., GFPS). The model is based on encoder-decoder neural networks with attention, and its training is driven by RL. In doing so, we presented training optimization schemes, progressive RTA-reward and DM-based guided learning, which are tailored for the GFPS problem. Our model shows robust performance in terms of schedulability ratio, compared to the existing heuristics and their integrated approach (i.e., union). This result illustrates the applicability of RL-based approaches to GFPS or other similar task scheduling problems.

Our future work will aim to develop a cost-efficient neural network model for complex combinatorial problems. In addition, we are interested in applying **Panda** to other scheduling problems in addition to real-time systems.

## REFERENCES


[1] O. Vinyals, M. Fortunato, and N. Jaitly, "Pointer networks," in *Proc. Conf. Neural Inf. Process. Syst.*, 2015, pp. 2692–2700.

[2] I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio, "Neural combinatorial optimization with reinforcement learning," 2016, *arXiv:1611.09940*. [Online]. Available: http://arxiv.org/abs/1611.09940

[3] W. Kool, H. van Hoof, and M. Welling, "Attention, learn to solve routing problems!" in *Proc. 7th Int. Conf. Learn. Represent. (ICLR)*, 2019, pp. 1–25.

[4] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proc. Conf. Neural Inf. Process. Syst.*, 2017, pp. 5998–6008.

[5] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, "Resource management with deep reinforcement learning," in *Proc. 15th ACM Workshop Hot Topics Netw.*, 2016, pp. 50–56.

[6] H. Mao, M. Schwarzkopf, S. B. Venkatakrishnan, Z. Meng, and M. Alizadeh, "Learning scheduling algorithms for data processing clusters," in *Proc. ACM Special Interest Group Data Commun.*, Aug. 2019, pp. 270–288.

[7] M. Cheong, H. Lee, I. Yeom, and H. Woo, "SCARL: Attentive reinforcement learning-based scheduling in a multi-resource heterogeneous cluster," *IEEE Access*, vol. 7, pp. 153432–153444, 2019.

[8] N. Guan, M. Stigge, W. Yi, and G. Yu, "New response time bounds for fixed priority multiprocessor scheduling," in *Proc. 30th IEEE Real-Time Syst. Symp.*, Dec. 2009, pp. 387–397.

[9] M. Bertogna and M. Cirinei, "Response-time analysis for globally scheduled symmetric multiprocessor platforms," in *Proc. 28th IEEE Int. Real-Time Syst. Symp. (RTSS)*, Dec. 2007, pp. 149–160.

[10] R. I. Davis and A. Burns, "Priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems," in *Proc. 30th IEEE Real-Time Syst. Symp.*, Dec. 2009, pp. 398–409.

[11] R. I. Davis and A. Burns, "Improved priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems," *Real-Time Syst.*, vol. 47, no. 1, pp. 1–40, Jan. 2011.

[12] R. I. Davis and A. Burns, "A survey of hard real-time scheduling for multiprocessor systems," *ACM Comput. Surv.*, vol. 43, no. 4, p. 35, 2011.

[13] B. Andersson and J. Jonsson, "Fixed-priority preemptive multiprocessor scheduling: To partition or not to partition," in *Proc. 7th Int. Conf. Real-Time Comput. Syst. Appl.*, 2004, pp. 337–346.

[14] B. Andersson, S. Baruah, and J. Jonsson, "Static-priority scheduling multiprocessors," in *Proc. 22nd IEEE Real-Time Syst. Symp. (RTSS)*, 2001, pp. 193–202.

[15] B. Andersson, "Global static-priority preemptive multiprocessor scheduling with utilization bound 38%" in *Proc. 12th Int. Conf. Princ. Distrib. Syst. (OPODIS)*, vol. 5401, 2008, pp. 73–88.

[16] R. I. Davis, L. Cucu-Grosjean, M. Bertogna, and A. Burns, "A review of priority assignment in real-time systems," *J. Syst. Archit.*, vol. 65, pp. 64–82, Apr. 2016.

[17] M. Bertogna, M. Cirinei, and G. Lipari, "Schedulability analysis of global scheduling algorithms on multiprocessor platforms," *IEEE Trans. Parallel Distrib. Syst.*, vol. 20, no. 4, pp. 553–566, Apr. 2009.

[18] N. C. Audsley, *Optimation Priority Assignment Feasibility Static Priority Tasks With Arbitrary Start Times*. Princeton, NJ, USA: Citeseer, 1991.

[19] N. C. Audsley, "On priority assignment in fixed priority scheduling," *Inf. Process. Lett.*, vol. 79, no. 1, pp. 39–44, May 2001.

[20] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Mach. Learn.*, vol. 8, nos. 3–4, pp. 229–256, May 1992.

[21] M. Bellemare, S. Srinivasan, G. Ostrovski, T. Schaul, D. Saxton, and R. Munos, "Unifying count-based exploration and intrinsic motivation," in *Proc Conf. Neural Inf. Process. Syst. (NIPS)*, 2016, pp. 1471–1479.

[22] T. Degris, M. White, and R. S. Sutton, "Linear off-policy actor-critic," in *Proc. 29th Int. Conf. Mach. Learn. (ICML)*, 2012, pp. 1–6.

[23] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proc. Int. Conf. Mach. Learn.*, vol. 37. Jul. 2015, pp. 448–456.

[24] J. Lei Ba, J. Ryan Kiros, and G. E. Hinton, "Layer normalization," 2016, *arXiv:1607.06450*. [Online]. Available: http://arxiv.org/abs/1607.06450

[25] E. Jang, S. Gu, and B. Poole, "Categorical reparameterization with gumbel-softmax," 2016, *arXiv:1611.01144*. [Online]. Available: http://arxiv.org/abs/1611.01144

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

[27] A. R. Mahmood, H. van Hasselt, and R. S. Sutton, "Weighted importance sampling for off-policy learning with linear function approximation," in *Proc Conf. Neural Inf. Process. Syst.*, 2014, pp. 3014–3022.

[28] P. Emberson, R. Stafford, and R. I. Davis, "Techniques for the synthesis of multiprocessor tasksets," in *Proc. 1st Int. Workshop Anal. Tools Methodologies Embedded Real-Time Syst. (WATERS)*, 2010, pp. 6–11.

[29] A. Wieder and B. B. Brandenburg, "Efficient partitioning of sporadic real-time tasks with shared resources and spin locks," in *Proc. 8th IEEE Int. Symp. Ind. Embedded Syst. (SIES)*, Jun. 2013, pp. 49–58.

[30] B. B. Brandenburg and M. Gul, "Global scheduling not required: Simple, near-optimal multiprocessor real-time scheduling with semi-partitioned reservations," in *Proc. IEEE Real-Time Syst. Symp. (RTSS)*, Nov. 2016, pp. 99–110.

[31] A. Gujarati, F. Cerqueira, and B. B. Brandenburg, "Multiprocessor real-time scheduling with arbitrary processor affinities: From practice to theory," *Real-Time Syst.*, vol. 51, no. 4, pp. 440–483, Jul. 2015.

[32] A. Paszke, "Pytorch: An imperative style, high-performance deep learning library," in *Proc. Conf. Neural Inf. Process. Syst.*, 2019, pp. 8024–8035.

[33] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*. [Online]. Available: http://arxiv.org/abs/1412.6980

[34] S. Behnel, R. Bradshaw, C. Citro, L. Dalcin, D. S. Seljebotn, and K. Smith, "Cython: The best of both worlds," *Comput. Sci. Eng.*, vol. 13, no. 2, pp. 31–39, Mar. 2011.

[35] J. J. Hopfield and D. W. Tank, "œneural computation of decisions in optimization problems," *Biol. Cybern.*, vol. 52, no. 3, pp. 141–152, 1985.

[36] Y. S. Foo and Y. Takefuji, "Integer linear programming neural networks for job-shop scheduling," in *Proc. Int. Conf. Neural Netw. (ICNN)*, 1988, pp. 341–348.

[37] Y. S. Foo and Y. Takefuji, "Stochastic neural networks for solving job-shop scheduling. i. problem representation," in *Proc. Int. Conf. Neural Netw. (ICNN)*, 1988, pp. 275–282.

[38] A. Beloglazov and R. Buyya, "Energy efficient resource management in virtualized cloud data centers," in *Proc. 10th IEEE/ACM Int. Symp. Cluster, Cloud Grid Comput. (CCGrid)*, 2010, pp. 826–831.

[39] X. Li, Z. Qian, S. Lu, and J. Wu, "Energy efficient virtual machine placement algorithm with balanced and improved resource utilization in a data center," *Math. Comput. Model.*, vol. 58, no. 5, pp. 1222–1235, 2013.

[40] Y. Bao, Y. Peng, and C. Wu, "Deep learning-based job placement in distributed machine learning clusters," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, 2019, pp. 505–513.

[41] D. Bega, M. Gramaglia, M. Fiore, A. Banchs, and X. Costa-Pérez, "Deepcog: Cognitive network management in sliced 5g networks with deep learning," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, 2019, pp. 280–288.

[42] T. Chen, T. Moreau, Z. Jiang, L. Zheng, E. Q. Yan, H. Shen, M. Cowan, L. Wang, Y. Hu, L. Ceze, C. Guestrin, and A. Krishnamurthy, "TVM: An automated end-to-end optimizing compiler for deep learning," in *Proc. 13th USENIX Symp. Oper. Syst. Des. Implement. (OSDI)*, 2018, pp. 578–594.

[43] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *J. ACM*, vol. 20, no. 1, pp. 46–61, 1973.


**HYUNSUNG LEE** was born in Seoul, South Korea. He received the B.S. degree in computer engineering from Sungkyunkwan University, Suwon, South Korea, in 2019, where he is currently pursuing the master's degree in electrical and computer engineering. His research interests include cluster orchestration and reinforcement learning.

**JINKYU LEE** (Member, IEEE) received the B.S., M.S., and Ph.D. degrees in computer science from the Korea Advanced Institute of Science and Technology (KAIST), South Korea, in 2004, 2006, and 2011, respectively. He was a Visiting Scholar/Research Fellow of the Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI, USA, from 2011 to 2014. He is an Associate Professor with the Department of Computer Science and Engineering, Sungkyunkwan University (SKKU), South Korea, where he joined in 2014. His research interests include system design and analysis with timing guarantees, QoS support, and resource management in real-time embedded systems, mobile systems, and cyber-physical systems. He won the Best Student Paper Award from the 17th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS) in 2011 and the Best Paper Award from the 33rd IEEE Real-Time Systems Symposium (RTSS) in 2012.

**IKJUN YEOM** received the B.S. degree in electronics engineering from Yonsei University, Seoul, South Korea, in February 1995, and the M.S. and Ph.D. degrees in computer engineering from Texas A&M University, in August 1998 and May 2001, respectively. He worked at DACOM Company from 1995 to 1996 and Nortel Networks in 2000. He was an Associate Professor with the Department of Computer Science, Korea Advanced Institute of Science and Technology (KAIST), from 2002 to 2008. He is currently a Full Professor with the Department Computer Science and Engineering, Sungkyunkwan University, Suwon, South Korea. His research interests include AQM, congestion control, TCP, wireless networks, and the future Internet architecture.

**HONGUK WOO** (Member, IEEE) was born in Seoul, South Korea. He received the B.S. degree in computer science from Korea University, Seoul, in 1995, and the M.S. and Ph.D. degrees in computer sciences from The University of Texas at Austin, Austin, TX, USA, in 2002 and 2008, respectively.

From 2008 to 2018, he worked with Samsung Research, Samsung Electronics, as a Principal Engineer and the Vice President. Since 2018, he has been an Assistant Professor with the Department of Computer Science and Engineering, Sungkyunkwan University, Suwon, South Korea. He has coauthored more than 30 research articles and holds ten patents. His research interests include intelligent applications, analytic monitoring, cloud computing, and networked cyber-physical systems.