Work-in-Progress: Learning to Refine Priority Assignment in Fixed-Priority Real-Time Scheduling

Binqi Sun*, Linghan Fang*, Andrea Bastoni, and Marco Caccamo
Technical University of Munich, Germany
Email: {binqi.sun, linghan.fang, andrea.bastoni, mcaccamo}@tum.de

***Abstract*—We address the problem of priority assignment for global fixed-priority scheduling on multicore real-time systems, where identifying a feasible priority ordering is a combinatorial challenge. We propose a learning-based framework that trains a lightweight policy network via reinforcement learning to refine existing priority assignments toward schedulable solutions. Based on the policy network, we propose an *inference-time policy refinement* mechanism that improves schedulability without additional training. It combines *breadth sampling*—generating candidate orderings via stochastic perturbations—with *depth refinement*, which iteratively enhances promising candidates. A continuous reward function based on a schedulability hazard metric enables effective training. Preliminary experiments show that the proposed method performs better than classical heuristics such as Deadline Monotonic and DkC, demonstrating its potential as an effective learning-assisted approach to real-time scheduling.**

***Index Terms*—Reinforcement Learning, Priority Assignment, Real-Time Scheduling, Global Scheduling**

I. Introduction

Task scheduling lies at the heart of virtually all computing systems. From general-purpose operating systems to cloud orchestration platforms, the ability to allocate limited computational resources efficiently is a foundational requirement. For the class of systems subject to real-time constraints imposed by the physical world, correctness itself becomes time-sensitive. A task that completes after its deadline is not merely delayed but functionally invalid, and such a delay may have hazardous consequences for the system. Consequently, scheduling strategies must provide predictable temporal behavior. Fixed-priority scheduling is widely used in practice due to its simplicity and theoretical grounding [1], where each task is assigned a static priority and the scheduler always dispatches the highest-priority ready task.

The widespread shift toward multicore architectures has, however, introduced new layers of complexity to this well-understood paradigm [2]. We focus on global fixed-priority (gFP) scheduling, where all processors share a single priority queue and tasks can migrate between cores [3]. While gFP offers flexibility and improved processor utilization, it also introduces subtle timing interference that complicates schedulability analysis [4]. For a given set of N sporadic tasks, determining whether a priority assignment ensures all tasks meet their deadlines is a combinatorial challenge. The number of possible priority orderings scales as $N!$, rendering exhaustive search infeasible even for moderately sized task sets. This factorial growth of the solution space necessitates intelligent methods that can efficiently navigate the vast and irregular landscape of possible priority assignments.

To address this challenge, researchers have developed a range of methods. Traditional heuristic methods, such as Deadline-Monotonic Priority Ordering (DMPO) [1], are efficient but rely on hand-crafted rules that often fail in complex or high-utilization scenarios [2]. More recently, supervised learning approaches like the Pointer Attention Learner [5] have shown promise by learning to predict schedulable assignments from data. However, these methods depend heavily on the quality and diversity of labeled training data, such as prior-known optimal solutions. Reinforcement learning (RL) has emerged as a compelling alternative that avoids the reliance on labeled datasets by learning to make scheduling decisions through direct interactions with the system environment [6]. To the best of our knowledge, Panda [7] is the only existing RL-based method for gFP priority assignment. Panda constructs task priorities in an iterative manner—selecting one task at a time until a complete ordering is formed. While this formulation allows direct learning from schedulability feedback, it produces each priority assignment in a single pass and lacks a mechanism to refine or improve an existing ordering. In practice, however, small changes in task ordering can significantly affect schedulability, and the ability to progressively refine a candidate solution offers a promising opportunity to improve scheduling performance without additional training.

Building on this insight, we propose a new learning-based priority assignment framework. Our framework uses RL to train a policy network that accepts an existing priority ordering as input and outputs a refined assignment, enabling an *inference-time policy refinement* mechanism. By repeatedly applying the trained policy, the model can iteratively enhance the quality of the priority ordering through a combination of *breadth sampling* and *depth refinement*. The framework can also be applied to improve priority assignments generated by other existing methods by using them as initial priority inputs. Additionally, to overcome the issue of sparse and binary feedback, we design a continuous reward function based on a schedulability hazard metric, providing dense and informative learning signals. Furthermore, we employ a lightweight one-dimensional convolutional architecture that captures inter-task relationships with low computational overhead. Although, at

*These authors contributed equally to this work.

Marco Caccamo was supported by an Alexander von Humboldt Professorship endowed by the German Federal Ministry of Education and Research.

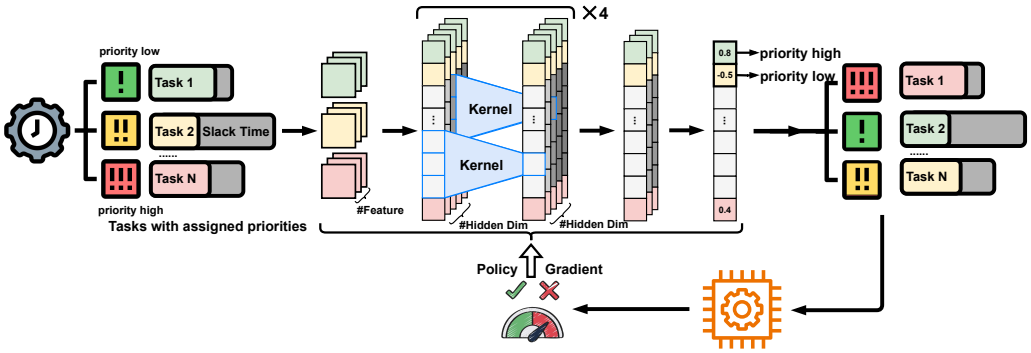


Fig. 1: Overview of the policy network design.

this time, we cannot directly compare against Panda [7] due to the lack of publicly available implementation, our preliminary experiments demonstrate the effectiveness of the proposed approach by comparing it with heuristic baselines across different utilization levels.

II. Problem Description

A. System Model

We consider a system of N sporadic real-time tasks, $\tau = \{\tau_1, \tau_2, \ldots, \tau_N\}$, scheduled on M identical processors. Each task τ_i is defined by a tuple $\tau_i = \{C_i, D_i, T_i\}$, where C_i is the worst-case execution time (WCET), D_i is the relative deadline, and T_i is the minimum inter-arrival time (period). We assume a constrained deadline model, where $D_i \leq T_i$ for all tasks. The tasks are allowed to execute on any processor and migrate from one processor to another. The system utilization is defined as $U = \frac{1}{M}\sum_{\tau_i \in \tau} U_i$, where $U_i = C_i/T_i$.

The system employs a preemptive global fixed-priority (gFP) scheduling policy that assigns each task a fixed priority offline. At runtime, all ready jobs are maintained in a single, global ready queue, and the scheduler executes the M highest-priority ready jobs in each time slot on the M processors. Without loss of generality, we assume tasks are indexed by priority, such that τ_i has higher priority than τ_j if $i < j$.

B. Schedulability Test

The schedulability problem is to determine whether a given task set τ can meet all its deadlines under a specific priority assignment. A task set is schedulable if the worst-case response time (WCRT) R_i of every task τ_i is less than or equal to its deadline, *i.e.*, $\forall i, R_i \leq D_i$.

To verify schedulability, we use response-time analysis (RTA) [4]. For a task τ_k, its WCRT is found by iteratively solving the following fixed-point equation for the smallest $R_k > 0$:

$$R_k = C_k + \left\lfloor \frac{1}{M} \sum_{i<k} I_{k \leftarrow i}(R_k) \right\rfloor,$$

where $I_{k \leftarrow i}(x)$ represents the maximum interference generated by a higher-priority task τ_i within any time interval of length x. In this work, we calculate $I_{k \leftarrow i}(x)$ using the upper bound derived in [4]. If the equation converges to a value $x \leq D_k$, task τ_k is schedulable. The entire set is schedulable if this condition holds for all tasks.

C. Priority Assignment Problem

The priority assignment problem is to find a priority ordering for the tasks in set τ that results in a schedulable system. Given that the schedulability of a task set under gFP scheduling is highly dependent on the priority ordering, the objective is to find a permutation of task priorities that satisfies the schedulability test.

Formally, the problem is to find a *priority assignment* $P : \tau \to \{P_1, P_2, \ldots, P_N\}$, where $\{P_1, P_2, \ldots, P_N\}$ forms an *ordered permutation* over the task set τ. Each element P_i represents the task index in τ that is assigned to the i-th highest priority. For example, $P_2 = 3$ means assigning τ_3 to the second-highest priority in priority assignment P. When the schedulability test is applied to the prioritized task set, the condition $R_i \leq D_i$ should hold for all $\tau_i \in \tau$.

III. Learning-Based Priority Assignment

A. Policy Network Design

We design a one-dimensional convolutional neural network (CNN) as the policy model, illustrated in Figure 1. Each task in a given set is represented as a feature vector capturing its essential parameters, including execution time, period, deadline, utilization, slack time, and the task's relative position within the system based on period ranking. Additionally, a system-level parameter, the number of available processors, is included as a global feature for all tasks. These feature vectors are arranged into a matrix, where each row corresponds to one task feature vector, and the rows are ordered by the input task priorities (*i.e.*, the i-th row is the feature vector of the task with the i-th priority in the input priority assignment).

The CNN processes this sequence through multiple 1D convolutional layers. This architecture is chosen to effectively capture local interactions between tasks that are close in the priority space. The network's final layer outputs a scalar score, s_i, for each task. These scores represent the policy's learned estimate of each task's priority urgency; a higher score indicates a preference for a higher priority. This vector of scores is then used to define a stochastic policy over the space of all possible permutations.

B. Policy Training via Reinforcement Learning

The policy network is trained using the episodic REINFORCE algorithm [8], which optimizes the network's parameters θ to maximize the expected reward. Starting from the initial (untrained) priority assignment, our training procedure integrates a continuous reward signal and an efficient one-shot sampling mechanism.

Reward Shaping. To provide a dense and informative learning signal, we design a reward function based on the *system hazard* metric: $h(\tau^P) = \max_i R_i/D_i$, where τ^P is the task set ordered by priority assignment P, R_i is the worst-case response time (WCRT) of task τ_i^P, and D_i is its relative deadline. A schedule is feasible if and only if its hazard satisfies $h(\tau^P) \leq 1$, which means that every task meets its deadline, i.e., $R_i \leq D_i$ for all $\tau_i \in \tau^P$.

We transform this hazard value into a reward in the range $[-1, 1]$ using the following sigmoid-like function:

$$H(\tau^P) = \frac{2}{1 + e^{4(h(\tau^P)-1)}} - 1.$$

This function is centered at the feasibility boundary ($h = 1$), providing a sensitive gradient that guides the agent not only to find feasible solutions, but also to increase the margin of safety for already feasible ones.

One-Shot Priority Sampling. We adopt the Gumbel-Top-k trick [9] to sample a complete priority ordering from the policy in a single step. By adding *i.i.d.* Gumbel noise to each task score ($\tilde{s}_i = s_i + g_i$, where $g_i \sim \text{Gumbel}(0, 1)$). This method transforms deterministic network outputs into a stochastic ranking; the final priority ordering P is then obtained by sorting the perturbed scores in descending order. The one-shot priority sampling can efficiently genereate a full permutation in one step, without the need for autoregressive selection as in prior RL methods (*e.g.*, [7]).

Policy Gradient Update. With a sampled priority ordering P and its corresponding reward $H(\tau^P)$, the probability of sampling the priority ordering P under the Gumbel-Top-k policy [9] is computed as

$$\pi_\theta(P) = \prod_{i=1}^{N} \frac{\exp(s_{P_i})}{\sum_{i \leq j \leq N} \exp(s_{P_j})},$$

where s_{P_i} denotes the score of the task assigned with the i-th highest priority. Then, the policy network is optimized using REINFORCE [8]: $\nabla_\theta J(\theta) = \mathbb{E}\big[(H(\tau^P) - b)\, \nabla_\theta \log \pi_\theta(P)\big]$, where b is the reward standardization within each batch (*i.e.*, subtracting the mean reward and dividing by its standard deviation) to reduce variance and stabilize training, as widely used in policy-gradient algorithms, *e.g.*, PPO [10].

C. Inference-Time Policy Refinement

Once the policy network is trained, its effectiveness is amplified at test time through three distinct refinement strategies that require no further parameter updates: (a) breadth sampling, (b) depth refinement, and (c) hybrid strategy, as illustrated in Figure 2.

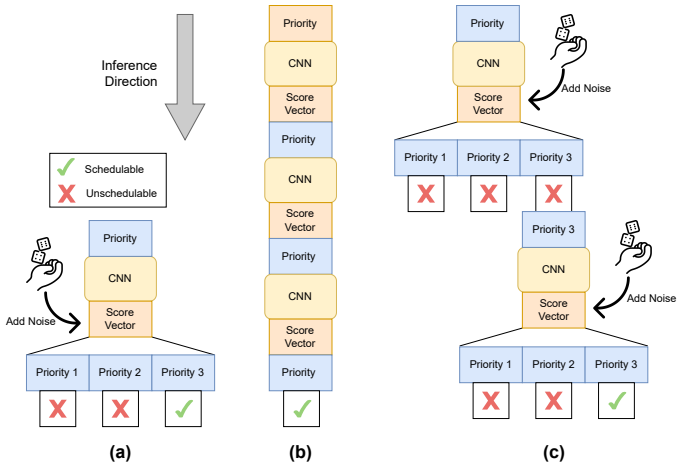


Fig. 2: Inference-time policy refinement strategies.

Breadth Sampling. This strategy emphasizes parallel diversity. It begins with a single forward pass through the CNN to obtain a base score vector. We then generate α diverse candidate orderings by adding α independent Gumbel noises to this single score vector and sorting each result. Because the CNN is performed only once, and the schedulability of each candidate can be verified in parallel, this process explores a wide range of candidates with minimal overhead.

Depth Refinement. This strategy focuses on progressive, iterative improvement. Starting with an initial priority ordering (*e.g.*, from a heuristic or the best candidate from a breadth sampling), we feed it into the CNN without noise to generate a deterministic refined ordering. This new ordering is then used as the input for the next iteration. This process, $P^{(1)} \to P^{(2)} \to \cdots \to P^{(\beta)}$, continues for a fixed number of steps $beta$ or until a feasible solution is found, allowing the model to correct local misorderings and improve the global quality of the schedule.

Hybrid Strategy. This strategy combines the strengths of both breadth sampling and depth refinement. It starts with a breadth sampling to quickly explore a diverse set of promising candidates. If no feasible solution is found, the single best candidate (*i.e.*, the one with the highest reward) is selected. This candidate then serves as the starting point for a depth refinement to exploit its local neighborhood. This hybrid approach balances global exploration with local exploitation, providing a robust and powerful inference procedure.

IV. Preliminary Results

We empirically evaluate our proposed learning-based scheduling method. The experiments are designed to answer the following key research questions:

Q1: How does our learning-based method perform compared to heuristic baselines as system utilization increases?

Q2: How effective is the combination of breadth sampling and depth refinement in improving schedulability?

A. Evaluation Setup

To answer these questions, we compare our method against two widely-used heuristics: Deadline Monotonic Priority Ordering (DMPO) [1] and DkC [11]. Our primary performance metric is the schedulability rate, defined as the percentage of task sets for which a method successfully finds a feasible schedule. Unless stated otherwise, our method employs the *hybrid strategy*, which iteratively refines solutions with a sampling breadth α and a search depth β. In our experiments, we set $\alpha = 100, \beta = 10$ to enable empirical convergence and discuss the effects of these parameters in Section IV-C.

The policy network is trained on 8,192 randomly generated task systems with target system utilizations $U \in [0.4, 0.6]$. First, the total system utilization is distributed across tasks using DRS [12] to obtain per-task utilizations U_i (*i.e.*, $\sum_{\tau_i \in \tau} U_i/M = U$). Task periods are then independently sampled from a log-uniform distribution over $[10, 100]$ ms, and the WCET is computed as $C_i = U_i \cdot T_i$, assuming implicit deadlines ($T_i = D_i$). For each system, we randomly sample the number of tasks $N \in [8, 16]$ and the number of processors $M \in [2, 6]$. The model is trained for 50 epochs on a single NVIDIA A100 GPU, taking approximately 5 hours.

After training, we evaluate on task sets with a larger range of utilizations $[0.05, 0.95]$ with a step of 0.1, to test the generalization capability of the trained policy. For each utilization level, we generate 1024 random task sets following the same task set generation procedure as the training phase.

B. Performance under Varying System Utilization

Figure 3 presents the schedulability rate across different utilizations. The results show that our method achieves the highest schedulability rate. The performance gap widens as the utilization increases. For $U \in [0.60, 0.75]$, our approach successfully schedules an average of nearly 90% of task sets, while DMPO and DkC find feasible priority assignments for 51% and 83% of the same task sets, respectively.

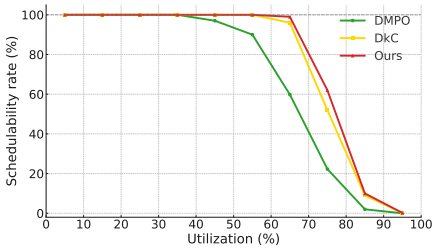
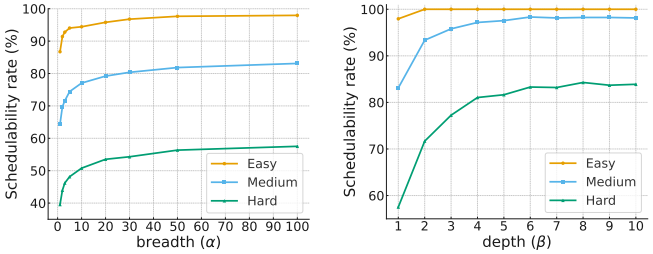


Fig. 3: Schedulability comparison across different utilizations.

C. Effects of Breadth α and Depth β

Finally, we evaluate the contributions of our two main inference strategies: breadth sampling and depth refinement and their corresponding parameters α and β. Figure 4 presents how the schedulability rate varies given different values of α and β. The results show that breadth search provides a significant performance lift when α increases from 1 to 30, and the growth slows down when $\alpha \in [50, 100]$. For depth refinement, high-utilization task sets require a larger depth β to iteratively improve schedulability, while the performance over low-utilization tasks converges with $\beta \leq 6$.



(a) Effect of α under the breadth sampling strategy.

(b) Effect of β under the depth refinement strategy.

Fig. 4: Schedulability rates across various α and depth β. Task sets are categorized into three utilization levels: *Easy* (40-60%), *Medium* (50-70%), and *Hard* (60-75%).

V. Conclusion and Future Work

This work presented a learning-based framework for refining priority assignments in global fixed-priority scheduling. By integrating a RL-trained policy with an inference-time refinement mechanism, the proposed method effectively balances exploration and exploitation to discover feasible schedules without retraining. Future work will focus on generalizing the framework to other system models, such as non-preemptive or parallel tasks, and evaluating its performance against more advanced priority assignment strategies (*e.g.*, [7]).

References

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

[2] R. I. Davis and A. Burns, "A survey of hard real-time scheduling for multiprocessor systems," *ACM Computing Surveys*, vol. 43, no. 4, pp. 35:1–35:44, 2011.

[3] M. Bertogna and M. Cirinei, "Response-time analysis for globally scheduled symmetric multiprocessor platforms," in *IEEE Real-Time Systems Symposium (RTSS)*, 2007, pp. 149–160.

[4] N. Guan, M. Stigge, W. Yi, and G. Yu, "New response time bounds for fixed priority multiprocessor scheduling," in *RTSS*, 2009, pp. 387–397.

[5] S. Lee, H. Baek, H. Woo, K. G. Shin, and J. Lee, "ML for RT: Priority Assignment Using Machine Learning," in *RTAS*, 2021.

[6] M. Theile, B. Sun, and M. Caccamo, "Position paper: deep reinforcement learning for real-time resource management," *Real-Time Systems*, pp. 1–6, 2025.

[7] H. Lee, J. Lee, I. Yeom, and H. Woo, "Panda: Reinforcement learning-based priority assignment for multi-processor real-time scheduling," *IEEE Access*, vol. 8, pp. 185 570–185 583, 2020.

[8] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine learning*, vol. 8, no. 3, pp. 229–256, 1992.

[9] W. Kool, H. Van Hoof, and M. Welling, "Stochastic beams and where to find them: The gumbel-top-k trick for sampling sequences without replacement," in *ICML*, 2019, pp. 3499–3508.

[10] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[11] B. Andersson and J. Jonsson, "Fixed-priority preemptive multiprocessor scheduling: To partition or not to partition," in *RTSS*, 2000, pp. 173–182.

[12] D. Griffin, I. Bate, and R. I. Davis, "Generating utilization vectors for the systematic evaluation of schedulability tests," in *RTSS*, 2020, pp. 76–88.