SCHED: Safe CPU Scheduling Framework with Reinforcement Learning and Decision Trees for Autonomous Vehicles

Dongjoo Seo[1], Changhoon Sung[2], Ping-Xiang Chen[1], Bryan Donyanavard[2], Nikil Dutt[1]
[1] *University of California, Irvine,* [2] *San Diego State University*
{dseo3, p.x.chen, dutt}@uci.edu, {csung7167, bdonyanavard}@sdsu.edu

***Abstract*—Autonomous vehicles (AVs) require consistently low-latency computations; however, operating system (OS) CPU scheduler can lead to high tail latency that threatens timely decision-making and safety. General-purpose OS schedulers prioritize fairness and throughput over individual task deadlines, posing challenges for complex AV workloads with mixed-criticality tasks. Despite extensive research on CPU scheduling, reinforcement learning (RL) has not been explored at the OS level due to feasibility concerns. This paper presents SCHED, the first RL-based CPU scheduling framework optimized for OS integration. SCHED learns a scheduling policy via reinforcement learning and deploys it as a lightweight, decision-tree-based scheduler. We demonstrate that SCHED remains sufficiently fast for OS-level integration compared to existing OS schedulers. Furthermore, we validate SCHED on a realistic autonomous vehicle pipeline, demonstrating its practical potential in maintaining low latency and ensuring safer, more responsive AV operations.**

***Index Terms*—Autonomous Vehicles, CPU Scheduling, Tail Latency, Reinforcement Learning, eBPF**

I. Introduction

Autonomous vehicles (AVs) consist of complex latency-sensitive workloads that require sophisticated operating system (OS) features, posing unique challenges in the design of CPU scheduling policies. Specifically, AVs must simultaneously execute tasks with different timing constraints and priorities, including sensor data processing, object detection, path planning, and control signal generation [1]. General-purpose OS CPU schedulers are typically designed to optimize fairness or overall throughput for a large number of user tasks from multiple applications, rather than strictly enforcing specific task deadlines. As a result, unpredictable delays can directly impact the safety of AV systems, increasing the risk of accidents on the road.

Adaptive CPU scheduling is a potential solution to address unpredictable excessive delays. Reinforcement learning (RL) is a promising approach to dynamically derive optimal scheduling strategies in complex environments. However, applying RL to CPU scheduling presents two critical challenges. First, RL-based CPU scheduling requires extensive training and trial-and-error exploration. However, AV systems cannot tolerate unsafe decisions during runtime or afford real-world validation, making direct deployment impractical [2]. Secondly, complex models face deployment limitations: deep neural networks introduce computational overhead, making real-time execution in the OS impractical. These two challenges necessitate a safer and more efficient RL-based scheduling approach.

This work was partially supported by the National Science Foundation under Grant No. 2348349.

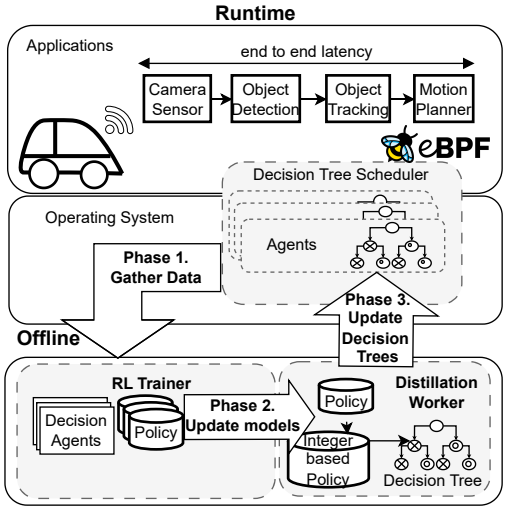


Fig. 1: SCHED Architecture

This paper proposes a RL-based CPU scheduling framework that leverages offline-trained RL agent and policy distillation to address key challenges. First, we construct a reliable expert dataset by recording system states and scheduling decisions from conventional schedulers. The RL agent then trains its policy using this dataset, ensuring a stable initial strategy without unsafe exploration. It further refines its performance while minimizing real-world risks by utilizing offline data or enforcing strict safety constraints. Finally, the trained policy, initially represented as a complex neural network, is distilled into a decision tree for lightweight runtime deployment within the OS. Our evaluation with an AV pipeline scenario demonstrates improved responsiveness compared to existing OS schedulers, highlighting the potential for broader applicability in safety-critical systems.

II. Proposed Framework

Figure 1 shows an overview of the three phase closed-loop approach. In Phase 1, Decision Tree Scheduler (DTS) generates and records scheduling decisions using an existing

CPU scheduler, creating a dataset of expert demonstrations. In Phase 2, RL Trainer trains an RL policy on data from DTS to learn an improved CPU scheduling policy. In Phase 3, Distillation Worker distills the trained policy into a decision-tree representation for deployment on AVs.

For Phase 1, we collect data using DTS, logging CPU scheduling behavior to build a dataset for training RL policies. The three functions our policy replaces are `select_cpu`, `enqueue_task`, and `dispatch_task`. Each function makes an essential scheduling decision: selecting a CPU for task execution, enqueuing a task to either a CPU or a software defined queue, and dispatching a task from a queue to a CPU. Table I shows the state and action of each function. For Phase 2, we train an RL-based scheduling policy with Proximal Policy Optimization (PPO) for each agent using the collected dataset from Phase 1. The agents are trained on data using behavior cloning, which leverages positive rewards to align with the safe actions of the conventional scheduler. For Phase 3, we distill the trained RL policy into a decision tree, ensuring low overhead and feasibility for OS-level deployment. The decision tree replicates the RL policy's decisions using simple if-then-else rules, allowing real-time execution with minimal computational cost [3]. Finally, we integrate this DTS into the AV platform, achieving both runtime safety and performance improvements over traditional schedulers.

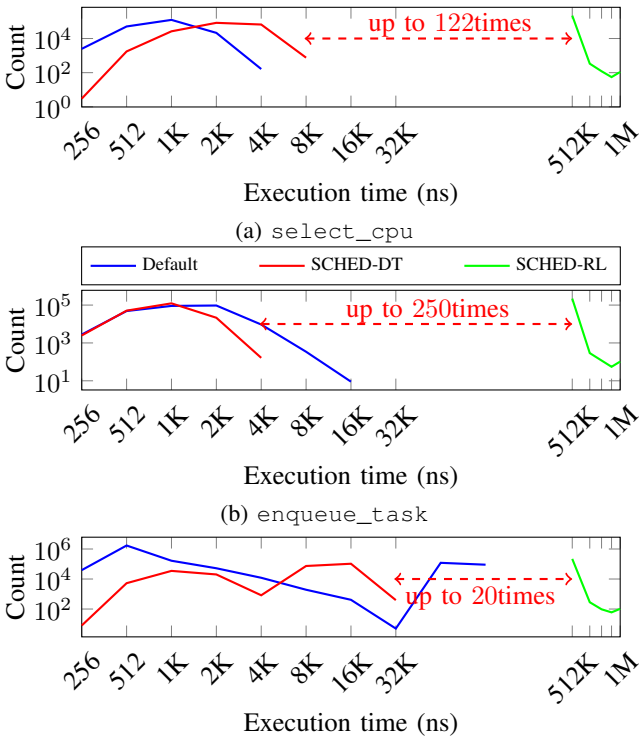
TABLE I: State-Action Set of Scheduling Functions

Target Function	State and Action
`select_cpu`	**State:** idleness of each CPU, current CPU id. **Action:** selection of CPU id.
`enqueue_task`	**State:** current task type, workload status, delay level. **Action:** direct dispatch or enqueue.
`dispatch_task`	**State:** queue length. **Action:** selection of queue.

III. Experimental Evaluation

We use Linux kernel version 6.12.0, ROS2 Iron, and Linux extensible scheduler class with extended berkeley packet filter (eBPF) to implement the framework[1]. To evaluate SCHED's ability to make OS scheduling decisions in realtime, we measure execution time per invocation of (1) the Linux Completely Fair Scheduler (Default), (2) SCHED before distillation (SCHED-RL), and (3) SCHED as a decision tree (SCHED-DT). We evaluate SCHED for a simulated AV workload using Chauffeur [1] to demonstrate its feasibility for real-world deployment. Figure 2 shows histograms of per-invocation execution latency for the three scheduler decision points. The Default scheduler exhibits a narrow distribution centred below 40 µs, reflecting its lightweight rule-based logic. The SCHED-RL incurs substantial inference overhead, producing tail latencies up to 250× longer than SCHED-DT and therefore proving unsuitable for in-kernel deployment. In contrast, SCHED-DT retains the low-latency profile of Default while leveraging the policy decisions learned by RL, demonstrating its viability for real-time OS integration in autonomous-vehicle systems.

[1]Source code available at: https://github.com/changhoon-sung/SCHED



(a) `select_cpu`

(b) `enqueue_task`

(c) `dispatch_task`

Fig. 2: Execution time distribution (histogram) of (a) `select_task`, (b) `enqueue_cpu`, and (c) `dispatch_task`, measured across all function invocations while running AV pipelined workloads. Y-axis reports the number of invocations in each latency bin.

IV. Conclusion

In this paper, we presented a RL-based framework for CPU scheduling in AVs, addressing the critical need for low-latency, safe decision-making within complex, mixed-criticality workloads. Our approach integrates offline data collection from a PPO-agent-based scheduler, followed by decision-tree distillation to ensure reduced runtime overhead at the OS level. Experimental evaluation on an AV pipeline demonstrated that our distilled RL-based CPU scheduler reduces scheduling decision execution time by up to 250 times compared with the SCHED-RL policy, achieving comparable execution time to existing solutions. This work introduces the first RL-based CPU scheduling framework, bridging learning-based optimization with OS-level feasibility.

References

[1] B. Maity, S. Yi, D. Seo, L. Cheng, S.-S. Lim, J.-C. Kim, B. Donyanavard, and N. Dutt, "Chauffeur: Benchmark suite for design and end-to-end analysis of self-driving vehicles on embedded systems," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 20, no. 5s, pp. 1–22, 2021.

[2] V. Venkataswamy, J. Grigsby, A. Grimshaw, and Y. Qi, "Launchpad: Learning to schedule using offline and online rl methods," in *Workshop on Job Scheduling Strategies for Parallel Processing*, pp. 60–83, Springer, 2024.

[3] A. Verma, V. Murali, R. Singh, P. Kohli, and S. Chaudhuri, "Programmatically interpretable reinforcement learning," in *International conference on machine learning*, pp. 5045–5054, PMLR, 2018.