# Learning-Based Task Scheduling Using Reinforcement Learning

Islah Haoues
*Department of Computer Engineering*
*Bahcesehir University*
Istanbul, Turkey
islah.haoues@bahceshir.edu.tr

Baker Huseyin
*Department of Computer Engineering*
*Bahcesehir University*
Istanbul, Turkey
baker.huseyin@bahceshir.edu.tr

***Abstract*—Task scheduling is a fundamental problem in computing systems and has traditionally been addressed using heuristic algorithms such as First-Come First-Served (FCFS), Shortest Job First (SJF), Round Robin (RR), and Earliest Deadline First (EDF) [1]–[3]. While these methods are effective under specific assumptions, they can be brittle under dynamic and heterogeneous workloads. This paper investigates reinforcement learning (RL) as a learning-based alternative for task scheduling [4]. We design a discrete-time simulation environment and implement a Deep Q-Network (DQN) scheduler [6] that learns scheduling decisions through interaction with the system. The approach is evaluated through a comparative study against classical scheduling algorithms on multiple synthetic workload types, including uniform, bursty, heavy-tailed, and deadline-constrained scenarios. Performance is assessed using average waiting time, deadline miss rate, and throughput. Experimental results show that SJF remains highly effective when task processing times are available, while the DQN scheduler exhibits stable, competitive performance comparable to strong baselines, but does not surpass SJF in the studied settings. These findings highlight both the potential and limitations of reinforcement learning for task scheduling in controlled environments.**

***Index Terms*—Reinforcement learning, Deep Q-Network, task scheduling, CPU scheduling, comparative evaluation, simulation-based study**

## I. Introduction

Task scheduling is a fundamental problem in computing systems, directly affecting performance, responsiveness, and resource utilization. In operating systems, cloud platforms, and real-time systems, a scheduler determines the order in which tasks are executed, influencing metrics such as waiting time, throughput, and deadline satisfaction [1], [2].

Traditional scheduling policies such as FCFS, SJF, RR, and EDF remain widely used due to their simplicity and strong properties under specific assumptions [1]–[3]. For example, SJF is known to minimize average waiting time when processing times are known, while EDF has optimality guarantees in certain real-time models [3]. However, these algorithms rely on fixed decision rules and are typically optimized for narrow objectives or workload characteristics. As system conditions change, their performance can degrade or become suboptimal.

Reinforcement learning (RL) offers a data-driven alternative for sequential decision-making in dynamic environments [4]. An RL agent learns behavior by interacting with an environment and optimizing long-term reward, rather than relying on manually designed heuristics. With deep function approximation, deep RL can handle larger state spaces, as demonstrated by Deep Q-Networks (DQN) [6]. These characteristics make RL attractive for adaptive scheduling, where decisions may depend on complex interactions among task attributes and queue dynamics.

In this work, we study RL for task scheduling using a controlled simulation-based comparative evaluation. We implement a DQN scheduler and compare it with FCFS, RR, SJF, and EDF under four synthetic workload types. Our aim is not to claim universal superiority of RL, but to quantify how close a learning-based policy can come to strong heuristics under identical conditions and to analyze where each method succeeds or fails.

### A. Contributions

The main contributions of this paper are:

- A discrete-time single-processor scheduling simulator that supports multiple workload families and reproducible evaluation.
- An RL-based scheduler using DQN, trained end-to-end from interaction with the simulator [6].
- A controlled comparative study across multiple workload types using average waiting time, deadline miss rate, and throughput as evaluation criteria.
- An analysis of observed behavior showing when classical heuristics dominate and when the RL approach provides competitive performance.

The remainder of this paper is organized as follows. Section II reviews related work. Section III describes the simulator, baselines, and RL formulation. Section IV presents the experimental setup and configuration. Section V reports the results. Section VI discusses implications and limitations. Section VII concludes and outlines future directions.

## II. Related Work

Classical scheduling algorithms such as FCFS, SJF, RR, and EDF form the foundation of practical CPU scheduling systems [1], [2]. SJF minimizes average waiting time when job sizes are known, though it can cause starvation for long jobs. RR improves fairness through time-slicing. EDF is widely

studied in real-time scheduling and has optimality properties in specific feasibility models [3].

Reinforcement learning has been applied to resource management and scheduling across systems domains. Mao *et al.* demonstrated that deep RL can learn effective policies for systems-level resource management, outperforming hand-tuned heuristics in certain cluster settings [7]. Earlier work explored reinforcement learning for autonomic resource allocation and control [8]. More broadly, RL foundations such as Q-learning [5] and modern deep RL methods such as DQN [6] provide a standard framework for learning policies from interaction [4].

Prior studies suggest that RL-based scheduling can be competitive, particularly under non-stationary dynamics or when handcrafted heuristics are difficult to design. At the same time, strong heuristics often remain difficult to beat under idealized assumptions such as perfect knowledge of processing times, which can favor SJF-like behavior. Motivated by this, our work emphasizes controlled comparisons that isolate the effect of the scheduling decision rule under multiple workload types.

## III. Methodology

This section describes the simulation environment, baseline algorithms, and the reinforcement learning formulation.

### A. Simulation Environment

We model a single-processor system in discrete time. Tasks are characterized by arrival time, processing time (job size), and deadline. At each time step, tasks with arrival time less than or equal to the current time join the ready queue. The scheduler selects one task to execute for one unit of time, decreasing its remaining processing time by one. Completed tasks are removed and their completion time is recorded. Waiting time accumulates for tasks in the ready queue that are not selected.

The simulation ends when all tasks complete or when a maximum time horizon is reached. This design provides fine-grained control over workload characteristics and supports consistent measurement of waiting time, deadline misses, and throughput.

### B. Baseline Scheduling Algorithms

We compare the learning-based scheduler against four classical algorithms:

- **FCFS:** schedules tasks in arrival order [1], [2].
- **SJF:** prioritizes tasks with the smallest remaining processing time [1], [2].
- **RR:** cycles through tasks using a fixed quantum of one time unit for fairness [1], [2].
- **EDF:** prioritizes tasks with the earliest deadlines [3].

### C. Problem Formulation as an MDP

We formulate scheduling as a Markov Decision Process (MDP) [4]. At time $t$, the agent observes a state $s_t$ derived from the ready queue and chooses an action $a_t$ corresponding to selecting a task to execute. After executing the action, the environment transitions to a new state $s_{t+1}$ and returns a reward $r_t$.

The objective is to learn a policy $\pi(a \mid s)$ that maximizes expected discounted return:

$$J(\pi) = \mathbb{E}\Big[\sum_{t=0}^{\infty} \gamma^t r_t\Big], \tag{1}$$

where $\gamma \in (0, 1]$ is the discount factor [4].

### D. DQN Scheduler

We employ a Deep Q-Network (DQN) approach [6]. The action-value function $Q(s, a)$ is approximated with a neural network parameterized by $\theta$. The DQN is trained using experience replay and a target network, following standard practice [6].

The Bellman target for Q-learning is:

$$y_t = r_t + \gamma \max_{a'} Q_{\theta^-}(s_{t+1}, a'), \tag{2}$$

and the parameters $\theta$ are updated by minimizing the squared temporal-difference error:

$$\mathcal{L}(\theta) = \mathbb{E}\big[(y_t - Q_\theta(s_t, a_t))^2\big], \tag{3}$$

where $\theta^-$ denotes target network parameters [5], [6].

### E. State and Action Representation

The state representation includes task-level features such as remaining processing time, accumulated waiting time, and deadline proximity. To keep the input dimension fixed, only the top $K$ tasks from the ready queue are encoded, ordered by deadline and remaining processing time. Global features such as queue length and average waiting time are included as additional inputs.

The action space corresponds to selecting one of the $K$ visible tasks. Invalid actions are masked when fewer than $K$ tasks are present.

### F. Reward Design

The reward is designed to reduce waiting and penalize deadline violations. At each step, the agent receives a negative component proportional to queue congestion and waiting, a positive reward upon task completion, and an additional penalty if a completed task misses its deadline. This reward structure encourages efficient execution while discouraging excessive waiting and missed deadlines.

### G. Implementation and Training Details

The DQN is implemented as a multilayer perceptron mapping the fixed-size state vector to $K$ Q-values. We follow standard DQN stabilization techniques, including experience replay and a periodically updated target network [6]. During training, actions are selected via an $\epsilon$-greedy policy to balance exploration and exploitation [4]. During evaluation, we use a greedy policy without exploration to measure deterministic performance.

To support reproducibility, all experiments are executed with fixed random seeds for workload generation and model initialization. Results are reported as mean and standard deviation across multiple seeds.

TABLE I
WORKLOAD FAMILIES USED FOR EVALUATION.

| Workload | Key characteristic |
|---|---|
| Uniform | Steady arrivals, moderate sizes |
| Bursty | Congestion spikes due to bursts |
| Heavy-tailed | Mix of short and long jobs |
| Deadline-constrained | Tight deadlines |

## IV. EXPERIMENTAL SETUP

### A. Workload Generation

We generate synthetic workloads to enable controlled comparisons. Four workload types are used:

- **Uniform:** steady arrivals with moderate job sizes and relatively loose deadlines.
- **Bursty:** clustered arrivals that create congestion followed by idle periods.
- **Heavy-tailed:** job sizes follow a heavy-tailed distribution, producing a mix of very short and very long tasks.
- **Deadline-constrained:** tighter deadlines relative to job sizes, emphasizing deadline compliance.

Each workload type is generated using multiple random seeds. Reported values are means and standard deviations across seeds.

### B. Evaluation Metrics

We report the following metrics:

- **Average Waiting Time:** mean time tasks spend waiting in the ready queue.
- **Deadline Miss Rate:** fraction of tasks completing after their deadline.
- **Throughput:** completed tasks per unit simulation time.

### C. Experimental Protocol

For each workload and seed, we evaluate FCFS, RR, SJF, EDF, and DQN using the same task set. The DQN scheduler is trained on separate training workloads and evaluated using a greedy policy without exploration. Baselines use deterministic policies. This ensures differences reflect scheduling decisions rather than dataset differences.

### D. Workload Summary Table

## V. EXPERIMENTAL CONFIGURATION

### A. Learning Hyperparameters

Table II summarizes representative DQN hyperparameters used for training. Hyperparameters were selected empirically to ensure stable learning within the computational limits of the project. If a different configuration is used, the table can be updated accordingly.

TABLE II
REPRESENTATIVE DQN HYPERPARAMETERS.

| Parameter | Value |
|---|---|
| Discount factor $\gamma$ | 0.99 |
| Replay buffer capacity | 50,000 transitions |
| Batch size | 64 |
| Learning rate | $10^{-3}$ |
| Target network update | every 500 steps |
| Exploration policy | $\epsilon$-greedy, decays to 0.05 |
| Optimizer | Adam |

### B. State Size and Action Space

The scheduler observes the top $K$ tasks from the ready queue to construct a fixed-dimensional state vector. Increasing $K$ increases visibility and potentially improves decisions, but it also expands the action space and can slow learning. The action set consists of selecting one of the visible tasks; invalid actions are masked when fewer than $K$ tasks exist.

### C. Runtime and Complexity Considerations

Classical baselines are computationally inexpensive. FCFS and RR are $O(1)$ per decision, while SJF and EDF require selecting the minimum processing time or earliest deadline among the ready queue, which can be implemented in $O(\log n)$ using priority queues. In contrast, DQN action selection requires a forward pass through the neural network, and training additionally performs backpropagation updates with replay batches. This increases computational cost but enables learning from data and potentially adapting to more complex decision boundaries.

### D. Reproducibility Notes

To facilitate reproducibility, the code fixes random seeds for workload generation and model initialization, logs evaluation metrics for each seed, and produces the plots and tables included in this report. A clean repository structure with separate notebooks for generation, training, and evaluation reduces the risk of configuration drift across runs.

## VI. RESULTS

This section reports quantitative results for all schedulers across the four workload types. We first discuss each metric, then provide an overall comparison. Figures are placed within their relevant subsections to reduce float reordering.

### A. Average Waiting Time

Across all workloads, SJF achieves the lowest average waiting time. For example, in the uniform workload, SJF achieves $1899.71 \pm 111.61$ compared to $3121.03 \pm 135.79$ for DQN and $3121.05 \pm 137.01$ for FCFS. In the heavy-tailed workload, SJF achieves $787.15 \pm 39.26$, substantially lower than DQN at $1926.88 \pm 128.46$ and FCFS at $1933.41 \pm 126.74$. These results are consistent with the known advantage of SJF when job sizes are available [1], [2].

The DQN scheduler closely tracks FCFS and EDF in waiting time across workloads. In uniform and bursty settings,

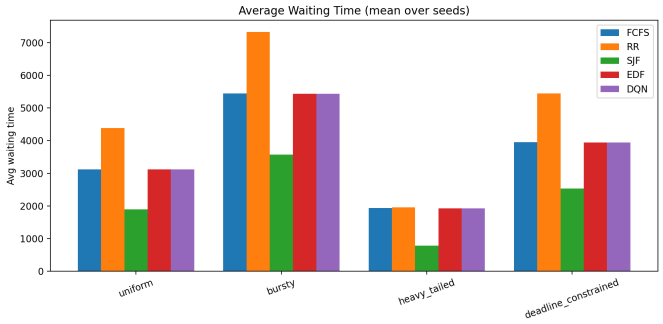


Fig. 1. Average waiting time across scheduling algorithms.

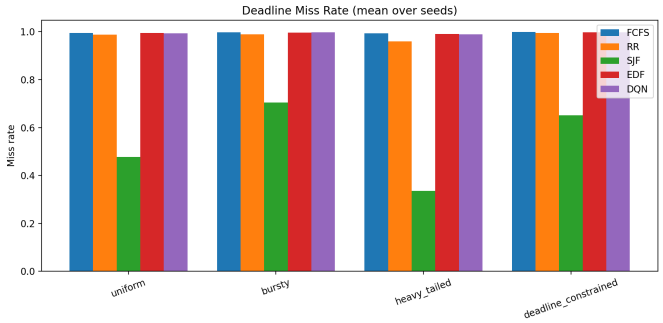


Fig. 2. Deadline miss rate across scheduling algorithms.

DQN is nearly indistinguishable from FCFS and EDF, indicating that the learned policy does not discover an ordering that meaningfully improves mean waiting time beyond those baselines under the current reward and state representation.

### *B. Deadline Miss Rate*

Deadline miss rates are generally high for FCFS, EDF, and DQN across workloads, particularly under the deadline-constrained setting. In the uniform workload, SJF reduces the miss rate to $0.478 \pm 0.019$, while FCFS, EDF, and DQN are near one at approximately $0.995$ to $0.993$. In the heavy-tailed workload, SJF again yields the lowest miss rate at $0.335 \pm 0.005$ compared to DQN at $0.990 \pm 0.003$.

In the deadline-constrained workload, all methods exhibit very high miss rates, with SJF achieving $0.651 \pm 0.002$ while FCFS, EDF, and DQN remain near $0.999$ to $0.998$. This suggests that the workload is close to infeasible under the simulated constraints, so EDF cannot substantially reduce misses. Under such conditions, heuristics that reduce queueing delay, such as SJF, can still provide partial improvements.

### *C. Throughput*

Throughput is identical across schedulers for each workload family, as shown in Table III. For example, throughput is $0.184 \pm 0.002$ for the uniform workload and $0.264 \pm 0.004$ for heavy-tailed across all methods. This indicates that differences among schedulers arise primarily from ordering effects that influence waiting time and deadline satisfaction, rather than from changes in CPU utilization or system completion capacity within the simulation assumptions.

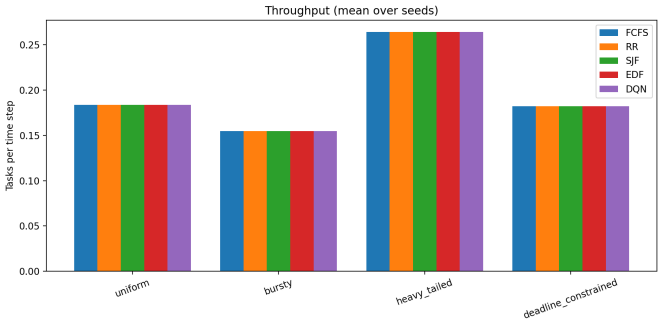


Fig. 3. System throughput across scheduling algorithms.

### *D. Overall Comparison*

Overall, SJF dominates average waiting time and significantly improves deadline miss rate relative to other methods in all workloads. The DQN scheduler is stable and reproducible, but behaves similarly to FCFS and EDF in the evaluated environments. These findings indicate that when processing times are directly accessible, strong classical heuristics remain difficult to beat, and learning-based methods may require richer state, improved reward shaping, or problem settings where handcrafted heuristics are less effective.

TABLE III
SUMMARY OF SCHEDULING PERFORMANCE ACROSS WORKLOADS (MEAN ± STD OVER SEEDS).

| Workload | Scheduler | AvgWait | MissRate | Throughput |
|---|---|---|---|---|
| uniform | FCFS | 3121.05 ± 137.01 | 0.995 ± 0.003 | 0.184 ± 0.002 |
| uniform | RR | 4383.05 ± 232.81 | 0.988 ± 0.002 | 0.184 ± 0.002 |
| uniform | SJF | 1899.71 ± 111.61 | 0.478 ± 0.019 | 0.184 ± 0.002 |
| uniform | EDF | 3117.13 ± 136.94 | 0.995 ± 0.003 | 0.184 ± 0.002 |
| uniform | DQN | 3121.03 ± 135.79 | 0.993 ± 0.002 | 0.184 ± 0.002 |
| bursty | FCFS | 5447.87 ± 83.35 | 0.998 ± 0.000 | 0.155 ± 0.001 |
| bursty | RR | 7323.81 ± 167.10 | 0.990 ± 0.007 | 0.155 ± 0.001 |
| bursty | SJF | 3569.81 ± 87.86 | 0.704 ± 0.014 | 0.155 ± 0.001 |
| bursty | EDF | 5434.59 ± 83.04 | 0.997 ± 0.000 | 0.155 ± 0.001 |
| bursty | DQN | 5434.91 ± 81.24 | 0.997 ± 0.001 | 0.155 ± 0.001 |
| heavy$_{tailed}$ | FCFS | 1933.41 ± 126.74 | 0.993 ± 0.002 | 0.264 ± 0.004 |
| heavy$_{tailed}$ | RR | 1958.17 ± 95.52 | 0.960 ± 0.008 | 0.264 ± 0.004 |
| heavy$_{tailed}$ | SJF | 787.15 ± 39.26 | 0.335 ± 0.005 | 0.264 ± 0.004 |
| heavy$_{tailed}$ | EDF | 1926.73 ± 126.20 | 0.990 ± 0.001 | 0.264 ± 0.004 |
| heavy$_{tailed}$ | DQN | 1926.88 ± 128.46 | 0.990 ± 0.003 | 0.264 ± 0.004 |
| deadline$_{constrained}$ | FCFS | 3949.56 ± 74.84 | 0.999 ± 0.000 | 0.182 ± 0.001 |
| deadline$_{constrained}$ | RR | 5446.00 ± 90.81 | 0.995 ± 0.003 | 0.182 ± 0.001 |
| deadline$_{constrained}$ | SJF | 2531.61 ± 47.92 | 0.651 ± 0.002 | 0.182 ± 0.001 |
| deadline$_{constrained}$ | EDF | 3943.28 ± 74.67 | 0.998 ± 0.000 | 0.182 ± 0.001 |
| deadline$_{constrained}$ | DQN | 3946.12 ± 75.55 | 0.998 ± 0.003 | 0.182 ± 0.001 |

## VII. DISCUSSION

The results confirm that strong classical heuristics remain highly competitive under idealized assumptions. SJF consistently yields the best waiting-time performance and also achieves the lowest miss rates in every workload family, including the deadline-constrained setting. This behavior is consistent with classical scheduling theory, where SJF is optimal for minimizing average waiting time when job sizes are known [1], [2]. In our experiments, SJF also reduces congestion, which indirectly improves deadline outcomes even when deadlines are tight.

A key observation is that EDF does not reduce deadline miss rate in this simulator. EDF is known to have strong properties in certain hard real-time feasibility models [3]. However, when the workload is heavily overloaded or deadlines are too tight, the system can become largely infeasible, and EDF cannot prevent misses. This aligns with our deadline-constrained results, where miss rates remain near one for FCFS, EDF, and DQN, while SJF still provides partial improvement by reducing queueing delay.

The DQN scheduler produces stable performance but does not surpass the strongest heuristic baselines. In the uniform and bursty workloads, DQN closely matches FCFS and EDF in both waiting time and miss rate. This suggests that, under the present state representation and reward shaping, the agent converges to policies that are effectively similar to simple non-preemptive ordering rules. When a heuristic like SJF is near-optimal given available information, the RL agent must discover a similarly informative decision boundary, which may require richer features, improved reward design, or additional training diversity.

These results also highlight that the value of RL in scheduling may be greatest when classical assumptions do not hold. In real systems, processing times and future arrivals may be uncertain, tasks may have complex resource requirements, and objectives may change over time. RL has been shown to be effective for systems-level decision-making in settings where designing heuristics is difficult [7], [8]. Extending the simulator to include partial observability, stochastic job sizes, multi-resource constraints, or non-stationary workloads is therefore a promising direction for future work.

## VIII. THREATS TO VALIDITY AND LIMITATIONS

Several factors limit the generality of the conclusions. First, workloads are synthetic and may not reflect real traces from production systems. Second, the environment models a single-processor system with a simplified task model, and the throughput metric is largely insensitive to scheduling choice under these assumptions. Third, SJF benefits from explicit knowledge of processing time, which is often unavailable or noisy in real systems, and this makes SJF difficult to beat under the chosen simulator settings.

On the learning side, the DQN policy depends strongly on reward design and state representation. The fixed-size encoding of the ready queue restricts visibility into long queues, which can limit the policy's ability to learn strategies that depend on long-range queue structure. Hyperparameter choices and finite training time can also prevent convergence to stronger policies. These limitations motivate future work using partial observability, stochastic job sizes, multi-resource constraints, and alternative RL formulations.

## IX. CONCLUSION

This paper presented a comparative study of classical and reinforcement learning-based task scheduling algorithms using a discrete-time simulation environment. A DQN scheduler [6] was implemented and evaluated alongside FCFS, RR, SJF, and EDF across uniform, bursty, heavy-tailed, and deadline-constrained workloads.

Experimental results show that classical heuristics remain highly effective under controlled assumptions. In particular, SJF consistently achieves the lowest average waiting time and substantially reduces deadline miss rates relative to other methods. The DQN scheduler exhibits stable performance but does not outperform SJF and behaves similarly to FCFS and EDF in the evaluated environments. These findings suggest that, when job sizes are explicitly available, strong heuristics remain difficult to surpass, and RL may be more suitable in settings with uncertainty, richer constraints, or changing objectives.

Future work can extend this study to multi-processor environments, partial observability, stochastic processing times, and multi-resource scheduling. Additionally, alternative RL formulations, reward shaping, and richer state representations may improve learning-based performance in dynamic scheduling scenarios.

## REFERENCES


[1] A. S. Tanenbaum and H. Bos, *Modern Operating Systems*, 4th ed. Upper Saddle River, NJ, USA: Pearson, 2015.

[2] W. Stallings, *Operating Systems: Internals and Design Principles*, 9th ed. Boston, MA, USA: Pearson, 2018.

[3] 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, Jan. 1973.

[4] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. Cambridge, MA, USA: MIT Press, 2018.

[5] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, no. 3, pp. 279–292, 1992.

[6] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015.

[7] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, "Resource management with deep reinforcement learning," in *Proc. ACM SIGCOMM*, 2016, pp. 50–63.

[8] G. Tesauro, N. K. Jain, M. N. Rosenberg, and S. M. Talukdar, "On the use of hybrid reinforcement learning for autonomic resource allocation," in *Proc. IEEE ICAC*, 2006, pp. 50–57.