Multi-Agent Reinforcement Learning for Non-Stationary Edge Task Scheduling

David Li
Katz School of Science and Health
Yeshiva University
david.li@yu.edu

Angela Li
Applied Mathematics and Physics
Stony Brook University
angela.li.4@stonybrook.edu

***Abstract*—Task scheduling in edge-enabled Internet-of-Things systems becomes fundamentally harder when device capability is non-stationary: repeated execution can degrade near-term processing efficiency, while idleness enables recovery. Existing learning-based schedulers typically optimize latency or utilization under stationary or exogenous service models, which limits their ability to learn when temporary deferral is beneficial. We formulate this problem as a multi-agent stochastic game in which each device carries a time-varying processing-efficiency state governed by controlled mean-reverting dynamics. Building on this model, we develop a degradation-recovery reinforcement-learning scheduler whose main instantiation is a tabular Q-learning policy over backlog, efficiency, and shared-context variables, and we discuss a policy-gradient extension for continuous-state settings. Experiments on heterogeneous edge IoT systems show that the learned policy acquires recovery-aware behavior, executing aggressively in high-efficiency states and deferring selectively when devices are degraded. Across low, medium, and high load regimes, the proposed method reduces deadline violations by 8–31% and improves throughput by 12–19% relative to fatigue-unaware reinforcement learning, with gains increasing as degradation strength grows. The method also maintains stable backlog as the number of devices increases and achieves sub-millisecond inference with constant per-device model size. These results show that explicitly modeling endogenous degradation can substantially improve reinforcement-learning-based scheduling in non-stationary resource-constrained environments.**

***Index Terms*—reinforcement learning, Q-learning, non-stationary environments, multi-agent scheduling**

I. Introduction

Edge-enabled Internet-of-Things (IoT) systems increasingly rely on distributed resource-constrained devices to process latency-sensitive workloads under tight timing, energy, and reliability constraints [1], [2]. In such systems, scheduling is not merely a question of matching tasks to available devices. In many realistic deployments, a device's effective service capability is itself non-stationary: sustained execution may reduce near-term processing efficiency because of thermal stress, energy drain, or workload accumulation, while idle periods allow partial recovery [8], [9]. As a result, a scheduler that always serves whenever backlog is nonzero can be myopic, because immediate execution may degrade future service quality.

Reinforcement learning (RL) has become an attractive approach for adaptive scheduling in edge and fog environments, especially when the system dynamics are stochastic, high-dimensional, or difficult to model analytically. Existing learning-based schedulers have been used to optimize latency, offloading, response time, and system utilization in edge computing and related distributed settings [3]–[6], and broader reviews likewise highlight the growing role of deep RL in scheduling and resource management problems [7]. However, most existing formulations treat device capability as fixed or exogenous. In contrast, the present paper focuses on *endogenous non-stationarity*: a device's own execution decisions change its future processing condition. This introduces an intertemporal scheduling trade-off that cannot be captured by backlog-only or fatigue-unaware policies.

The challenge becomes even sharper in multi-device edge systems. Devices interact through shared communication or computation resources, so one agent's decision affects not only its own queue evolution but also the operating conditions experienced by others. The resulting problem is sequential, stochastic, partially observed, and multi-agent. To model this structure, we represent each device with a time-varying processing-efficiency state that evolves according to controlled mean-reverting dynamics related to the Ornstein–Uhlenbeck process [15]. This state acts as a compact representation of degradation and recovery, allowing the scheduler to distinguish between moments when immediate execution is valuable and moments when temporary deferral is preferable.

Based on this formulation, we develop a multi-agent reinforcement-learning framework for non-stationary edge task scheduling. Our main implementation is a tabular Q-learning scheduler [17] that selects between execution and deferral using backlog, processing efficiency, and shared-context information. We also discuss a policy-gradient extension for continuous-state or partially observed settings, motivated by the broader RL literature on continuous control [16]. The central idea is that scheduling decisions should depend not only on queue pressure, but also on short-term device condition and its expected recovery.

The main contributions of this paper are as follows:

- We propose a stochastic scheduling model for edge IoT systems in which each device has a time-varying processing-efficiency state governed by degradation and recovery, and formulate the resulting problem as a multi-agent stochastic game.

- We develop a reinforcement-learning scheduler based on tabular Q-learning for adaptive execute/defer decisions, and discuss a policy-gradient extension for continuous-state and partially observed environments.
- We provide a comprehensive empirical evaluation across multiple load regimes, heterogeneous device classes, ablations, degradation-strength sweeps, and scaling experiments, showing consistent improvements over greedy and fatigue-unaware RL baselines.

The rest of the paper is organized as follows. Section II reviews related work. Section III presents the system model and problem formulation. Section IV develops the reinforcement-learning method. Section V discusses the multi-agent setting under heterogeneity and partial observability. Section VI reports the experimental results. Section VII concludes the paper.

II. Related Work

A. RL for Edge and IoT Scheduling

RL-based scheduling has become increasingly important in edge and fog computing, where dynamic arrivals, variable resource availability, and delayed feedback make hand-crafted policies difficult to tune. Early work studied learning-based offloading and execution control for virtual edge systems [2], while later methods applied deep RL directly to IoT task scheduling [3], digital-twin-assisted scheduling [6], and large-scale edge/fog environments emphasizing load and response-time optimization [4]. More recent work has introduced richer neural architectures, including transformer-enhanced distributed DRL schedulers for IoT applications across edge and cloud platforms [5]. At a broader level, recent reviews also document the expanding role of deep RL in job scheduling and resource management [7]. These studies demonstrate the promise of learning-based scheduling, but they typically model service capability as stationary or externally varying rather than as a state variable shaped by the agent's own recent actions.

B. Time-Dependent Processing and Fatigue Effects

The scheduling literature has long recognized that processing times and service rates may depend on temporal effects such as aging, deterioration, or accumulated workload. Surveys on time-dependent processing-time models [8] and later work on aging effects in machine scheduling [9] provide important conceptual foundations for treating service capability as non-stationary. Related ideas also appear in recent dynamic scheduling work that incorporates worker fatigue and skill effects into a learning-based manufacturing framework [10]. Our setting is related in spirit, but differs in both domain and modeling objective. We study distributed edge IoT devices rather than human workers or single-machine production systems, and our goal is not simply to encode time-dependent service, but to make degradation and recovery explicit state variables inside a reinforcement-learning scheduler.

C. Multi-Agent RL Under Coupled Decisions

Because edge devices interact through shared load, contention, and common system constraints, the scheduling problem is naturally multi-agent. A large body of multi-agent RL work provides algorithmic foundations for such settings, including actor-critic methods for mixed cooperative environments [11], counterfactual policy gradients for multi-agent credit assignment [12], and value-factorization methods such as QMIX [13]. Selective overviews further summarize the theoretical and algorithmic landscape of multi-agent RL under partial observability and non-stationarity [14]. These works motivate the multi-agent perspective adopted here. However, our focus is different: rather than introducing a new MARL algorithm, we show that explicitly modeling endogenous degradation and recovery changes the scheduling state itself, and that even a lightweight Q-learning method can exploit this structure effectively.

Overall, our work lies at the intersection of RL-based edge scheduling, time-varying service models, and multi-agent learning. Its distinguishing feature is the explicit treatment of degradation and recovery as a controlled latent resource state that drives non-stationary execution quality. This perspective bridges classical time-dependent scheduling ideas with reinforcement learning for intelligent edge systems.

III. System Model and Problem Formulation

We consider a stochastic multi-agent scheduling environment for an edge-enabled IoT system with N resource-constrained devices. Time is slotted and indexed by $t = 0, 1, 2, \ldots$. Each device is modeled as an adaptive agent that repeatedly decides whether to process a task immediately or defer execution in order to preserve future service capability.

A. System Dynamics

Let $q_{i,t} \in \mathbb{Z}_{\geq 0}$ denote the local backlog of device i at time t, and let $P_{i,t} \in [0, P_{\max}]$ denote its effective processing efficiency. The variable $P_{i,t}$ captures the instantaneous ability of device i to execute tasks under current workload, thermal condition, and energy state. Higher values of $P_{i,t}$ correspond to faster or more reliable task execution.

At each slot, device i chooses an action

$$a_{i,t} \in \mathcal{A}_i = \{\texttt{execute}, \texttt{defer}\}. \tag{1}$$

We define the binary execution indicator

$$u_{i,t} = \mathbb{I}\{a_{i,t} = \texttt{execute}\}. \tag{2}$$

If device i executes at time t, it attempts to process one pending task. Let $C_{i,t} \in \{0, 1\}$ denote the task-completion indicator. We model successful completion as a Bernoulli random variable with success probability increasing in the current processing efficiency:

$$\mathbb{P}(C_{i,t} = 1 \mid s_t, a_{i,t}) = u_{i,t}\, \psi_i(P_{i,t}, \xi_t), \tag{3}$$

where $\psi_i(\cdot, \cdot) \in [0, 1]$ is a monotone function and ξ_t denotes optional shared context such as contention level, channel condition, or common system load. In the simplest implementation used in our simulations, ψ_i depends only on $P_{i,t}$.

The queue dynamics of device i are

$$q_{i,t+1} = \left[q_{i,t} - C_{i,t}\right]^+ + A_{i,t}, \tag{4}$$

where $A_{i,t} \in \mathbb{Z}_{\geq 0}$ denotes the random task arrivals at slot t. When the focus is on a finite shared task pool rather than streaming arrivals, (4) reduces to a finite-horizon depletion model by setting $A_{i,t} \equiv 0$ after initialization.

B. Performance Degradation and Recovery

A central feature of the model is that each device's service capability is time-varying and affected by its own execution history. Repeated execution can reduce near-term efficiency because of workload accumulation, thermal throttling, or energy depletion, while idleness enables partial recovery. We model this degradation-recovery behavior through the controlled mean-reverting dynamics

$$P_{i,t+1} = \Pi_{[0,P_{\max}]}\left(P_{i,t} + \kappa_i(\mu_i - P_{i,t})\Delta t - \beta_i u_{i,t} + \sigma_i \varepsilon_{i,t}\right), \tag{5}$$

where $\mu_i > 0$ is the nominal processing level, $\kappa_i > 0$ is the recovery rate, $\beta_i \geq 0$ is the execution-induced degradation coefficient, $\sigma_i \geq 0$ controls stochastic variability, $\varepsilon_{i,t} \sim \mathcal{N}(0,1)$, and $\Pi_{[0,P_{\max}]}(\cdot)$ denotes projection onto $[0, P_{\max}]$.

Equation (5) is a discrete-time, control-dependent mean-reverting model. It is closely related to Ornstein–Uhlenbeck-type dynamics, while making the effect of execution decisions explicit: the term $-\beta_i u_{i,t}$ captures immediate degradation caused by processing load, and the drift term $\kappa_i(\mu_i - P_{i,t})$ captures recovery toward the nominal operating point during low activity.

C. Shared Resource Coupling

The devices are coupled through shared system conditions. This coupling may arise through common load, contention for transmission or execution opportunities, or a finite pool of pending tasks. Let

$$\xi_t \in \Xi \tag{6}$$

denote the shared context variable summarizing such system-level effects. For example, ξ_t may represent the number of devices attempting to execute simultaneously, an aggregate congestion signal, or the current size of a common task pool.

When multiple devices execute concurrently, shared resource contention can reduce completion probability or limit the number of tasks that can be served. This interaction is incorporated through the transition kernel and the completion model in (3), rather than through a fixed deterministic service rule. As a result, one device's execution decision can affect not only its own future efficiency but also the effective scheduling environment faced by others.

D. State, Observation, and Reward

The global system state is

$$s_t = \left(q_{1,t}, \ldots, q_{N,t},\, P_{1,t}, \ldots, P_{N,t},\, \xi_t\right) \in \mathcal{S}. \tag{7}$$

In a centralized setting, the full state s_t is available to the learner. In a decentralized IoT setting, device i observes only a local signal

$$o_{i,t} = \phi_i(s_t), \tag{8}$$

which may include its local backlog $q_{i,t}$, local efficiency $P_{i,t}$, and limited side information about system load or contention.

To capture the trade-off between immediate task service and long-term system efficiency, we use a per-device reward of the form

$$r_{i,t} = \omega_c C_{i,t} - \omega_q q_{i,t} - \omega_e e_i(u_{i,t}, P_{i,t}), \tag{9}$$

where $\omega_c, \omega_q, \omega_e \geq 0$ are weights and $e_i(u_{i,t}, P_{i,t})$ is an optional energy or execution-cost term. The first term rewards task completion, the second penalizes queueing delay through backlog, and the third discourages excessive execution when such behavior is energetically inefficient or systemically costly. Other delay- or deadline-aware reward structures can be incorporated without changing the learning framework.

Each device seeks to maximize its expected discounted return

$$J_i(\pi_i, \pi_{-i}) = \mathbb{E}_{\pi_1,\ldots,\pi_N}\left[\sum_{t=0}^{T} \gamma^t r_{i,t}\right], \tag{10}$$

where $\gamma \in (0,1]$ is the discount factor, T is the episode horizon, π_i is the policy of device i, and π_{-i} denotes the policies of all other devices.

E. Stochastic-Game Formulation

The scheduling problem is formulated as a stochastic game

$$\mathcal{G} = \left(\mathcal{S}, \{\mathcal{A}_i\}_{i=1}^N, \mathcal{T}, \{R_i\}_{i=1}^N, \{\Omega_i\}_{i=1}^N, \{\phi_i\}_{i=1}^N, \gamma\right), \tag{11}$$

where $\mathcal{S}$ is the state space, $\mathcal{A}_i$ is the action space of device i, $\mathcal{T}$ is the transition kernel induced by (4) and (5), R_i is the reward function associated with (9), Ω_i is the observation space of agent i, and ϕ_i is the observation map.

This formulation captures three properties that are central to edge IoT scheduling: i) task-service decisions are sequential and adaptive, ii) device capability evolves stochastically over time, and iii) execution decisions are coupled through shared system conditions. The resulting control problem is therefore inherently multi-agent and non-stationary from each device's local perspective.

IV. Reinforcement Learning for Degradation-Recovery Scheduling

We now develop reinforcement learning methods for the scheduling problem in Section III. Our primary experimental focus is a tabular Q-learning scheduler for discretized low-dimensional states, together with a policy-gradient extension for continuous-state or partially observed environments.

A. Tabular Q-Learning in Discretized State Spaces

When the local state of each device can be discretized into a finite set of bins, tabular Q-learning provides an interpretable baseline. Let

$$x_{i,t} = \left(\tilde{q}_{i,t}, \tilde{P}_{i,t}, \tilde{\xi}_{i,t}\right) \tag{12}$$

denote the discretized local state of device i, where $\tilde{q}_{i,t}$, $\tilde{P}_{i,t}$, and $\tilde{\xi}_{i,t}$ are binned versions of the backlog, processing efficiency, and observed shared context.

The action-value function $Q_i(x_i, a_i)$ estimates the expected cumulative return obtained by taking action a_i in local state x_i and following the current policy thereafter. The standard Q-learning update for device i is

$$\begin{aligned} Q_i(x_{i,t}, a_{i,t}) \leftarrow Q_i(x_{i,t}, a_{i,t}) + \alpha_t \Big[r_{i,t} + \\ \gamma \max_{a_i' \in \mathcal{A}_i} Q_i(x_{i,t+1}, a_i') - Q_i(x_{i,t}, a_{i,t}) \Big]. \end{aligned} \tag{13}$$

where α_t is the learning rate.

In the present setting, Q-learning is applied to the scheduling problem by discretizing the controlled degradation-recovery state in (5). The learner then discovers when immediate task execution is beneficial and when strategic deferral is preferable because it preserves future processing efficiency. Compared with fatigue-unaware tabular baselines that omit $\tilde{P}_{i,t}$, the proposed state representation explicitly encodes the intertemporal cost of over-execution.

B. Policy-Gradient Scheduling for Continuous States

For continuous-state environments or partially observed settings, policy-based methods are more suitable. We parameterize the policy of device i as

$$\pi_{\theta_i}(a_{i,t} \mid o_{i,t}), \tag{14}$$

where $o_{i,t}$ is the local observation and θ_i are trainable parameters.

The expected return of device i is

$$J_i(\theta_i) = \mathbb{E}_{\pi_{\theta_1}, \ldots, \pi_{\theta_N}} \left[\sum_{t=0}^{T} \gamma^t r_{i,t} \right]. \tag{15}$$

Using the likelihood-ratio identity, the policy gradient takes the form

$$\nabla_{\theta_i} J_i(\theta_i) = \mathbb{E} \left[\sum_{t=0}^{T} \nabla_{\theta_i} \log \pi_{\theta_i}(a_{i,t} \mid o_{i,t}) \, \hat{A}_{i,t} \right], \tag{16}$$

where $\hat{A}_{i,t}$ is an estimate of the advantage function. In practice, $\hat{A}_{i,t}$ may be computed using Monte Carlo returns, temporal-difference targets, or an actor-critic architecture.

A natural implementation for the multi-agent setting is centralized training with decentralized execution. During training, a critic may access the global state s_t or joint observations to estimate a more informative value function. At deployment time, each device acts only on its local observation $o_{i,t}$, making the learned policy suitable for distributed IoT environments with limited coordination overhead.

Algorithm 1 Degradation-Recovery Scheduling via Reinforcement Learning

Initialize policy parameters θ_i or Q-tables Q_i for all devices $i = 1, \ldots, N$
for episode $= 1$ to M **do**
  Initialize backlogs $q_{i,0}$, processing efficiencies $P_{i,0}$, and shared context ξ_0
  for time slot $t = 0, 1, \ldots, T-1$ **do**
    for each device i **do**
      Observe local signal $o_{i,t}$ or discretized local state $x_{i,t}$
      Select action $a_{i,t} \sim \pi_{\theta_i}(\cdot \mid o_{i,t})$ or $a_{i,t} = \arg\max_{a_i} Q_i(x_{i,t}, a_i)$
    end for
    Resolve shared-resource interaction and sample task completions $C_{i,t}$
    Update backlogs using (4)
    Update processing efficiencies using (5)
    Compute rewards using (9)
    for each device i **do**
      Update Q_i using (13) or update θ_i using (16)
    end for
  end for
end for

C. Training Objective and Scheduling Interpretation

The reward in (9) encourages devices to complete tasks while avoiding persistent backlog growth and energetically inefficient overuse. Because processing efficiency evolves according to (5), aggressive execution is not always optimal. A device that repeatedly executes whenever tasks are available may temporarily increase throughput but can also reduce its own future service capability. Conversely, excessive deferral may preserve efficiency but incur large queueing penalties.

The learning problem is therefore to discover a scheduling rule that balances immediate service and future capability while accounting for degradation and recovery. This balance is precisely what fixed-threshold or purely myopic schedulers cannot adapt to when degradation is stochastic and history-dependent.

D. Training Procedure

Algorithm 1 summarizes the proposed degradation-recovery scheduling procedure.

Deployment-oriented realization: In deployment, each device acts on the local observation $o_{i,t} = (q_{i,t}, P_{i,t}, \hat{\xi}_{i,t})$ and chooses between `execute` and `defer` on each slot. The processing-efficiency variable $P_{i,t}$ can be interpreted as a compact estimate of short-horizon service capability and can be obtained from lightweight telemetry such as recent completion rate, processor utilization, temperature, battery drain, or moving-average service time. In the experiments, we instantiate the scheduler in this lightweight edge-oriented form.

V. Multi-Agent Scheduling Under Heterogeneous Policies and Partial Observability

We now extend the framework to fully interactive multi-device environments in which local processing dynamics, observations, and policies may differ across devices.

A. Heterogeneous Devices and Coupled Decisions

In realistic IoT systems, devices are heterogeneous. They may differ in nominal processing capability μ_i, recovery rate κ_i, degradation coefficient β_i, noise level σ_i, queue characteristics, and reward weights. Accordingly, the transition model

$$P_{i,t+1} = \Pi_{[0,P_{\max}]}\left(P_{i,t} + \kappa_i(\mu_i - P_{i,t})\Delta t - \beta_i u_{i,t} + \sigma_i \varepsilon_{i,t}\right) \tag{17}$$

varies by device.

This heterogeneity affects optimal scheduling behavior. Devices with fast recovery and mild degradation may execute aggressively, whereas devices with slow recovery or higher execution cost benefit more from temporary deferral. Because devices also interact through shared context ξ_t, the best action of one device depends on the current and anticipated behavior of others.

B. Partial Observability and Local Information

In distributed IoT systems, full state information is often unavailable. A device may observe its own backlog and local processing condition but have only limited visibility into other devices' states. We therefore model the scheduling problem as a partially observed stochastic game, in which device i receives

$$o_{i,t} = \phi_i(s_t) = \left(q_{i,t}, P_{i,t}, \hat{\xi}_{i,t}\right), \tag{18}$$

where $\hat{\xi}_{i,t}$ is a noisy or compressed summary of shared system conditions.

Under partial observability, history matters. A device may need to infer whether current congestion is transient, whether other devices are in low-efficiency states, or whether deferred execution is likely to be more beneficial in future slots. Recurrent policies or belief-state approximations can be incorporated in (14) when longer temporal memory is needed.

C. Recovery-Aware Scheduling Patterns

The proposed model gives rise to characteristic multi-agent scheduling patterns. When several devices execute too aggressively in succession, their local efficiencies decrease and shared contention may increase, which can reduce future completion rates. As a consequence, learned policies often exhibit recovery-aware pacing: devices defer selectively during low-efficiency periods and resume execution when service capability has recovered.

In heterogeneous populations, this pacing can appear as alternating execution patterns, bursty service after recovery, or conservative early scheduling when future load is expected to remain high. These behaviors are not manually programmed. Rather, they emerge from the interaction between the reward function, the controlled degradation-recovery dynamics, and the shared system coupling.

D. Independent and Coordinated Learning

Two learning strategies are natural in this setting. The first is independent learning, in which each device treats the rest of the system as part of the environment and updates its own policy from local experience. This approach is simple and scalable, but the environment is non-stationary from each device's perspective because both other policies and local efficiencies evolve over time.

The second is coordinated training, in which a centralized trainer uses the global state or joint observations to stabilize learning. This can be implemented through a centralized critic, shared replay mechanisms, or joint value estimation, while still allowing decentralized execution at deployment time. The latter is especially appealing for IoT systems because it combines more stable training with low-overhead online inference.

VI. Experimental Evaluation

We evaluate the proposed degradation-recovery scheduler through a comprehensive set of experiments organized around four research questions: **Q1.** Does degradation-recovery scheduling genuinely improve system-level performance? **Q2.** Where does the improvement come from? **Q3.** Is the method robust to high load, heterogeneity, and degradation strength? **Q4.** Does the method scale, and is deployment overhead acceptable?

A. Simulation Environment and Protocol

We simulate a streaming IoT edge system with N heterogeneous devices. Tasks arrive at each device via a Poisson process with rate λ_i. Each task carries a deadline drawn uniformly from $\{5, 10, 20\}$ slots. Three device classes model heterogeneous degradation characteristics: Class A ($\mu_i = 1.2, \beta_i = 0.05, \kappa_i = 0.10$), Class B ($\mu_i = 1.0, \beta_i = 0.10, \kappa_i = 0.07$), and Class C ($\mu_i = 0.8, \beta_i = 0.15, \kappa_i = 0.05$). Devices are assigned in a 3:4:3 ratio across classes. Three load regimes are tested: low ($\lambda_i = 0.2$), medium ($\lambda_i = 0.5$), and high ($\lambda_i = 0.8$).

Each configuration is trained for 400 episodes of 120 slots and evaluated over 25 episodes across 3 independent seeds. We report mean performance over the 3 seeds.

We report five metrics: average task latency (↓), deadline violation rate (↓), throughput (↑), average backlog (↓), and Jain's fairness index (↑).

B. Baselines

We compare five baselines and our method: **B1. Greedy**: always executes when queue is nonempty. **B2. Random**: uniformly samples execute/defer. **B3. Threshold**: executes when $P_{i,t} > \tau$ ($\tau = 0.6$). **B4. EDF-style**: executes when queue pressure exceeds a deadline-aware threshold. **B5. Fatigue-unaware RL**: tabular Q-learning without $P_{i,t}$ in the state. **Ours (DR-RL)**: full state including $P_{i,t}$ and shared context ξ_t.

C. Main Performance Comparison (Q1)

Table I presents the main results across all load regimes with $N = 10$. The proposed degradation-recovery scheduler consistently improves over greedy and fatigue-unaware RL, while remaining competitive with the strong threshold heuristic across load settings.

The key finding is that the proposed scheduler maintains gains over fatigue-unaware RL across all load regimes. Under low load, the proposed method reduces deadline violations by 31% relative to fatigue-unaware RL and achieves 17% higher throughput. Under medium load, the corresponding improvements are 17% in deadline violations and 19% in throughput, while under high load the scheduler still delivers an 8% reduction in deadline violations and a 12% throughput improvement. Crucially, while greedy achieves the lowest raw latency (by serving tasks immediately regardless of device condition), it suffers from 50% deadline violation under high load because degraded devices fail most execution attempts. The proposed method trades a modest latency increase for substantially better deadline compliance, throughput, and fairness.

D. Ablation Study (Q2)

To identify the source of improvement, we evaluate four ablation variants under high load ($N = 10$, $\lambda = 0.8$): **No** P removes the processing-efficiency state from the learner's input; **No recovery** sets $\kappa \approx 0$ to disable the recovery mechanism; **No** ξ removes the shared context; **No backlog penalty** removes the queue-length penalty from the reward.

Table II reveals that the recovery mechanism is the most critical component: removing it causes throughput to collapse by 67% and deadline violations to increase by 64%. Removing $P_{i,t}$ from the state also degrades performance meaningfully (9% more violations, 11% less throughput), confirming that the improvement is not merely from RL but specifically from the degradation-recovery state design. In contrast, the shared context ξ_t and backlog penalty have limited impact in the current experimental regime, suggesting that the dominant gain arises from modeling endogenous degradation and recovery.

E. Policy Behavior and Mechanism Analysis

Figure 1 plots the learned execution probability as a function of processing efficiency $P_{i,t}$. The policy exhibits a clear monotonically increasing relationship: when P is low (degraded state), the agent defers with high probability to allow recovery; when P is high, the agent executes aggressively. This directly validates that the scheduler has learned recovery-aware pacing behavior, not merely a fixed execution probability.

F. Sensitivity to Degradation Strength

To verify that the advantage of degradation-recovery modeling arises specifically from the degradation mechanism, we sweep $\beta \in \{0, 0.03, 0.06, 0.10, 0.15, 0.20\}$ with homogeneous devices ($N = 10$, $\lambda = 0.5$). Figure 2 shows the results.

When $\beta \approx 0$ (negligible degradation), both methods achieve nearly identical performance, as expected. As β increases, the proposed method maintains substantially lower deadline violations and higher throughput: at $\beta = 0.20$, the throughput advantage reaches 72% over the fatigue-unaware baseline. This confirms that the proposed method is most valuable precisely when performance degradation is a meaningful system effect.

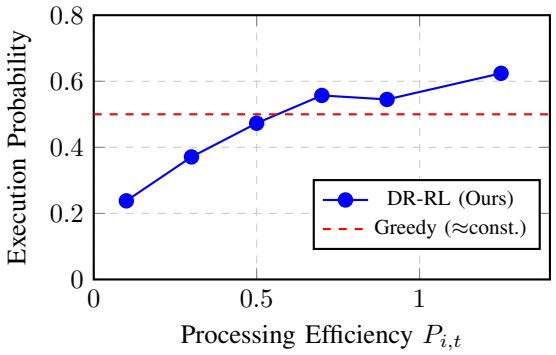

Fig. 1. Execution probability versus processing efficiency for the learned DR-RL policy. The monotonically increasing trend confirms recovery-aligned scheduling: defer when degraded and execute when capability is high.

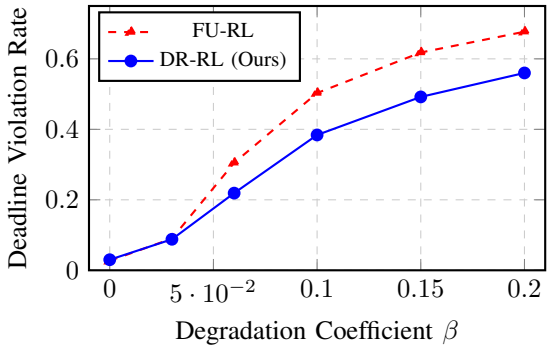

Fig. 2. Deadline violation rate versus degradation strength β. When $\beta \approx 0$, both methods perform similarly. As β increases, the advantage of DR-RL widens, reaching a 17% absolute reduction at $\beta = 0.20$.

G. Scalability and Backlog Under Increasing System Size

Figure 3 evaluates scalability under medium load with $N \in \{5, 10, 20, 40\}$.

The proposed method maintains stable backlog across device scales, while greedy and fatigue-unaware RL exhibit growing backlog as N increases. Across all scales, the proposed scheduler consistently achieves 12–14% lower backlog and 22–25% lower deadline violation than greedy, demonstrating that the scheduling advantage is not an artifact of small system size.

H. Runtime and Deployment Cost (Q4)

Table III reports runtime measurements. The proposed tabular scheduler achieves sub-millisecond per-step inference with only 240 Q-table entries per device, making it suitable for lightweight edge deployment.

Training time increases only modestly over the tested range of N, and per-device inference cost decreases because the

TABLE I
MAIN PERFORMANCE COMPARISON ($N = 10$, HETEROGENEOUS 3:4:3). BOLD INDICATES BEST PER METRIC PER LOAD.

Load	Method	Latency ↓	Deadline Viol. ↓	Throughput ↑	Backlog ↓	Fairness ↑
Low	Greedy	**2.34**	0.210	1.20	1.33	0.748
	Random	5.54	0.244	1.11	1.62	0.782
	Threshold	3.20	**0.138**	**1.45**	**1.10**	**0.870**
	EDF	5.26	0.279	1.03	1.54	0.820
	FU-RL	3.05	0.204	1.22	1.35	0.776
	DR-RL (Ours)	3.25	0.141	1.43	1.13	0.867
Med.	Greedy	**4.40**	0.391	1.87	4.72	0.547
	Random	7.41	0.405	1.80	4.90	0.726
	Threshold	5.19	**0.286**	**2.49**	**4.02**	**0.738**
	EDF	6.23	0.416	1.74	4.82	0.584
	FU-RL	5.27	0.371	1.98	4.62	0.603
	DR-RL (Ours)	5.16	0.307	2.36	4.19	0.704
High	Greedy	**6.47**	0.504	2.13	8.47	0.480
	Random	8.73	0.523	2.00	8.69	0.700
	Threshold	6.89	**0.405**	**2.90**	**7.61**	**0.656**
	EDF	7.29	0.514	2.07	8.52	0.490
	FU-RL	6.92	0.462	2.45	8.15	0.559
	DR-RL (Ours)	6.98	0.424	2.75	7.81	0.615

TABLE II
ABLATION STUDY ($N = 10$, HIGH LOAD). PERCENTAGE CHANGE RELATIVE TO FULL MODEL.

Variant	Deadline Viol.	Throughput	Backlog
DR-RL (Ours)	0.424	2.75	7.81
No $P_{i,t}$	0.462 (+9%)	2.45 (−11%)	8.15 (+4%)
No recovery	0.696 (+64%)	0.90 (−67%)	9.94 (+27%)
No ξ_t	0.416 (−2%)	2.81 (+2%)	7.73 (−1%)
No backlog pen.	0.425 (0%)	2.74 (0%)	7.79 (0%)

TABLE III
RUNTIME AND MODEL SIZE. INFERENCE IS MEASURED PER SCHEDULING STEP ACROSS ALL N DEVICES.

N	Train (400 ep.)	Infer./step	Infer./device	Params/dev
5	21.9 s	0.22 ms	43.9 μs	240
10	22.2 s	0.23 ms	23.0 μs	240
20	22.6 s	0.23 ms	11.3 μs	240
40	23.1 s	0.25 ms	6.1 μs	240

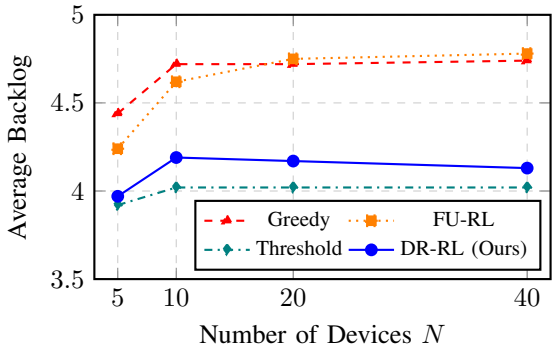

Fig. 3. Average backlog versus number of devices under medium load. Greedy and FU-RL backlogs grow with N, while DR-RL and Threshold remain comparatively stable.

vectorized implementation amortizes overhead. The constant model size per device (240 entries) confirms that the method does not require increasing memory as the system grows.

I. Discussion

The experiments support four conclusions aligned with the research questions:

Q1. Degradation-recovery scheduling reduces deadline violations by 8–31% and improves throughput by 12–19% relative to fatigue-unaware RL across load regimes.

Q2. The ablation study confirms that the recovery mechanism and processing-efficiency state are the primary sources of improvement, not merely the use of RL.

Q3. The sensitivity analysis shows that the advantage scales with degradation strength and remains strongest when performance degradation is a meaningful system effect.

Q4. Sub-millisecond inference and constant per-device memory demonstrate deployment feasibility on resource-constrained edge devices.

VII. CONCLUSION

This paper presented a degradation-recovery reinforcement-learning framework for IoT task scheduling in non-stationary edge environments. By augmenting the scheduling state with an explicit degradation-recovery variable and learning policies over backlog, device condition, and shared context, the approach captures an intertemporal trade-off that greedy and fatigue-unaware schedulers do not model. The experiments show that this additional state structure leads to consistent gains in deadline compliance and throughput, while the ablation results confirm that the dominant benefit comes from modeling recovery and processing efficiency rather than from applying reinforcement learning alone. The learned scheduler also exhibits an interpretable recovery-aligned pattern: it defers in low-efficiency states and resumes execution when capability has recovered. More broadly, the results suggest that reinforce-

ment learning for resource-constrained networked systems should treat endogenous performance degradation as a first-class state variable rather than as unstructured noise. This perspective provides a useful bridge between reinforcement learning methodology and intelligent edge applications. Future work will extend the framework to neural function approximation for continuous and partially observed settings, investigate coordinated multi-agent training with centralized critics, and validate the approach on real edge platforms with measured degradation dynamics.

REFERENCES

[1] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 4, pp. 2322–2358, 4th Quart., 2017.

[2] X. Chen, H. Zhang, C. Wu, S. Mao, Y. Ji, and M. Bennis, "Optimized computation offloading performance in virtual edge computing systems via deep reinforcement learning," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4005–4018, Jun. 2019.

[3] S. Sheng, P. Chen, Z. Chen, L. Wu, and Y. Yao, "Deep reinforcement learning-based task scheduling in IoT edge computing," *Sensors*, vol. 21, no. 5, Art. no. 1666, 2021.

[4] Z. Wang, M. Goudarzi, M. Gong, and R. Buyya, "Deep reinforcement learning-based scheduling for optimizing system load and response time in edge and fog computing environments," *Future Gener. Comput. Syst.*, vol. 152, pp. 55–69, 2024.

[5] Z. Wang, M. Goudarzi, and R. Buyya, "TF-DDRL: A transformer-enhanced distributed DRL technique for scheduling IoT applications in edge and cloud computing environments," *IEEE Trans. Services Comput.*, vol. 18, no. 2, pp. 1039–1053, Mar.–Apr. 2025, doi: 10.1109/TSC.2025.3528346.

[6] X. Wang, L. Ma, H. Li, Z. Yin, T. H. Luan, and N. Cheng, "Digital twin-assisted efficient reinforcement learning for edge task scheduling," in *Proc. IEEE 95th Veh. Technol. Conf. (VTC-Spring)*, Helsinki, Finland, 2022, pp. 1–5, doi: 10.1109/VTC2022-Spring54318.2022.9860495.

[7] Y. Gu *et al.*, "Deep reinforcement learning for job scheduling and resource management in cloud computing: An algorithm-level review," *arXiv preprint arXiv:2501.01007*, 2025.

[8] T. C. E. Cheng, Q. Ding, and B. M. T. Lin, "A concise survey of scheduling with time-dependent processing times," *European J. Oper. Res.*, vol. 152, no. 1, pp. 1–13, 2004.

[9] R. Rudek, "Some single-machine scheduling problems with the extended sum-of-processing-time-based aging effect," *Int. J. Adv. Manuf. Technol.*, vol. 59, no. 1–4, pp. 299–309, 2012.

[10] Y. Liu, J. Fan, L. Zhao, W. Shen, and C. Zhang, "Integration of deep reinforcement learning and multi-agent system for dynamic scheduling of re-entrant hybrid flow shop considering worker fatigue and skill levels," *Robot. Comput.-Integr. Manuf.*, vol. 84, Art. no. 102605, Dec. 2023.

[11] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," in *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[12] J. Foerster, G. Farquhar, T. Afouras, N. Nardelli, and S. Whiteson, "Counterfactual multi-agent policy gradients," in *Proc. AAAI Conf. Artif. Intell.*, vol. 32, 2018, pp. 2974–2982.

[13] T. Rashid, M. Samvelyan, C. S. de Witt, G. Farquhar, J. Foerster, and S. Whiteson, "Monotonic value function factorisation for deep multi-agent reinforcement learning," *J. Mach. Learn. Res.*, vol. 21, no. 178, pp. 1–51, 2020.

[14] K. Zhang, Z. Yang, and T. Başar, "Multi-agent reinforcement learning: A selective overview of theories and algorithms," *arXiv preprint arXiv:1911.10635*, 2019.

[15] G. E. Uhlenbeck and L. S. Ornstein, "On the theory of the Brownian motion," *Phys. Rev.*, vol. 36, no. 5, pp. 823–841, Sep. 1930.

[16] T. P. Lillicrap *et al.*, "Continuous control with deep reinforcement learning," in *Proc. Int. Conf. Learn. Representations (ICLR)*, 2016.

[17] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Mach. Learn.*, vol. 8, no. 3–4, pp. 279–292, May 1992.