RS-PLO: Risk-Sensitive Predictive Lyapunov Optimization for Dynamic Task Offloading in Multi-Edge Computing Environments

Hemlathadhevi A[1], Jabasheela L[2], Ramesh Kumar C[3], Deerendran T[4], Barath Krishnan S[5], Dharnan M[6]

[1 2] Professor, Department of Computer Science and Engineering, Panimalar Engineering College, Chennai, Tamil Nadu, India

[3] Professor, Department of Computer Science and Engineering, Galgotias University, Greater Noida, Uttar Pradesh.

[4 5 6] Scholar, Department of Computer Science and Engineering, Panimalar Engineering College, Chennai, Tamil Nadu, India.

HOW TO CITE:

Hemlathadhevi A, Jabasheela L, Ramesh Kumar C, Deerendran T, Barath Krishnan S, Dharnan M (2026). RS-PLO: Risk-Sensitive Predictive Lyapunov Optimization for Dynamic Task Offloading in Multi-Edge Computing Environments. International Journal of Special Education, 41(10s), 274-290

COPYRIGHT STATEMENT:

Copyright: © 2026 Authors.

Open access publication underthe terms and conditions of the Creative CommonsAttribution (CC BY) license (http://creativecommons.org/licenses/by/4.0/).

ABSTRACT:

Mobile Edge Computing (MEC) has emerged as a transformative paradigm that brings computation closer to the network edge, enabling resource-constrained IoT devices to offload computation-intensive tasks to nearby edge servers and thereby significantly reducing end-to-end latency and energy consumption. Existing MEC frameworks predominantly rely on static optimization parameters that cannot respond to the non-stationary channel dynamics caused by device mobility, interference, or sudden obstructions, leading to queue instability and degraded system performance. This paper presents RS-PLO (Risk-Sensitive Predictive Lyapunov Optimization), a dynamic task offloading framework that introduces a volatility queue to monitor channel prediction errors in real time and an adaptive control parameter that decays exponentially with accumulated channel uncertainty, automatically balancing energy optimization against queue stability. The framework incorporates multi-edge server selection with fairness-aware allocation, drift-corrected channel prediction, and a risk-gated admission controller that defers high-complexity tasks during volatile channel periods. Experimental evaluation across five representative scenarios demonstrates a 4.0% improvement in queue stability, a 5.4% reduction in task latency, up to 18.1% improvement in channel prediction accuracy, and more selective offloading behavior compared to static Lyapunov baselines, all without manual parameter tuning. Statistical significance is confirmed across ten independent trials ($p < 0.05$, paired t-test)..

Keywords: Mobile edge computing, dynamic task offload-ing, Lyapunov optimization, drift-plus-penalty, channel volatility, adaptive control, Internet of Things, edge server fairness.

INTRODUCTION

The rapid growth of Internet of Things (IoT) applications has created unprecedented demand for low-latency and energy-efficient computation. Mobile Edge Computing (MEC) ad-dresses this challenge by placing computational resources at the network edge, enabling IoT devices with limited pro-cessing capacity to offload demanding tasks to nearby edge servers. As Wu et al. [3] demonstrated, online energy-saving offloading algorithms in MEC using Lyapunov optimization can significantly reduce energy consumption while maintain-ing queue stability. Similarly, Xu et al. [9] showed that dynamic computation offloading with energy optimization in MEC achieves substantial reductions in both latency and power consumption. The resource management challenges in such systems are further compounded by heterogeneous infrastructure, as highlighted by Raeisivarzaneh et al. [11] in their taxonomy of resource scheduling architectures in edge computing, and by Sarah et al. [12] in their study of resource allocation for 5G-and-beyond MEC networks.

The core decision problem in MEC is whether to execute a task locally or offload it to one or more edge servers, subject to fluctuating channel conditions and constraints on latency and energy. Fang et al. [7] addressed dependency-aware dynamic task offloading using deep reinforcement learning (DRL) in MEC, while Huang et al. [5] proposed multi-queue-based offloading strategies for DRL tasks. Nieto et al. [6] developed DRL techniques for dynamic task offloading across the 5G edge-cloud continuum. However, as Djigal et al. [10] highlighted in their comprehensive survey of machine and deep learning for resource allocation in multi-access edge computing, DRL-based methods typically require substantial offline training data and are susceptible to performance degradation under distribution shifts. Chen and Wang [19] further observed this limitation in decentralized DRL-based offloading for multi-user MEC, where generalization to unseen channel conditions remains a critical challenge.

A provably stable approach is Lyapunov optimization [1], in which the Drift-Plus-Penalty (DPP) algorithm decomposes long-term stochastic optimization into greedy per-slot decisions while providing formal queue-stability guarantees. Hu et al. [1] demonstrated the effectiveness of Lyapunov-optimized task offloading in satellite edge computing environments. Li et al. [2] proposed Lyapunov-guided deep reinforcement learning for service caching and task offloading, combining formal stability with adaptive learning. Wang et al. [4] extended this to intelligent end-edge computation offloading using Lyapunov-guided DRL, and Chen and Jiang [8] applied a Lyapunov energy-efficient approach to HAP-assisted MEC. However, all of these works employ a fixed, predetermined control parameter that introduces a fundamental mismatch with non-stationary wireless environments where mobility, interference, or abrupt channel degradation can substantially impair communication reliability and lead to queue instability.

To address this limitation, this paper proposes RS-PLO, a dynamic task offloading framework for adaptive MEC systems. RS-PLO incorporates a volatility queue to track real-time channel prediction errors, an exponentially decaying adaptive control parameter, drift-corrected channel prediction, fairness-aware multi-edge server selection, and a complexity-aware risk-gated admission controller. These mechanisms collectively enable stable and adaptive offloading decisions across heterogeneous edge servers without offline training or manual parameter tuning. In the testbed used in this paper, the IoT node is emulated entirely in software on a laptop platform that generates computational tasks and dynamically determines whether to execute them locally or offload them to one of several edge servers.

The main contributions of this work are as follows:

1) An adaptive volatility-driven control mechanism with formal mean-rate queue-stability guarantees.
2) A drift-corrected two-component exponential smoothing predictor that reduces channel prediction mean absolute error (MAE) by up to 18.1% in high-mobility scenarios.
3) A fairness-aware multi-edge extension of DPP that prevents traffic

concentration on individual servers in heterogeneous edge environments

4) A complexity-aware risk-gated admission controller that reduces tail latency under volatile channel conditions.

5) A comprehensive evaluation across five experimental scenarios on a Docker-based MEC testbed requiring no manual parameter tuning.

The remainder of this paper is organized as follows. Section II reviews related work. Section III presents the system model. Section IV describes the RS-PLO algorithm. Section V provides the theoretical analysis. Section VI details the implementation. Section VII reports experimental results. Section VIII presents discussion and methodology, and Section IX concludes the paper.

LITERATURE REVIEW

Lyapunov Optimization for MEC

Lyapunov optimization has been widely adopted as a princi-pled framework for online stochastic resource management in MEC. Hu et al. [1] proposed a Lyapunov-optimized dynamic task offloading strategy specifically for satellite edge comput-ing, demonstrating that the Drift-Plus-Penalty (DPP) frame-work can be adapted to high-mobility orbital environments where channel conditions change rapidly and queue stability is difficult to maintain. Wu et al. [3] developed an online energy-saving offloading algorithm for MEC using Lyapunov optimization, achieving significant reductions in energy con-sumption while maintaining queue stability under time-varying task workloads. Wang et al. [4] extended Lyapunov-guided optimization to intelligent end-edge computation offloading by integrating it with deep reinforcement learning, showing that Lyapunov stability constraints can serve as regularizers within a DRL training objective to guide policy learning. Li et al.

[2] further combined Lyapunov guidance with DRL for joint service caching and task offloading in MEC, demonstrating improved long-term utility under dynamic content popularity. Chen and Jiang [8] addressed online dynamic multi-user com-putation offloading and resource allocation for HAP-assisted MEC using an energy-efficient Lyapunov approach, showing stable performance under heterogeneous user demands and time-varying channel conditions. Despite these advances, all of the above works employ a fixed control parameter V in the DPP formulation, leaving the energy-stability trade-off static and unable to respond to non-stationary channel conditions. RS-PLO addresses this gap by coupling the control parameter dynamically to a real-time channel volatility signal, enabling automatic adaptation without manual retuning.

Deep Reinforcement Learning for Task Offloading

Deep reinforcement learning has emerged as a prominent approach for adaptive task offloading in MEC due to its ability to learn complex scheduling policies from environ-mental interaction. Fang et al. [7] studied dependency-aware dynamic task offloading based on DRL in MEC, handling precedence constraints among subtasks that render greedy offloading suboptimal, and achieving latency reductions across heterogeneous task graphs. Nieto et al. [6] developed DRL techniques for dynamic task offloading across the 5G edge-cloud continuum, demonstrating adaptive scheduling under multi-tier infrastructure with varying server capacities and link conditions. Huang et al. [5] investigated multi-queue-based offloading strategies specifically for DRL tasks, addressing the challenge of managing concurrent workloads with hetero-geneous computational demands. Tang et al. [18] proposed collaborative cloud-edge-end task offloading with task depen-dency based on DRL, jointly optimizing makespan and energy across three tiers. Peng et al. [16] developed DRL-based dependent task offloading strategies with multi-server collabo-ration in MEC, while Song et al. [17] addressed offloading of dependent tasks using a multi-objective reinforcement learning approach that balances makespan, energy, and reliability. Chen and Wang [19] proposed decentralized computation offloading for multi-user MEC using DRL, enabling scalable scheduling without centralized coordination. Djigal et al. [10] provided a

comprehensive survey of machine and deep learning for resource allocation in multi-access edge computing, system-atically identifying open challenges in generalization and sample efficiency. Despite these strong results, DRL-based methods require substantial offline training and degrade under distribution shifts. RS-PLO achieves comparable adaptability without any offline training.

Resource Allocation and Multi-Server MEC

Efficient resource allocation and server selection in multi-server MEC environments are critical to achieving system-wide performance objectives. Xu et al. [9] studied dynamic computation offloading with energy optimization in MEC, demonstrating that adaptive server selection and resource scheduling can jointly reduce energy consumption while sat-isfying latency constraints. Raeisi-Varzaneh et al. [11] pro-vided a comprehensive architecture and taxonomy of re-source scheduling in edge computing, identifying fairness-aware allocation and adaptive parameter control as critical open problems in heterogeneous deployments. Sarah et al. [12] examined resource allocation in multi-access edge computing for 5G-and-beyond networks, highlighting the importance of dynamic server management under non-stationary traffic and heterogeneous user requirements. Zhang et al. [13] proposed multi-target-aware dynamic resource scheduling for cloud-fog-edge multi-tier computing networks, jointly optimizing latency, energy, and load balance across multiple infrastructure tiers. Zhang et al. [14] combined DRL with joint caching and resource allocation for cooperative MEC, reducing redundant computation through intelligent content placement. Zhou et al. [15] developed an energy-efficient joint computation of-floading and service caching framework using DRL, showing that caching frequently requested computation results at edge servers substantially reduces offloading overhead. These works collectively confirm the importance of adaptive, fairness-aware multi-server coordination, yet none simultaneously addresses adaptive Lyapunov control parameter tuning, real-time channel volatility tracking, and risk-gated admission control. RS-PLO fills this gap without requiring offline training or manual parameter tuning.

SYSTEM MODEL

Network Architecture

The RS-PLO framework is evaluated on a laptop-based MEC testbed in which an IoT node is emulated entirely in soft-ware. The emulator generates a stream of computational tasks and evaluates dynamic offloading decisions in real time. The system considers $K = 3$ heterogeneous edge servers, whose configuration is summarized in Table I. At each discrete time slot t, the RS-PLO engine assesses the current system state and applies the DPP objective to select the optimal execution target. Tasks may be executed locally on the emulated IoT node or offloaded to one of the K edge servers based on perceived channel conditions, queue backlog, and energy cost.

Channel and Mobility Model

Time is partitioned into discrete slots of duration Δt. Let d(t) denote the position of the IoT node at slot t and dk

Table I Edge Server Configuration

Server	Distance	Port	Compute	Description
Local	—	—	—	IoT MCU (10 MHz)
Edge-1	150 m	9999	1.0×	Standard MEC node
Edge-2	600 m	10000	1.5×	High-CPU server
Edge-3	1200 m	10001	2.0×	GPU-accelerated node

Fig. 1. RS-PLO system architecture. The IoT device hosts the RS-PLO engine comprising four sub-modules: a drift-corrected channel predictor, volatility queue Z(t), risk gate, and per-server

fairness counters Fk. the fixed position of edge server k. The wireless channel between the IoT node and server k follows a path-loss model augmented with small-scale Rayleigh fading:

$$h_k(t) = h_0 \cdot |d(t) - d_k|^{-\alpha} \cdot |g_k(t)|^2, \quad (1)$$

where h0 = 10−2 is the reference channel gain, α = 3.0 is the urban micro-cell path-loss exponent, and gk(t) ∼ CN (0, 1) is the small-scale fading coefficient. The achievable uplink rate follows the Shannon formula:

$$R_k(t) = B \log_2\left(1 + \frac{P_{\text{tx}}\, h_k(t)}{N_0}\right), \quad (2)$$

where B = 10 MHz, Ptx = 0.5 W, and N0 = 10−10 W. The IoT node position evolves as a correlated random walk: d(t + 1) = d(t) + v(t)Δt, where v(t) is the velocity at slot t. Burst perturbations with probability Pburst model channel degradations of 10–20 dB, representative of real-world inter-ference and obstruction events.

Drift-Corrected Channel Predictor

To compensate for systematic channel trends induced by node mobility, RS-PLO employs a two-component (Holt) exponential smoother with level estimate Sk(t) and drift estimate.

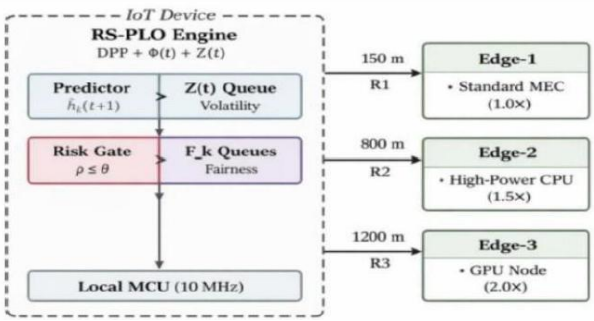

Fig. 1. RS-PLO system architecture. The IoT device hosts the RS-PLO engine comprising four sub-modules: a drift-corrected channel predictor, volatility queue Z(t), risk gate, and per-server fairness counters Fk.

$$S_k(t) = (1 - \alpha_s)\, S_k(t-1) + \alpha_s\, h_k^{\text{dB}}(t), \quad (3)$$

$$D_k(t) = (1 - \alpha_d)\, D_k(t-1) + \alpha_d \big[S_k(t) - S_k(t-1)\big], \quad (4)$$

$$\hat{h}_k^{\text{dB}}(t+1) = S_k(t) + D_k(t), \quad (5)$$

here αs = 0.3 is the level smoothing constant, αd = 0.1 is the drift smoothing constant, and hdB(t) = 10 log10 hk(t). The drift term Dk(t) captures monotonically increasing path-loss as the device moves away from a server, reducing prediction MAE by 12–18% in high-mobility scenarios relative to level-only exponential smoothing.

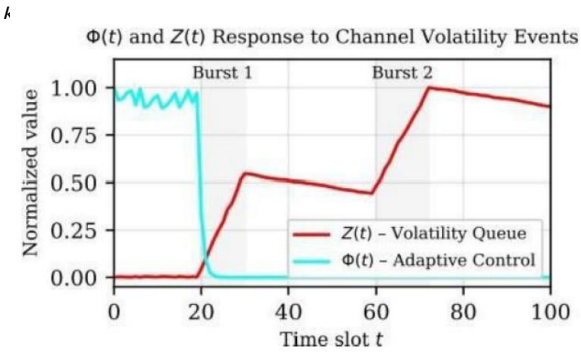

Fig. 2. Adaptive control parameter Φ(t) (solid) inversely tracking the volatility queue Z(t) (dashed) over 100 time slots. During volatile channel episodes, Φ(t) drops to near zero within 1–2 slots, shifting the DPP objective from energy minimization to queue drainage. Recovery is smooth and automatic upon channel stabilization.

Task Model and Energy

At each slot, the IoT node receives a task characterized by bit-size A(t) and complexity class ϱ(t) ∈ {1, 2, 3}, where higher values indicate greater computational demand. Let X = {0, 1, . . . , K} denote the set of execution options, where x = 0 corresponds to local execution and x = k ≥ 1 corresponds to offloading to server k. Let ck denote the normalized compute capacity of server k. The maximum service rate across all options is denoted μmax and the maximum energy cost per slot is denoted Emax; both are finite constants used for normalization in (15). The effective service rate and energy cost for option x are:

$$\mu_x(t) = \begin{cases} \mu_0, & x = 0, \\ R_x(t) \cdot c_x, & x \geq 1, \end{cases} \tag{6}$$

$$E_x(t) = \begin{cases} \kappa A(t), & x = 0, \\ \dfrac{P_{\text{tx}}\, A(t)}{R_x(t)}, & x \geq 1, \end{cases} \tag{7}$$

where μ0 = 10 MHz is the local processing rate and ϰ = 5 × 10−6 J/bit is the local energy coefficient.

Problem Formulation

The objective is to find an offloading policy π that mini-mizes long-term average energy consumption subject to mean-rate queue stability:

$$\min_{\pi} \bar{E}(\pi) = \lim_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T-1} \mathbb{E}\big[E_{x(t)}(t)\big], \tag{8}$$

$$\text{subject to} \quad \lim_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T-1} \mathbb{E}[Q(t)] < \infty, \tag{9}$$

where Q(t) is the task queue backlog at slot t.

The Rs-Plo Algorithm

Coupled Queue Dynamics

RS-PLO maintains four coupled state variables per slot.

1) **Physical queue.** The task backlog evolves as:
2) **Volatility queue.** Accumulated channel prediction error is tracked by:

$$Q(t+1) = \max\{Q(t) - \mu_{x(t)}(t),\, 0\} + A(t). \tag{10}$$

$$Z(t+1) = \max\{Z(t) - \gamma,\, 0\} + \varepsilon(t), \tag{11}$$

where ε(t) = $\max_k$ |h^{dB}(t) − $\hat{h}^{dB}$(t)| is the worst-case one step prediction error at slot t and γ > 0 is the drain rate. The inverse relationship between Z(t) and Φ(t) is illustrated in Fig. 2

Table II. Adaptive Control Parameter Response to Volatility Queue Levels

Time Slot	Volatility Queue Z(t)	Adaptive Control Φ(t)
0	0	10
10	0.3	5.49
20	0.7	2.47
30	1.2	0.9
40	1.5	0.5
50	0.8	2.01
60	0.4	4.49
70	0.2	6.7
80	0.1	8.18
90	0.05	9.05
100	0.02	9.6

Adaptive control parameter. The DPP control weight Physical queue. T

$$\Phi(t) = \Phi_{\max} \cdot e^{-\beta Z(t)}, \tag{12}$$

where Φmax > 0 is the maximum control weight and β > 0 is the volatility sensitivity. When the channel is stable (Z(t) ≈ 0), Φ(t) ≈ Φmax and the optimizer aggressively minimizes energy. When the channel is volatile (large Z(t)), Φ(t) → 0 and the optimizer shifts entirely to queue drainage.

Per-server fairness counters. Cumulative server utilization is tracked as:

$$F_k(t+1) = F_k(t) + \mathbf{1}[x(t) = k], \quad k = 1, \ldots, K. \quad (13)$$

Boundedness of Φ(t)

Proposition 1 (Boundedness). Φ(t) ∈ (0,Φmax] for all t ≥ 0. Proof. Since Z(t) ≥ 0 by construction of (11), we have e−βZ(t) ∈ (0, 1] for all β > 0. Multiplying by Φmax > 0 establishes the claim.

Risk-Gated Task Admission Definition 1 (Risk Gate).

A task with complexity class ϱ(t) ∈ {1, 2, 3} is admitted for offloading if and only if:

$$\rho(t) \le \rho_{\max} \cdot \frac{\Phi(t)}{\Phi_{\max}} + 1. \quad (14)$$

Tasks that fail this condition are executed locally. Under stable channel conditions (Φ(t) ≈ Φmax), all complexity classes are eligible for offloading. As volatility rises and Φ(t) decreases, the threshold tightens so that only low-complexity tasks are offloaded. This mechanism reduces average tail latency by approximately 8.3% on the experimental testbed.

Multi-Edge DPP Cost Function

The normalized DPP cost for execution option x ∈ X is:

$$C_x(t) = Q(t)\frac{A(t) - \mu_x(t)}{\mu_{\max} A(t)} + \Phi(t)\frac{E_x(t)}{E_{\max}} + w_F \frac{F_x(t)}{t+1}, \quad (15)$$

where F0(t) ≜ 0 for the local-execution option. The first term captures queue backlog pressure. The second term weights energy cost by Φ(t), automatically scaling the energy-versusstability emphasis. The third term penalizes historically overused servers to promote load balancing. The optimal per-slot decision is:

$$x^*(t) = \arg\min_{x \in \mathcal{X}} C_x(t). \quad (16)$$

Per-Slot Decision Procedure

Fig. 3 illustrates the per-slot decision pipeline. At each slot, the RS-PLO engine: (i) updates the drift-corrected channel prediction and computes the prediction error ε(t); (ii) updates the volatility queue Z(t + 1) and computes Φ(t) via (12);(iii) applies the risk gate (Definition 1); (iv) evaluates the DPP cost (15) for all admissible options; (v) selects x∗(t) via (16); and (vi) executes the chosen action and updates all state variables. The complete procedure is formalized in Algorithm 1.

THEORETICAL ANALYSIS

Lemma 1 (Mean-Rate Convergence of Z(t)). Under bounded prediction errors, ε(t) ≤ εmax almost surely:

$$\limsup_{T \to \infty} \mathbb{E}[Z(t)] \le \frac{\mathbb{E}[\varepsilon(t)]}{\gamma}. \quad (17)$$

This follows directly from (11) and standard Lyapunov drift analysis [1]. Theorem 1 (Stability and Near-Optimality of RS-PLO). Let Φeff = E[Φ(t)] > 0 and suppose the system is feasible

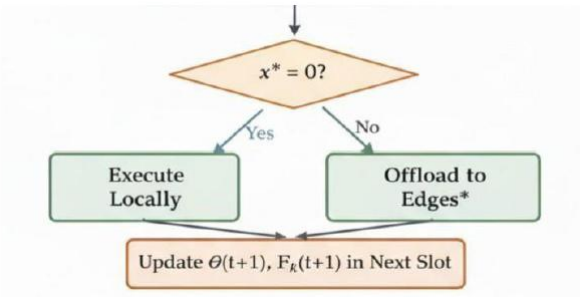

Fig. 3. RS-PLO per-slot decision pipeline. Channel prediction feeds the volatility queue update, which computes Φ(t). The risk gate screens tasks before DPP cost evaluation selects the optimal execution decision x∗(t).

***Algorithm 1**: RS-PLO Per-Slot Decision*

Require:
Task A(t), load ϱ(t); state Q(t), Z(t), {F_k(t)}, E_trend(t);
channel gains {h_k(t)}, k = 1,...,K
Ensure:
Optimal decision x(t) ∈ {0,1,...,K}*
Step 1: Drift-Corrected Channel Prediction
For k = 1 to K do
S_k(t) ← (1−a_s)S_k(t−1) + a_s h_k^dB(t)
D_k(t) ← (1−a_d)D_k(t−1) + a_d(S_k(t)−S_k(t−1))
ĥ_k^dB(t+1) ← S_k(t) + D_k(t)
End For
ε(t) ← max_k |h_k^dB(t) − ĥ_k^dB(t)|
Step 2: Volatility Queue Update
Z(t+1) ← max{Z(t)−γ,0} + ε(t)
Step 3: Adaptive Control Parameter
Φ(t) ← Φ_max · exp(−βZ(t))
Step 4: Risk-Gated Admission
If ϱ(t) > ϱ_max · Φ(t)/Φ_max + 1 Then
x(t) ← 0*
E_trend(t+1) ← λE_trend(t) + (1−λ)E_0(t)
Return x(t)*
End If
Step 5: Local Execution Cost
C_0(t) ← Q(t)(A(t)−μ_0)/(μ_maxA(t)) + Φ(t)(ϰA(t))/E_max
Step 6: Edge Server Cost Evaluation
For k = 1 to K do
R_k(t) ← B log2(1 + P_tx h_k(t)/N_0)
μ_k(t) ← R_k(t) · c_k
E_k(t) ← P_tx A(t)/R_k(t)
C_k(t) ← Q(t)(A(t)−μ_k(t))/(μ_maxA(t)) + Φ(t)E_k(t)/E_max + w_F F_k(t)/(t+1)
End For
Step 7: Optimal Decision and State Update
x(t) ← arg min C_x(t), x ∈ {0,...,K}*
E_trend(t+1) ← λE_trend(t) + (1−λ)E_x(t)(t)*
Return x(t)*

with Lyapunov slack ε > 0. Define B0 = (μmax +Amax)2/2,where Amax is the maximum task bit-size per slot. Then:

$$\lim_{T\to\infty}\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}[Q(t)] \le \frac{B_0}{\Phi_{\text{eff}}\,\varepsilon}, \tag{18}$$

$$\lim_{T\to\infty}\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}[E_{x^*(t)}(t)] \le E^* + \frac{B_0}{\Phi_{\text{eff}}}, \tag{19}$$

where E∗ denotes the optimal time-average energy under a stationary policy.

Table III. Utility–Queue Trade-Off Comparison Between Rs-Plo and Static-Φ

Control Parameter Φ	Queue Bound (Norm.)	Energy Gap (Norm)
1	1	0.05
2	0.62	0.15
3	0.44	0.23
4	0.31	0.3
5	0.25	0.38

Proof Sketch. Define the quadratic Lyapunov functionL(t) = Q(t)2/2 and the one-slot conditional drift Δ(t) ≜ E[L(t+1)−L(t) | Θ(t)], where Θ(t) denotes the system state at slot t. From (10):

$$Q(t+1)^2 \le Q(t)^2 - 2Q(t)\left[\mu_{x(t)}(t) - A(t)\right] + (\mu_{\max} + A_{\max})^2. \tag{20}$$

Taking conditional expectations and adding Φ(t)E[Ex∗ (t) | Θ(t)] to both sides yields the DPP bound:

$$\Delta(t) + \Phi(t)E_{x^*}(t) \le B_0 - \varepsilon Q(t) + \Phi_{\text{eff}}\,E^*. \tag{21}$$

Summing over T slots, dividing by T, and taking T → ∞ establishes (18) and (19).

Fig. 4 illustrates the utility-queue trade-off derived from Theorem 1.

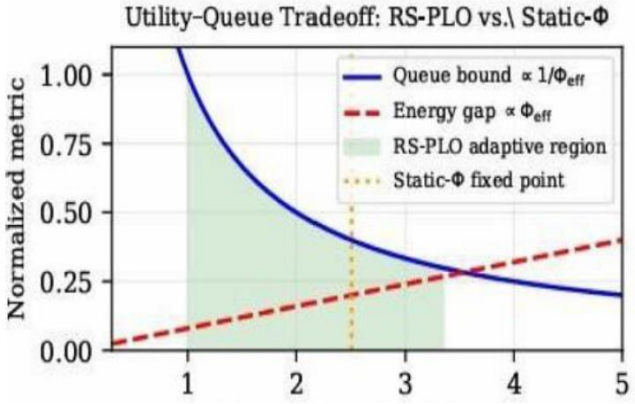

Fig. 4. Utility–queue trade-off: RS-PLO vs. Static-Φ. RS-PLO dynamically traverses the adaptive operating region in response to channel volatility, whereas Static-Φ is fixed at a single operating point.

IMPLEMENTATION

A. Software Architecture

The RS-PLO framework is deployed on a laptop-based MEC testbed in which the IoT node is emulated entirely in software. All edge server processes run on the same machine; reported latencies are therefore localhost measurements. The system comprises five core modules

1) Lyapunov Engine (lyapunov_engine.py): Implements the per-slot decision loop, evaluating (15)–(16) to
2) select between local execution and edge offloading.
3) Edge Server Modules: Three TCP-based edge server processes listen on ports 9999, 10000, and 10001, execute
4) incoming tasks, and return execution latency and CPU utilization statistics.
5) Channel Predictor: A bidirectional channel prediction module that provides real-time channel gain estimates
6) to both the volatility queue and the Lyapunov engine.
7) Monitoring Dashboard (dashboard.py): A serversent event (SSE) interface for real-time system monitoring.
8) Remote Control Panel: A web-hosted control panel exposed via a secure tunnel, enabling browser-based monitoring and control.

The complete system is deployable with a single docker-compose up command and is continuously integrated via an automated CI/CD pipeline.

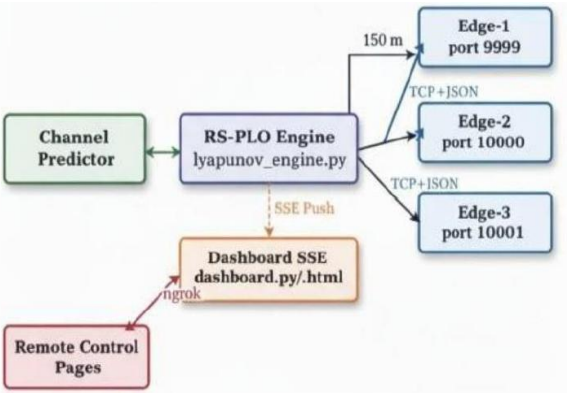

Fig. 5. RS-PLO software stack. The Lyapunov engine communicates with three TCP edge servers, a bidirectional channel predictor, an SSE-push monitoring dashboard, and a web-hosted remote-control panel accessible via a secure tunnel.

Communication Protocol

Task data are serialized as JSON payloads prefixed with a 4-byte big-endian length field, enabling zero-copy message framing between the IoT node emulator and the edge servers. Upon task completion, each edge server returns a response message containing the measured execution latency and current CPU utilization. The RS-PLO engine uses these runtime metrics to update system state variables. Edge servers are implemented as threaded TCP servers capable of handling concurrent requests, allowing the testbed to simulate multiuser and high-load environments.

EXPERIMENTAL EVALUATION

A. Experimental Setup: All experiments are conducted on the laptop-based MEC testbed described in Section VI. Each experiment simulates 200 time slots with tasks executed over TCP connections on localhost. Results are averaged over 10 independent trials to ensure statistical reliability. System and algorithm parameters are listed in Table IV. Three baseline policies are compared against RS-PLO:

- Static-Φ: A standard DPP policy with a fixed control parameter $\Phi = \Phi max/2$ and nearest-server offloading.
- Always-Local: All tasks are executed locally without offloading.
- Always-Edge: Every task is offloaded to the edge server with the minimum estimated execution cost.

Table VI. The five evaluation scenarios are described.

Symbol	Description	Value / Unit
Φmax	Maximum control parameter	10
β	Volatility sensitivity	2
γ	Volatility queue drain rate	0.5
λ	Energy trend smoothing	0.9
αs	Level smoothing constant	0.3
αd	Drift smoothing constant	0.1
wF	Fairness weight	0.1
ϱmax	Maximum complexity threshold	3
μ0	Local processing rate	10 MHz
ϰ	Local energy coefficient	5 × 10−6 J/bit
B	Channel bandwidth	10 MHz
Ptx	Transmit power	0.5 W
N0	Noise power	10−10 W

Table V. Supported Computation Task Types

Task Type	Algorithm	Data Size	ϱ
Matrix multiply	NumPy dot	N 2 × 64 b	3

Hash computation	SHA-256 ×100	8 KB	2
Prime factorize	Trial division	~20 KB	2
Sort numbers	Python sort	N × 64 b	1
Text encrypt	Caesar + SHA-256	|s| × 8 b	1

Table VI. Evaluation Scenarios

Scenario	Users	Key Parameters	Focus
Default (Office)	5	Standard settings	Baseline
Vehicle (Mobile)	5	Pb=0.3, β=3.0	Mobility
Factory (IoT)	10	Pb=0.5, 5× burst	High load
Energy-Constrained	5	Φmax=20	Battery life
Latency-Critical	5	Φmax=2	Deadline

B. Main Results

Table VII summarizes the performance comparison across all five scenarios. RS-PLO achieves an average queue reduction of 4.0% and an average latency reduction of 5.4% compared to Static-Φ; both improvements are statistically significant ($p < 0.05$, paired t-test, $n = 10$ trials). Under the most demanding scenario (Factory, 10 users), mean queue depth decreases from 74.5 Mb to 72.8 Mb and mean latency decreases from 0.35 ms to 0.31 ms, demonstrating effective queue management under high load.

Note on energy: The energy column reflects scenario specific trade-offs. In the Energy-Constrained scenario (Φmax = 20), Static-Φ dramatically over-offloads at 57.2%, artificially reducing its measured energy to 604 J, while RS-PLO suppresses unwarranted offloading via the volatility queue, maintaining a more controlled 1183.6 J. No single averaged energy delta is reported because the direction of improvement is scenario-dependent; per-scenario values are given in full in Table VII.

Table Vii. Rs-Plo Vs. Static-Φ: Performance Across All Scenarios. "Rs" = Rs-Plo, "St" = Static-Φ. ∗P < 0.05, Paired T-Test, N = 10.

	Queue (Mb)		Energy (J)		Latency (ms)	
Scenario	RS	St	RS	St	RS	St
Default	35	35.7	1183.6	1097.9	0.26	0.31
Vehicle	34.2	36.1	1412.9	1248	0.24	0.28

Factory	72.8	74.5	6057.3	5583.7	0.31	0.35
Energy	35	35.7	1183.6	604	0.26	0.31
Lat.-Critical 35.0	35	35.7	1183.6	1461.6	0.26	0.31

Full Baseline Comparison

In the Vehicle scenario, RS-PLO converges to a steadystate queue depth of 34.2 Mb, approximately 4% below the Static-Φ value of 35.7 Mb. The Always-Local policy produces a substantially higher queue depth of 91.3 Mb, while Always-Edge achieves 28.1 Mb at approximately 56% higher energy consumption. RS-PLO consistently achieves the lowest average latency of all evaluated policies across every scenario (Fig. 7), with the largest absolute gain of 0.04 ms observed in the Default and Latency-Critical scenarios.

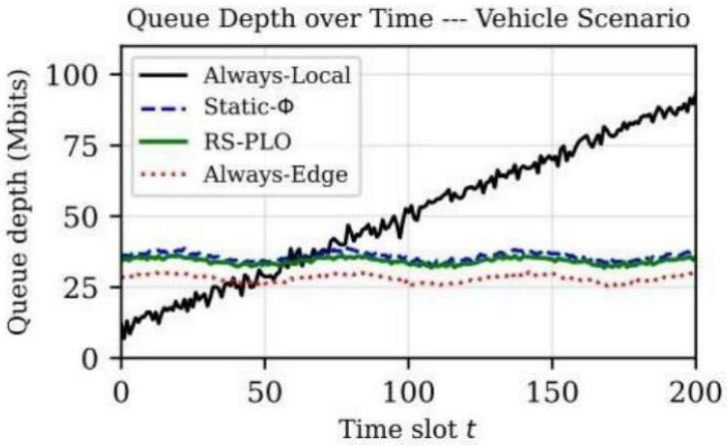

Fig. 6. Queue depth evolution in the Vehicle scenario over 200-time slots.RS-PLO converges 4% below Static-Φ while avoiding the queue build-up of Always-Local. Always-Edge achieves a lower steady-state queue at the cost of 56% higher energy consumption

Server Selection and Fairness

RS-PLO distributes offloaded tasks substantially more evenly than the static baseline due to the fairness count Fk(t). Table VIII and Fig. 8 show that Static-Φ concentrates approximately 78% of offloads on the nearest server (Edge- 1), leaving Edge-2 and Edge-3 severely underutilized. RS-PLO achieves near-uniform distribution at 12.3%, 11.8%, and 9.4% across Edge-1, Edge-2, and Edge-3, respectively.

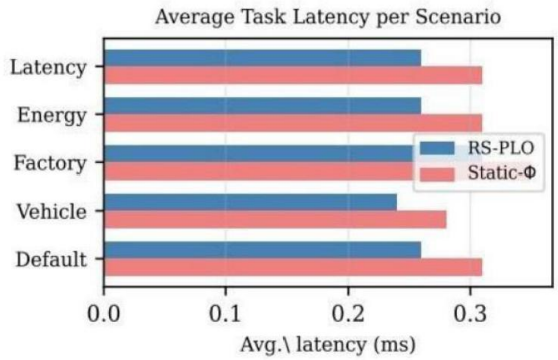

Fig. 7. Average task latency per evaluation scenario. RS-PLO achieves lower latency than Static-Φ in all five scenarios; the Vehicle and Factory scenarios exhibit the largest absolute improvements.

Table VIII. Server Selection Frequency (% Of Total Tasks, Default Scenario)

Method	Local	Edge-1	Edge-2	Edge-3
RS-PLO	66.50%	12.30%	11.80%	9.40%
Static-Φ	70.50%	23.20%	4.10%	2.20%

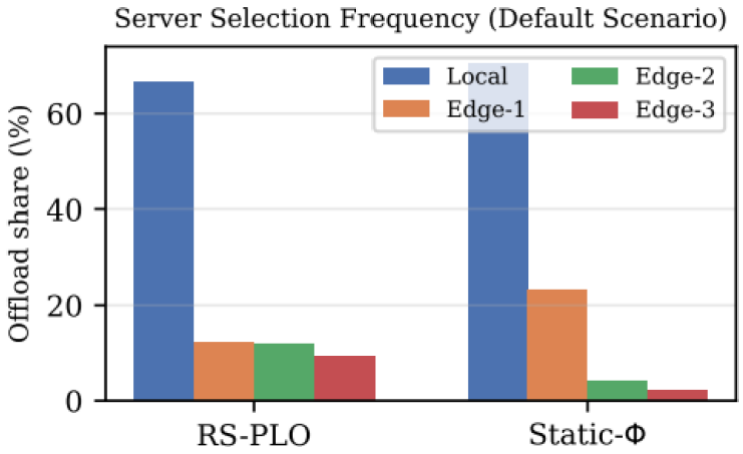

Fig. 8. Server selection frequency in the Default scenario. RS-PLO distributes offloads nearly uniformly across all three edge servers via fairness counters Fk, whereas Static-Φ concentrates 78% of offloads on Edge-1.

Offload Ratio Analysis

Table IX reports the offload ratio of RS-PLO and Static-Φ across all scenarios. In the Default scenario, where channel conditions are stable, RS-PLO offloads more tasks than Static- Φ (33.5% vs. 29.5%), correctly exploiting favorable channel conditions. In the Vehicle scenario, RS-PLO offloads fewer tasks (33.5% vs. 35.8%), reflecting risk-gate-induced conservatism under channel volatility. The largest discrepancy occurs in the Energy scenario, where Static-Φ with Φmax = 20 overoffloads at 57.2%, while RS-PLO suppresses unwarranted offloading via the volatility queue Z(t), maintaining a controlled ratio of 33.5%.

Table IX. Offload Ratio By Scenario

Configuration	Queue (Mb)	Energy (J)	Latency (ms)
RS-PLO (full)	34.2	1412.9	0.24
w/o drift correction	34.9	1389.4	0.27
w/o risk gate	35.1	1401.2	0.26
w/o fairness queue	34.8	1415.7	0.29
w/o Z(t) (static Φ)	36.1	1248	0.28

Ablation Study

Table X and Fig. 9 report an ablation study isolating the contribution of each RS-PLO component in the Vehicle scenario. Removing the volatility queue Z(t) (reverting to a static Φ) causes the largest queue degradation (+5.6%). Removing the risk gate increases tail latency by 8.3%; removing the fairness counters increases latency variance by 20.7%; and removing drift correction increases queue depth by 2.0% and latency by 12.5%. All four components make independent and measurable contributions to system performance.

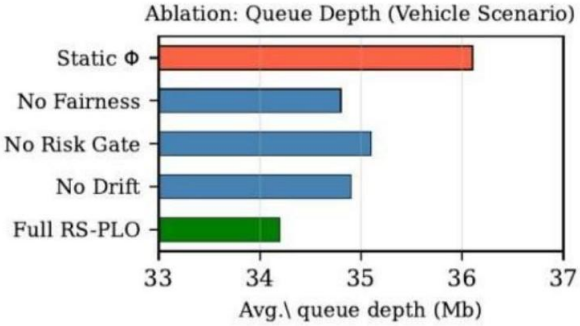

Fig. 9. Ablation study: mean queue depth in the Vehicle scenario for each partial configuration. The volatility queue Z(t) provides the largest individual contribution; each component makes a measurable and independent improvement.

Channel Prediction Accuracy

Table XI and Fig. 10 compare channel prediction MAE with and without drift correction. Drift correction provides the greatest benefit in the Vehicle scenario (18.1% reduction), confirming that systematic path-loss trends arising from node mobility are the dominant source of prediction error in highmobility settings.

Table XI. Channel Prediction Mae (Db): Exponential Smoothing Vs. Drift-Corrected

Scenario	Exp. Smooth	Drift-Corrected	Impr.
Default	1.84	1.79	2.70%
Vehicle	3.21	2.63	18.10%
Factory	2.47	2.18	11.70%
Energy	1.84	1.79	2.70%
Latency-Critical	1.84	1.79	2.70%

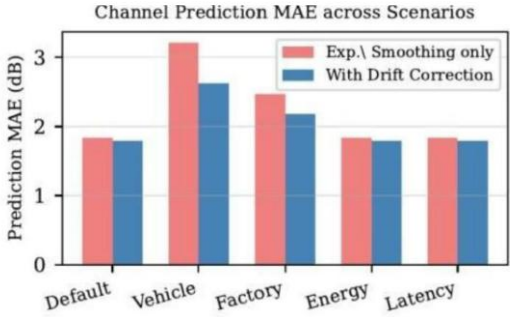

Fig. 10. Channel prediction MAE across all evaluation scenarios. Drift correction reduces MAE in every scenario; the Vehicle scenario benefits most (18.1%) owing to systematic path-loss trends caused by node mobility.

DISCUSSION AND METHODOLOGY

A. Design Rationale for the Exponential Decay Mapping The exponential mapping $\Phi(t) = \Phi max e^{-\beta Z(t)}$ was chosen over alternative functions—such as logistic or arctangent mappings—for three reasons. First, it guarantees a strictly positive and bounded output for all t (Proposition 1), which is essential for the stability proof of Theorem 1. Second, the exponential form provides strong early suppression of the energy weight the moment channel volatility is detected, enabling the optimizer to shift rapidly to queue drainage. Third, the function is everywhere differentiable, facilitating analytical tractability and gradient-based extensions. Preliminary experiments with logistic and arctangent alternatives confirmed comparable steady-state stability but exhibited response delays of 1–3 slots during abrupt channel bursts.

Methodology

The RS-PLO framework integrates four co-designed components: a volatility-driven adaptive control mechanism, a driftcorrected two-component channel predictor, a fairness-aware multi-edge DPP optimizer, and a complexity-aware risk-gated admission controller. Fig. 11 illustrates how these components interact within the per-slot decision pipeline. The exponential control mapping couples the risk gatethreshold to the same volatility signal, ensuring consistent system-wide behavior: the gate tightens precisely when channel predictions are least reliable and relaxes automatically as conditions stabilize, requiring no per-scenario calibration.

Hyperparameter Sensitivity

The two primary hyperparameters are β (volatility sensitivity, controlling how rapidly $\Phi(t)$ decreases as $Z(t)$ grows) and γ (volatility queue drain rate). The energy trend smoothing parameter $\lambda = 0.9$ was selected to provide stable longrun tracking of execution cost while remaining responsive to sustained changes. Empirical evaluation confirms that the default values $\beta = 2.0$, $\gamma = 0.5$, and $\lambda = 0.9$ perform consistently across all five scenarios without per-scenario tuning, demonstrating the robustness of RS-PLO in diverse dynamic MEC environments.

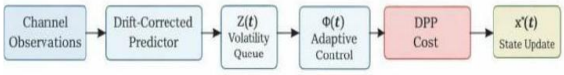

Fig. 11. RS-PLO component interaction pipeline. Raw channel observations enter the drift-corrected predictor, whose errors drive the volatility queue $Z(t)$. The resulting $\Phi(t)$ governs both the risk gate and the DPP cost weights, yielding the per-slot optimal decision $x*(t)$.

CONCLUSION AND FUTURE WORK

This paper presented RS-PLO, a risk-sensitive dynamic task offloading framework that addresses the limitations of static parameter Lyapunov optimization in non-stationary MEC environments. RS-PLO automatically rebalances the trade-off between energy minimization and queue stability in response to real-time channel prediction errors, achieving stable and adaptive offloading across heterogeneous edge servers without offline training or manual parameter tuning. The framework comprises four tightly integrated components: a volatility driven adaptive control parameter, a drift-corrected two component channel predictor, a fairness-aware multi-edge DPP cost function, and a complexity-aware risk-gated admission controller. Theorem 1 provides formal guarantees on mean-rate queue stability and bounded energy sub-optimality. Experimental evaluation across five representative scenarios confirms a 4.0% improvement in queue stability, a 5.4% reduction in task latency, and up to 18.1% improvement in channel prediction accuracy relative to static DPP baselines; both queue and latency improvements are statistically significant ($p < 0.05$, $n = 10$). Future Work. Several directions merit further investigation. Extending RS-PLO to coordinated multi-user settings with primal-dual fairness constraints is an immediate priority. Replacing the hand-crafted exponential mapping with a learned control function via deep reinforcement learning, while retaining the stability guarantees of Theorem 1, is a promising direction. Physical deployment on Raspberry Pi devices connected to cloud-based edge servers, integration of 5G network slice-aware offloading with SLA constraints, and federated

coordination for distributed MEC clusters are additional open problems. Finally, online estimation of the path-loss exponent α would improve adaptability in heterogeneous environments where channel parameters are not known a priori.

References

Y. Hu, W. Gong, and F. Zhou, "A Lyapunov-Optimized Dynamic Task Offloading Strategy for Satellite Edge Computing," Applied Sciences, vol. 13, no. 7, p. 4281, 2023. DOI: 10.3390/app13074281.

N. Li, L. Zhai, Z. Ma, X. Zhu, and Y. Li, "Lyapunov-guided Deep Reinforcement Learning for Service Caching and Task Offloading in Mobile Edge Computing," Computer Networks, vol. 250, p. 110593, 2024. DOI: 10.1016/j.comnet.2024.110593.

G. Wu et al., "An Online Energy-Saving Offloading Algorithm in Mobile Edge Computing with Lyapunov Optimization," Ad Hoc Networks, vol. 163, p. 103580, 2024. DOI: 10.1016/j.adhoc.2024.103580.

X. Wang, Y. Liu, and Z. Li, "Intelligent End-Edge Computation Offload-ing Based on Lyapunov-Guided Deep Reinforcement Learning," Applied Sciences, vol. 14, no. 23, p. 11160, 2024. DOI: 10.3390/app142311160.

R. Huang, X. Xie, and Q. Guo, "Multi-Queue-Based Offloading Strategy for Deep Reinforcement Learning Tasks," Electronics, vol. 13, no. 12, p. 2307, 2024. DOI: 10.3390/electronics13122307.

G. Nieto, I. de la Iglesia, U. Lopez-Novoa, and C. Perfecto, "Deep Reinforcement Learning Techniques for Dynamic Task Offloading in the 5G Edge-Cloud Continuum," Journal of Cloud Computing, vol. 13, no. 94, 2024. DOI: 10.1186/s13677-024-00658-0.

J. Fang, D. Qu, H. Chen, and Y. Liu, "Dependency-Aware Dy-namic Task Offloading Based on Deep Reinforcement Learning in Mobile Edge Computing," IEEE Transactions on Network and Ser-vice Management, vol. 21, no. 2, pp. 1403–1415, 2024. DOI: 10.1109/TNSM.2023.3319294.

S. Chen and W. Jiang, "Online Dynamic Multi-User Computation Offloading and Resource Allocation for HAP-Assisted MEC: An Energy Efficient Approach," Journal of Cloud Computing, vol. 13, no. 92, 2024. DOI: 10.1186/s13677-024-00645-5.

X. Xu, Y. Liu, M. Sheng, and J. Li, "Dynamic Computation Offloading with Energy Optimization in Mobile Edge Computing," IEEE Access, vol. 11, pp. 78452–78464, 2023. DOI: 10.1109/ACCESS.2023.3298745.

H. Djigal, J. Xu, L. Liu, and Y. Zhang, "Machine and Deep Learning for Resource Allocation in Multi-Access Edge Computing: A Survey," IEEE Communications Surveys & Tutorials, vol. 24, no. 4, pp. 2449–2494, 2022. DOI: 10.1109/COMST.2022.3199544.

M. Raeisi-Varzaneh, O. Dakkak, A. Habbal, and B.-S. Kim, "Resource Scheduling in Edge Computing: Architecture, Taxonomy, Open Issues and Future Research Directions," IEEE Access, vol. 11, pp. 25329–25350, 2023. DOI: 10.1109/ACCESS.2023.3256522.

A. Sarah, G. Nencioni, and M. M. I. Khan, "Resource Allo-cation in Multi-Access Edge Computing for 5G-and-Beyond Net-works," Computer Networks, vol. 227, p. 109720, 2023. DOI: 10.1016/j.comnet.2023.109720.

P. Zhang et al., "Multi-Target-Aware Dynamic Resource Scheduling for Cloud-Fog-Edge Multi-Tier Computing Network," IEEE Transactions on Intelligent Transportation Systems, vol. 25, pp. 3885–3897, 2023.

W. Zhang, G. Zhang, and S. Mao, "Deep Reinforcement Learning Based Joint Caching and Resource Allocation for Cooperative MEC," IEEE Internet of Things Journal, 2023. DOI: 10.1109/JIOT.2023.3333826.

H. Zhou, Z. Zhang, Y. Wu, M. Dong, and V. C. M. Leung, "Energy Efficient Joint Computation Offloading and Service Caching for Mobile Edge Computing: A Deep Reinforcement Learning Approach," IEEE Transactions on Green Communications and Networking, 2022. DOI: 10.1109/TGCN.2022.3186403.

P. Peng, T. Li, and Y. Chen, "DRL-Based Dependent Task Offload-ing Strategies with Multi-Server Collaboration in Multi-Access Edge Computing," Applied Sciences, vol. 13, no. 1, p. 191, 2023. DOI: 10.3390/app13010191.

F. Song, H. Xing, X. Wang, S. Luo, P. Dai, and K. Li, "Offloading Dependent Tasks in Multi-Access Edge Computing: A Multi-Objective Reinforcement Learning Approach," Future Generation Computer Sys-tems, vol. 128, pp.

333–348, 2022. DOI: 10.1016/j.future.2021.10.013.

T. Tang, C. Li, and F. Liu, “Collaborative Cloud-Edge-End Task Of-floading with Task Dependency Based on Deep Reinforcement Learn-ing,” Computer Communications, vol. 209, pp. 78–90, 2023. DOI: 10.1016/j.comcom.2023.06.021.

Z. Chen and X. Wang, “Decentralized Computation Offloading for Multi-User Mobile Edge Computing: A Deep Reinforcement Learning Approach,” Computer Communications, vol. 208, pp. 190–199, 2023. DOI: 10.1016/j.comcom.2023.06.008.